

Пермский филиал федерального государственного
автономного образовательного учреждения высшего
образования

«Национальный исследовательский университет
«Высшая школа экономики»

Факультет социально-экономических и компьютерных наук

Некрасов Александр Юрьевич

**РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ЖЕРЕБЬЁВКИ ШАХМАТНЫХ
ТУРНИРОВ**

Курсовая работа

студента образовательной программы «Программная
инженерия» по направлению подготовки 09.03.04 Программная
инженерия

Руководитель
к.т.н., доцент кафедры ин-
формационных технологий в
бизнесе НИУ ВШЭ – Пермь

О. Л. Викентьева

Пермь, 2023

Аннотация

Курсовая работа посвящена разработке программы по организации и проведении жеребьевки шахматных турниров.

Отчёт содержит следующие главы:

1. Анализ задачи.
2. Проектирование информационной системы.
3. Реализация информационной системы.
4. Тестирование информационной системы.

Отчёт содержит 135 страниц формата А4 основного текста, не считая аннотацию, оглавление, введение, заключение, библиографического списка и приложений, содержащих также техническую документацию.

В отчёте содержится 142 иллюстрации и 24 таблицы, демонстрирующие работу программы, описание использованных при разработке алгоритмов, примерами тестирования программы.

Библиографический список состоит из 9 источников.

Оглавление

Глава 1 Анализ задачи.....	6
1.1 Анализ бизнес-процесса проведения турнира	6
1.2 Анализ недостатков текущего бизнес-процесса	7
1.3 Обзор существующих решений.....	8
1.4 Формирование функциональных требований.....	12
1.5 Формирование нефункциональных требований.....	13
1.6 Объекты и атрибуты	13
Глава 2 Проектирование информационной системы.....	15
2.1 Описание прецедентов «to-be»	16
2.2 Описание активностей процесса «to-be»	27
2.3 Проектирование базы данных	31
2.4 Проектирование информационной системы.....	47
Глава 3 Реализация информационной системы.....	76
3.1 Реализация взаимодействия классов модуля интерфейса	76
3.2 Реализация алгоритмов модуля бизнес-логики	87
3.3 Реализация запросов LINQ модуля доступа к данным	89
Глава 4 Тестирование информационной системы	98
4.1 Компонентное тестирование.....	98
4.2 Интеграционное тестирование	131
4.3 Системное тестирование	136
Заключение.....	141
Библиографический список.....	142
ПРИЛОЖЕНИЕ А Терминология	143
ПРИЛОЖЕНИЕ Б Техническое задание	144
ПРИЛОЖЕНИЕ В Руководство оператора	162
ПРИЛОЖЕНИЕ Г Требования к данным	173
ПРИЛОЖЕНИЕ Д Листинги встроенных процедур	180
ПРИЛОЖЕНИЕ Е Листинги триггеров	185
ПРИЛОЖЕНИЕ Ж Листинги представлений.....	187

Введение

В современном мире очные шахматы [1] не утрачивают своей ценности, несмотря на развитие технологий, появление онлайн-турниров.

Так как на соревнования приходит большое количество людей, становится не рационально использовать бумажные варианты жеребьевок или варианты, проводимые с помощью электронных таблиц.

Далее приведена таблица количества турниров, проведенных в России за период с 2018 по 2022 годы (см. таблицу 1.1).

Таблица 1.1 – Количество проведенных турниров

Год проведения	Количество турниров
2018	18 467
2019	22 891
2020	9 424
2021	19 666
2022	25 674

Как видно из таблицы, количество турниров по шахматам, проводимых в России, растет, однако, заметно падение в 2020 и 2021 годах, связанное с пандемией, но в 2022 году было проведено гораздо большее количество турниров в сравнении с годами до пандемии, что говорит о том, что популярность шахматных турниров будет продолжать расти.

Из данных можно сделать вывод, что автоматизировать процесс жеребьевки необходимо, так как с ростом популярности турниров, все большее количество людей будут принимать участие в них, тем труднее вести учет игр вручную в бумажном виде или в электронных таблицах, тем более востребованными будут программы для проведения турниров.

Объектом автоматизации является процесс проведения жеребьевки турнира.

Предметом автоматизации является разработка приложения для организации турниров.

Задачи, необходимые для реализации приложения для автоматизации организации турнира:

1. Проанализировать предметную область, выявить проблемные места текущих бизнес-процессов.
2. Проанализировать существующие решения, выявить недостатки и положительные стороны.
3. Разработать функциональные и нефункциональные требования к системе.
4. Спроектировать и реализовать информационную систему.
5. Провести тестирование системы.
6. Подготовить программную документацию в соответствии с ГОСТ 19 (техническое задание, руководство пользователя).

При выполнении задач информационная система должна быть способна обеспечить автоматизацию жеребьевки турниров, подведения результатов, как результат, упрощение и ускорение процесса организации, в этом заключается практическая значимость системы.

Глава 1 Анализ задачи

1.1 Анализ бизнес-процесса проведения турнира

Текстовое описание бизнес-процесса по организации турнира представлено далее.

Организатор турнира перед его началом собирает список участников, запись происходит либо в бумажном, либо в электронном виде.

После того, как информация об участниках собрана, происходит построение таблицы встреч игроков. Далее пользователь строит таблицу в электронной таблице или в бумажном варианте, записывая игроков в первый столбец и первую строку, создавая тем самым перекрестную таблицу (см. рисунок 1.1).

		1	2	3	4	5	6	7	8	○	M
1											
2											
3											
4											
5											
6											
7											
8											

Рисунок 1.1 – Пример перекрестной таблицы

Непосредственное проведение жеребьевки заключается в отслеживании в таблице игроков, которые не играли друг с другом, а также в вводе результатов встреч в ячейки, соответствующие пересечению игроков в таблице.

Необходимо также учитывать то, что каждый игрок должен сыграть примерно одинаковое количество партий как первым (за белых), так и вторым (за черных).

Также если турнир командный или содержит несколько групп и необходимо не допустить встреч между участниками одной команды или разных групп, то организатору также необходимо разделить игроков по разным таблицам.

После того, как все игры проведены, для каждого игрока подсчитывается сумма очков, набранных им, формируется рейтинг.

Если существуют игроки с одинаковым количеством очков, то для их распределения обычно организуются дополнительные игры, или результат личной встречи.

1.2 Анализ недостатков текущего бизнес-процесса

Исходя из приведенного описания бизнес-процесса, можно выделить следующие его недостатки:

1. Необходимость в самостоятельной организации и построения перекрестной таблицы.
2. Необходимость самостоятельного выбора пар игроков и отслеживания чередования цвета игроков.
3. Если присутствуют ограничения на встречи игроков (запрет на игры между игроками одной команды, группы), то необходимо строить дополнительные таблицы.
4. Необходимость решения проблем, связанных с одинаковым количеством игроков.

В условиях проведения турнира – большое количество людей, шум, нервная обстановка – тяжело проводить ручное проведение жеребьевки – постоянные отвлечения и прочие обстоятельства могут привести к тому, что процесс жеребьевки нарушается, например, при некорректном заполнении результата встречи, результаты всего турнира становятся некорректными.

Далее приведена диаграмма прецедентов бизнес-процесса as-is (как есть) (см. рисунок 1.2).



1.3 Обзор существующих решений

Так как турниры по шахматам проводятся значительно давно, за это время было создано множество решений, среди которых большинство иностранных лицензионных программ, таких как Swiss Manager, SwissChess.

1.3.1 Обзор ChessResults.ru

Среди решений, созданных в России, был найден сайт ChessResults.ru, позволяющий провести жеребьевку турнира.

Однако для пользования сайтом необходимо оплачивать тариф за каждый проведенный турнир (см. рисунок 1.3).

Среди возможностей сайта включена предварительная регистрация участников до начала турнира, что удобно, так как позволяет сэкономить время на заполнение списка участников перед непосредственным началом турнира. Также присутствует возможность обсчета рейтинга РШТ, что полезно тем организаторам, которые имеют аккредитацию на присвоение рейтинга и разрядов.

Таким образом, приложение позволяет провести жеребьевку турниров, организуя при этом возможность самостоятельной регистрации участников и обсчета рейтинга, однако плата за проведение каждого турнира может отпугнуть многих пользователей.

Тарифы и стоимость

Программа платная. Тарификация за пользование программой происходит исключительно по каждому проведённому турниру без какой-либо дополнительной абонентской платы. Расчёт стоимости турнира происходит автоматически и зависит от выбранного тарифа.

Действует два тарифа: Коммерческий и Бюджетный (для просмотра описания тарифа необходимо нажать на его название ниже).

▼ Коммерческий — для турниров с денежными взносами от 501₽	50₽/участник
▲ Бюджетный — для турниров бесплатных или с денежными взносами до 500₽	100₽/платеж

Включает:

- ✓ Форма предварительной регистрации участников
- ✓ Обсчёт отчётных турниров на рейтинг РШТ

Условия:

- ✓ Стоимость за турнир(ы), рассчитывается путём умножения фактического количества участников на количество туров с учётом бюджетной скидки в 50% и итоговая сумма вычитается из имеющейся на Балансе.
- ✓ Минимальный платёж — 100₽ или кратный 100₽. Сумма добавляется к Балансу.
- ✓ Пример, турнир с 15 участниками в 7 туров тарифицируется как: 15 (игроков) * 7 (туров) * 50% (бюджетная скидка) = 53₽.

Рисунок 1.3 – Тарифы приложения ChessResults.ru

1.3.2 Обзор Swiss Manager

Программа, доступная на многих языках, включая русский, позволяет проводить турниры по швейцарской системе [2], круговой [3], также возможны командные соревнования.

Поддерживает вывод данных в EXCEL, txt, HTML, печать. Также возможно проведение турниров с участниками до 2000 с 23 турами максимально по швейцарской системе, и 1500 участников с 150 турами максимально в круговой системе.

Доступна онлайн база турниров с возможностью просмотра информации о турнирах, участниках и результатах.

Доступна на операционных системах Windows 7 и 10.

Полная версия доступна по оплате за 150 евро, облегченная версия (60 участников и 11 туров) доступна за 75 евро.

Таким образом, данная программа подходит для проведения масштабных соревнования государственного и краевого уровня, среднестатистическому тренеру секции не подходит ввиду перегруженности возможностей и высокой стоимости.

1.3.3 Обзор SwissChess

Программа доступна на немецком, английском и русском языках. Позволяет проводить турниры по круговой системе, швейцарской с ее разновидностями и другими системами. Возможно проведение командных соревнований.

Максимальное число участников – 988, число туров – 25.

Для проведения групповых турниров предполагается запускать программу несколько раз для создания нового турнира, что может быть неудобным из-за необходимости в переключении между турнирами.

Также немаловажным минусом является то, что некоторые ошибки в программе могут быть на немецком языке, хотя основной язык – русский.

Для неопытного пользователя обилие настроек может быть отпугивающим фактором, так как непонятно, что делать дальше — меню программы содержит множество подпунктов (см. рисунок 1.4).

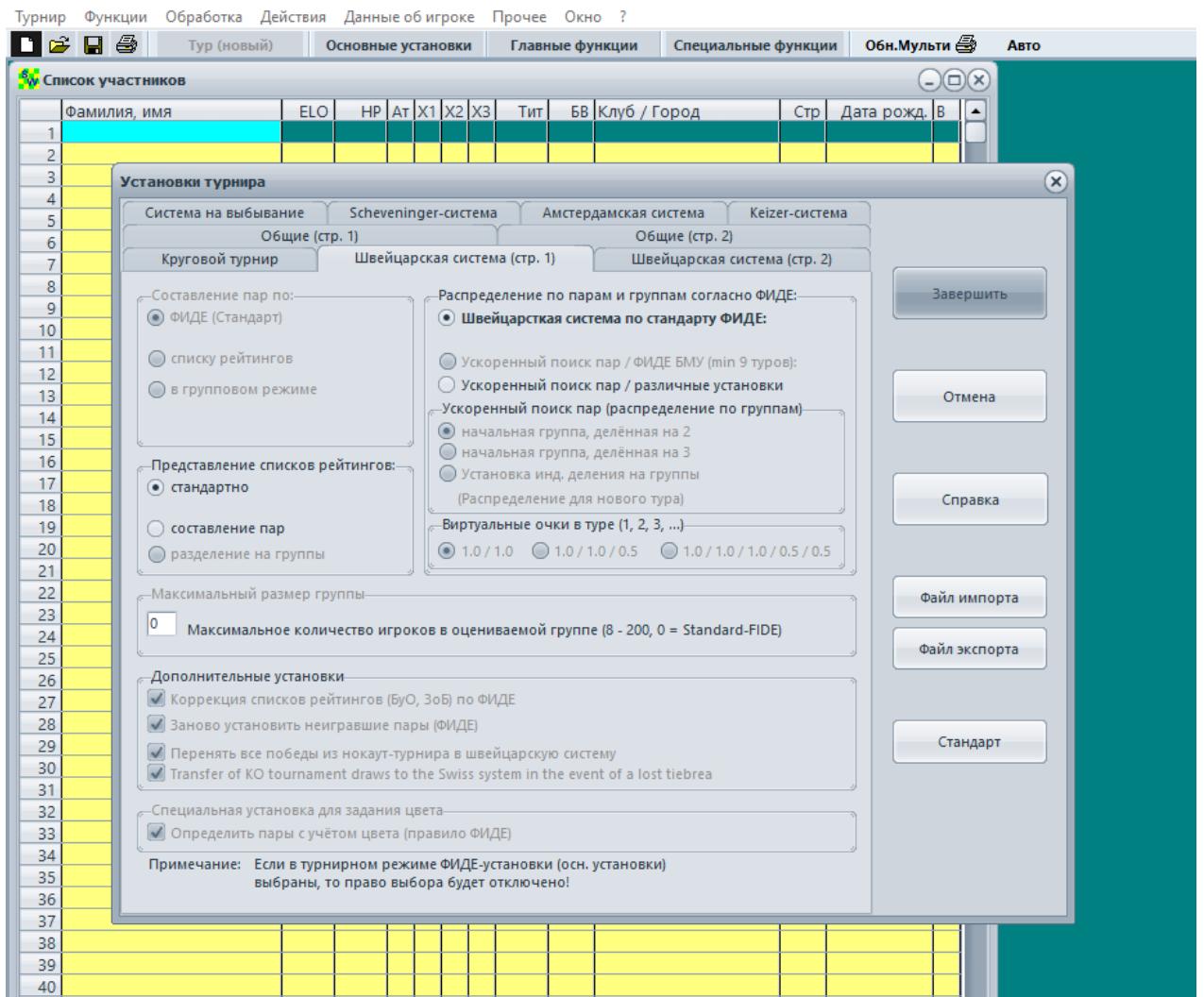


Рисунок 1.4 – Пример интерфейса программы SwissChess

Полная версия программы доступна за 4900 рублей, также доступна демоверсия, но с ограничениями: максимальное число туров – 4, максимальное число участников – 20.

1.3.4 Сравнение решений

Далее приведем сравнение описанных ранее решений по следующим критериям:

1. Локализация.
2. Поддерживаемые операционные системы (ОС).
3. Организация командных турниров.
4. Разделение игроков по группам.
5. Стоимость.

Таблица 1.1 – Сравнение программных решений

Решение	Локализация	Поддерживаемые ОС	Организация командных турниров	Разделение игроков по группам	Стоимость
ChessResults.Ru.	Русский, английский.	Любая ОС, поддерживающая запуск интернет-браузера.	Присутствует	Присутствует	Тарифный план на турнир
SwissManager	24 языка, включая русский	Windows 7, Windows 8, Windows 10, Windows 11	Присутствует	Присутствует	75 евро – облегченная версия, 150 евро – полная
SwissChess	Русский, английский, немецкий	Windows 7, Windows 8, Windows 10, Windows 11	Присутствует	Отсутствует	4900 руб.

Таким образом, данные программы позволяют организовывать турниры, однако стоимость каждого из решений может оттолкнуть потенциального пользователя.

1.4 Формирование функциональных требований

Исходя из анализа бизнес-процесса организации турнира, а также его недостатков и обзора существующих решений, были сформулированы следующие функциональные требования (см. таблицу 1.2).

Источники требований определены на диаграмме прецедентов бизнес-процесса «as-is» (см. рисунок 1.2).

Таблица 1.2 – Функциональные требования

Требование	Источник
1. Возможность ввода участников турнира в турнирные списки, их редактирование, удаление.	
2. Возможность просмотра списков турниров, их редактирование, создание, удаление.	1. Сбор информации об участниках.
3. Возможность импорта списка участников из файла (csv).	
4. Возможность выбора вида турнира (командный, личный, лично-командный).	
5. Возможность ввода команд игроков, если турнир командный или лично-командный.	2. Разделение на группы и команды.
6. Возможность разделения игроков на группы.	
7. Создание алгоритма жеребьевки.	3. Организация встреч. 4. Контроль чередования цвета игроков.
8. Возможность экспорта таблиц (xlsx, pdf).	5. Построение перекрестных таблиц.
9. Возможность заполнения результатов встречи.	
10. Возможность редактирования результатов сыгранных партий.	6. Запись результатов встречи при сыгранной партии.
11. Осуществление автоматического подсчёта очков игроков.	7. Формирование рейтинга.
12. Создание коэффициентов ранжирования игроков.	8. Проведение дополнительных встреч при неоднозначном рейтинге.

1.5 Формирование нефункциональных требований

Для обеспечения удобства использования программой, она должна отвечать следующим нефункциональным требованиям:

- программа должна обеспечивать бесперебойную работу на протяжении всего процесса организации турнира;
- программа должна предоставлять пользователю возможность регистрации и входа в систему;
- программа должна выводить информацию о неверных действиях пользователя и предлагать решение проблемы;
- программа должна обеспечивать бесперебойный доступ к данным при наличии интернет-соединения;
- программа должна обеспечивать автоматическое сохранение данных;
- программа не должна требовать объем оперативной памяти в размере более 500 мегабайт;
- программа не должна занимать более 1 гигабайта дискового пространства.

1.6 Объекты и атрибуты

На основании описания бизнес-процесса предметной области и функциональных требований, можно выделить следующие объекты и их атрибуты. (см. таблицу 1.3).

Таблица 1.3 – Объекты и атрибуты

Объект	Атрибуты
ПОЛЬЗОВАТЕЛЬ (организатор)	1) ID пользователя; 2) электронная почта; 3) ФИО пользователя; 4) пароль; 5) организуемые турниры.
ТУРНИР	1) ID турнира; 2) название турнира; 3) даты начала и окончания; 4) информация об организаторе и судьях; 5) система турнира; 6) вид турнира; 7) количество туров.

Объект	Атрибуты
ИГРОК	1) ID игрока; 2) ФИО игрока; 3) пол; 4) атрибут; 5) команда; 6) год рождения; 7) статус.
СПИСОК ПАР	1) номер пары; 2) номер тура; 3) пары игроков; 4) краткая информация о каждом игроке; 5) результаты встреч игроков.
КОМАНДА	1) ID команды; 2) название команды; 3) информация о каждом игроке; 4) атрибут команды; 5) статус команды.
ГРУППА	1) ID группы; 2) полное название группы; 3) краткий идентификатор группы.

Глава 2 Проектирование информационной системы

В результате анализа предметной области, т.е. существующего процесса и используемых программных решений для жеребьевки турниров, были сформулированы требования к системе жеребьевки и спроектирован бизнес-процесс описанный ниже.

Пользователь системы во время регистрации с помощью логина (электронная почта) и пароля автоматически становится организатором турниров, с уровнем прав доступа, позволяющего управлять созданными турнирами, а также максимально возможным количеством создаваемых турниров.

Во время создания турнира Организатор вводит уникальное название турнира, количество туров, максимальное количество учитываемых игроков в команде (если турнир предполагает наличие команд), место проведения турнира, дата и время начала, место проведения, длительность (в часах). Также, при создании турнира, Организатор должен выбрать Систему турнира (круговая или швейцарская), Вид турнира (личный, командный, лично-командный), Коэффициенты [4].

Для Турнира Организатор определяет список Игроков. О каждом Игроке известно его имя, фамилия, атрибут (короткая строка, который может использоваться в качестве разделения игроков на группы), пол. Во время прохождения Турнира, игроки играют друг с другомарами, при этом в каждом турнире игроки встречаются друг с другом ровно 1 раз, определяется результат встречи, исходя из которого игрокам будут присуждены очки, обновятся их коэффициенты.

Игроки могут состоять в Командах (если это предполагается турниром), которые определяются во время организации Турнира. Для каждой команды известно её уникальное название, её статус (активная или неактивная), атрибут.

2.1 Описание прецедентов «to-be»

Исходя из функциональных требований к системе, описанных в главе 1, были определены следующие варианты использования (см. рисунки 2.1-2.3).

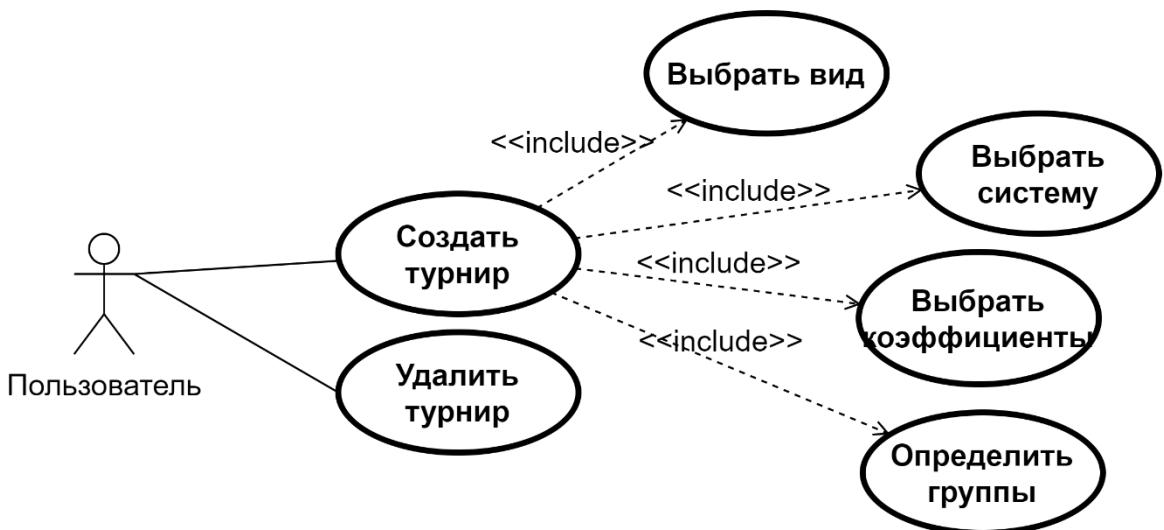


Рисунок 2.1 – Описание прецедентов взаимодействия пользователя с системой (1 часть)

Описание прецедентов взаимодействия пользователя с системой, связанные с созданием и удалением турниров приведены далее.

Название: Создание турнира.

Авторы: Пользователь.

Краткое описание: При создании турнира пользователю предлагается заполнить данные о виде турнира (личный, командный, лично-командный), системе (круповая, швейцарская), коэффициентах, также предлагается определить группы игроков с определением возможности смешения групп или строго раздельной жеребьевки.

Триггер: Нажатие кнопки создания турнира.

Основной поток (таблица 2.1):

Таблица 2.1 – Прецедент "Создание турнира"

Действия акторов	Отклик системы
1. Пользователь нажимает на кнопку создания турнира.	2. Система предоставляет форму заполнения данных о турнире.
3. Пользователь вводит данные.	4. Проверка корректности введенных данных, если данные корректны, то пользователю открывается окно управления турниром (E1).

Альтернативные потоки:

E1: Если данные введены некорректно, пользователю выводится ошибка с предложением ввести данные заново.

Название: Удаление турнира.

Авторы: Пользователь.

Краткое описание: При необходимости удаления существующего турнира пользователь, находясь в главном окне со списком своих турниров нажимает кнопку удаления турнира напротив соответствующего турнира.

Триггер: Запуск меню создания турнира.

Основной поток (таблица 2.2):

Таблица 2.2 – Прецедент "Удаление турнира"

Действия акторов	Отклик системы
1. Пользователь нажимает на кнопку удаления турнира.	2. Система возвращает диалоговое окно с подтверждением удаления.
3. Пользователь подтверждает удаление (E1).	4. Система удаляет турнир из списка.

Альтернативные потоки:

E1: Если пользователь в диалоговом окне подтверждения решает отменить удаление, процесс удаление прерывается, турнир остается неизменным.



Рисунок 2.2 – Описание прецедентов взаимодействия пользователя с системой (2 часть)

Описание прецедентов, связанных с работой с конкретным турниром приведено далее.

Название: Открытие турнира.

Авторы: Пользователь.

Краткое описание: При необходимости работы с турниром (редактирование, проведение жеребьевки) пользователь, находясь в главном окне со списком доступных ему турниров, выбрав нужный, открывает его, система при этом перенаправляет его на открывшееся окно для работы с данным турниром.

Триггер: Нажатие на название турнира.

Основной поток (таблица 2.3):

Таблица 2.3 – Прецедент "Открытие турнира"

Действия акторов	Отклик системы
1. Пользователь нажимает на турнир из списка.	2. Отображение окна турнира.

Название: Добавление участника.

Авторы: Пользователь.

Краткое описание: При необходимости добавления участника в список, пользователь может начать заполнять данные в таблицу списка участников, участник

автоматически добавится, если все указанные данные корректны. Независимо от того, началась жеребьевка, или нет, добавление участников возможно.

Триггер: Начало добавления данных в список участников.

Основной поток (таблица 2.4):

Таблица 2.4 – Прецедент "Добавление участника"

Действия акторов	Отклик системы
1. Пользователь открывает турнир из списка турниров.	2. Отображение окна турнира.
3. Пользователь заполняет данные в таблицу.	4. Если строка об участнике заполнена полностью, то участник добавляется (Е1).

Альтернативные потоки:

Е1: Если строка данных заполнена не полностью, или в ней присутствуют ошибки, то строка в таблице с ошибочными данными выделяется красным цветом, данные не заносятся в базу.

Название: Редактирование участника.

Авторы: Пользователь.

Краткое описание: При необходимости редактирования данных об участнике, пользователь может так же, как и при добавлении участника, изменить данные в ячейках таблицы, соответствующие участнику. Редактирование участников возможно вне зависимости от начала жеребьевки турнира.

Триггер: Начало изменения данных об участнике в списке.

Основной поток (таблица 2.5):

Таблица 2.5 – Прецедент "Редактирование участника"

Действия акторов	Отклик системы
1. Пользователь открывает турнир из списка турниров.	2. Отображение окна турнира.
3. Пользователь изменяет данные в таблице списка участников.	4. Если строка об участнике заполнена полностью, то участник добавляется (Е1).

Альтернативные потоки:

Е1: Если строка данных заполнена не полностью, или в ней присутствуют ошибки, то строка в таблице с ошибочными данными выделяется красным цветом, данные не обновляются в базе.

Название: Сделать участника неактивным.

Авторы: Пользователь.

Краткое описание: При необходимости исключения участника из жеребьевки ввиду его отсутствия во время турнира, пользователь может сделать его неактивным.

Триггер: Переключение статуса участника о его активности.

Основной поток (таблица 2.6):

Таблица 2.6 – Прецедент "Сделать участника неактивным"

Действия акторов	Отклик системы
1. Пользователь открывает турнир из списка турниров.	2. Отображение окна турнира.
3. Пользователь переключает статус участника кнопкой в строке таблицы об участнике.	4. Стока таблицы с участником выделяется цветом, участник исключается из жеребьевки следующих туров (E1).

Альтернативные потоки:

E1: Если жеребьевка текущего тура уже идет, то исключение участника начнет действовать после ее окончания.

Название: Удаление участника.

Авторы: Пользователь.

Краткое описание: Пользователь может удалить участника из турнира, если тот не играет ни в одной игре.

Триггер: Нажатие на кнопку удаления участника.

Основной поток (таблица 2.7):

Таблица 2.7 – Прецедент "Удаление участника"

Действия акторов	Отклик системы
1. Пользователь открывает турнир из списка турниров.	2. Отображение окна турнира.
3. Пользователь нажимает на кнопку удаления участника в строке таблицы этого участника (E1).	4. Удаление участника.

Альтернативные потоки:

E1: Если участник принимает участие в игре, то данная функция будет недоступна.

Название: Добавление команды.

Авторы: Пользователь.

Краткое описание: При необходимости добавления команды в список, пользователь может начать заполнять данные в таблицу списка команд, команда автоматически добавится, если все указанные данные корректны. Независимо от того, началась жеребьевка, или нет, добавление команд возможно.

Триггер: Начало добавления данных в список команд.

Основной поток (таблица 2.8):

Таблица 2.8 – Прецедент "Добавление команды"

Действия акторов	Отклик системы
1. Пользователь открывает турнир из списка турниров.	2. Отображение окна турнира.
3. Пользователь открывает вкладку со списком команд.	4. Отображение окна со списком команд.
5. Пользователь заполняет данные в таблицу.	6. Если строка о команде заполнена полностью, то команда добавляется (E1).

Альтернативные потоки:

E1: Если строка данных заполнена не полностью, или в ней присутствуют ошибки, то строка в таблице с ошибочными данными выделяется красным цветом, данные не заносятся в базу.

Название: Редактирование команды.

Авторы: Пользователь.

Краткое описание: При необходимости редактирования данных о команде, пользователь может так же, как и при добавлении команды, изменить данные в ячейках таблицы, соответствующие команде. Редактирование команды возможно вне зависимости от начала жеребьевки турнира.

Триггер: Начало изменения данных о команде в списке.

Основной поток (таблица 2.9):

Таблица 2.9 – Прецедент "Редактирование команды"

Действия акторов	Отклик системы
1. Пользователь открывает турнир из списка турниров.	2. Отображение окна турнира.

Действия акторов	Отклик системы
3. Пользователь открывает вкладку со списком команд.	4. Отображение окна со списком команд.
5. Пользователь изменяет данные в таблице списка команд.	6. Если строка о команде заполнена полностью, то команда добавляется (E1).

Альтернативные потоки:

E1: Если строка данных заполнена не полностью, или в ней присутствуют ошибки, то строка в таблице с ошибочными данными выделяется красным цветом, данные не обновляются в базе.

Название: Удаление команды.

Авторы: Пользователь.

Краткое описание: При необходимости удалить команду до начала турнира, ввиду его известного отсутствия, либо ошибки ввода, пользователь может удалить строку в таблице списка участников, тем самым удалить и саму команду. Удаление команды возможно только если в ней отсутствуют игроки.

Триггер: Нажатие на кнопку удаления команды.

Основной поток (таблица 2.10):

Таблица 2.10 – Прецедент "Удаление команды"

Действия акторов	Отклик системы
1. Пользователь открывает турнир из списка турниров.	2. Отображение окна турнира.
3. Пользователь открывает вкладку со списком команд.	4. Отображение окна со списком команд.
5. Пользователь нажимает на кнопку удаления команды (E1).	6. Команда удаляется.

Альтернативные потоки:

E1: Если команда содержит участников, то кнопка будет недоступна.

Прецеденты, связанные с проведением жеребьевки турнира и подведения итогов приведены далее.

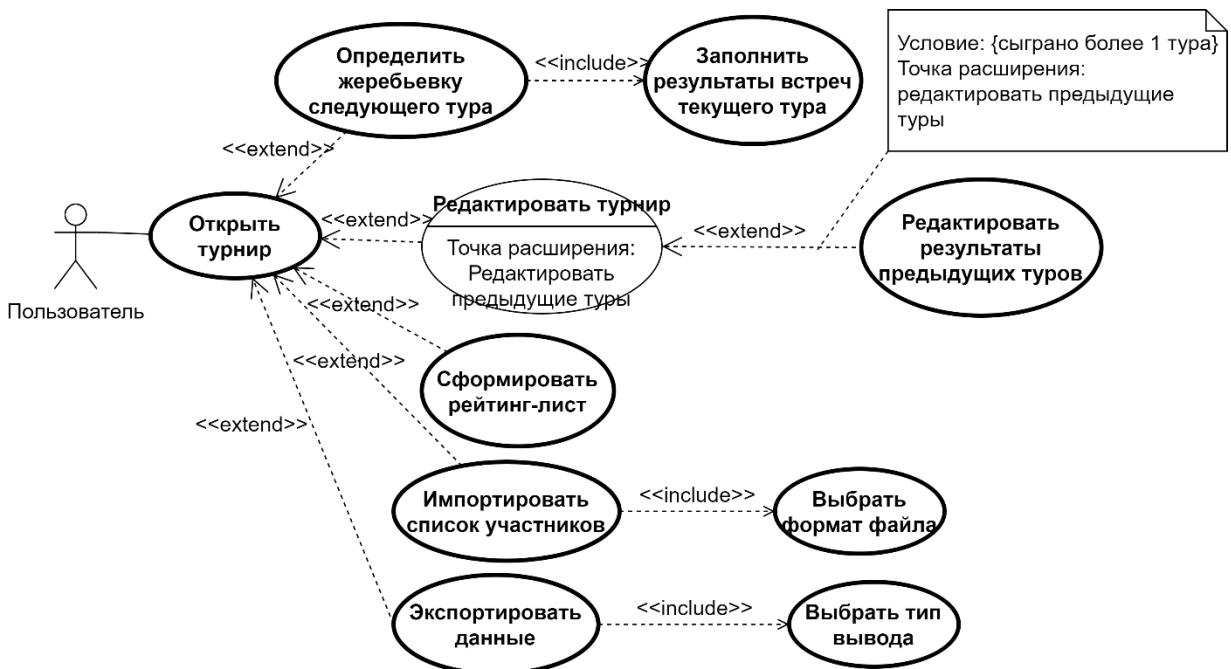


Рисунок 2.3 – Описание прецедентов взаимодействия пользователя с системой (3 часть)

Название: Определение жеребьевки следующего тура.

Авторы: Пользователь.

Краткое описание: Запуск распределения игроков на пары, заданного алгоритмом. После определения списка пар текущего тура, для продолжения пользователю нужно заполнить результаты встреч.

Триггер: Нажатие кнопки запуска жеребьевки.

Основной поток (таблица 2.11):

Таблица 2.11 – Прецедент "Определение жеребьевки следующего тура"

Действия акторов	Отклик системы
1. Пользователь открывает турнир из списка турниров.	2. Отображение окна турнира.
3. Пользователь нажимает на кнопку запуска жеребьевки нового тура.	4. Если результаты предыдущего тура заполнены или это первый тур, то запускается алгоритм жеребьевки (Е1).
5. Пользователь вводит результаты встреч.	6. Система обновляет таблицу пар.

Альтернативные потоки:

E1: Если результаты предыдущего тура не заполнены, то пользователю выводится сообщение о необходимости заполнения результатов предыдущего тура.

Название: Редактирование турнира.

Авторы: Пользователь.

Краткое описание: При необходимости редактирования существующего турнира пользователь, находясь в окне открытого турнира, открывает подменю, в котором находятся параметры турнира, задаваемые при его создании, которые можно отредактировать. Также, если турнир уже начался, у пользователя есть возможность отредактировать результаты жеребьевки прошедших туров.

Триггер: Открытие подменю с параметрами турнира.

Основной поток (таблица 2.12):

Таблица 2.12 – Прецедент "Редактирование турнира"

Действия акторов	Отклик системы
1. Пользователь открывает турнир из списка турниров.	2. Отображение окна турнира.
3. Пользователь нажимает на подменю с параметрами турнира.	4. Разворачивание подменю с параметрами (Е1).
5. Пользователь изменяет параметры.	6. Система применяет параметры.

Альтернативные потоки:

Е1: Если жеребьевка еще не начата, то возможно изменение всех параметров. Если жеребьевка уже началась, то нельзя изменять количество туров, вид, систему турнира.

Название: Редактирование результатов предыдущего тура.

Авторы: Пользователь.

Краткое описание: При необходимости редактирования результатов прошедшего тура (если при распределении очков произошла опечатка), пользователь может открыть список пар предыдущего тура и изменить результаты встреч.

Триггер: Нажатие на кнопку изменения результатов предыдущего тура.

Основной поток (таблица 2.13):

Таблица 2.13 – Прецедент "Редактирование результатов предыдущего тура"

Действия акторов	Отклик системы
1. Пользователь открывает турнир из списка турниров.	2. Отображение окна турнира.

Действия акторов	Отклик системы
3. Пользователь нажимает на вкладку со списком пар.	4. Предоставление списка пар текущего тура (E1) и навигацию по спискам предыдущих турнов (E2).
5. Пользователь выбирает перейти на список какого-либо из предыдущих турнов.	6. Предоставление списка пар выбранного тура.
7. Пользователь выбирает отредактировать результаты предыдущего тура.	8. Разблокировка редактирования результатов.
9. Пользователь изменяет результаты.	10. Обновление результатов.

Альтернативные потоки:

E1: Если жеребьевка еще не начата, то во вкладке со списками пар списков не будет.

E2: Если текущий тур первый, то предыдущих турнов нет, следовательно, навигация по спискам предыдущих турнов невозможна.

Название: Формирование рейтинг-листа.

Авторы: Пользователь.

Краткое описание: При необходимости редактирования результатов прошедшего тура (если при распределении очков произошла опечатка), пользователь может открыть список пар предыдущего тура и изменить результаты встреч.

Триггер: Нажатие на кнопку изменения результатов предыдущего тура.

Основной поток (таблица 2.14):

Таблица 2.14 – Прецедент "Формирование рейтинг-листа"

Действия акторов	Отклик системы
1. Пользователь открывает турнир из списка турниров.	2. Отображение окна турнира.
3. Пользователь нажимает на вкладку с рейтингом.	4. Предоставление списка участников, отсортированных по количеству очков и коэффициентам.

Название: Импорт списка участников.

Авторы: Пользователь.

Краткое описание: Пользователь может не вводить данные об участниках из одного турнира в другой, а использовать средства импорта списка из файлов (.csv, .xls).

Триггер: Нажатие кнопки импорта списка участников.

Основной поток (таблица 2.15):

Таблица 2.15 – Прецедент "Импорт списка участников"

Действия акторов	Отклик системы
1. Пользователь открывает турнир из списка турниров.	2. Отображение окна турнира.
3. Пользователь нажимает на кнопку импорта списка участников (Е1).	4. Отображение окна с выбором файлов с допустимыми типами файлов.
5. Пользователь выбирает файл.	6. Загрузка участников в список.

Альтернативные потоки:

Е1: Если жеребьевка турнира уже начата, импорт участников невозможен ввиду того, что нельзя заменить участников, которые уже находятся в рейтинг-листе.

Название: Экспорт данных.

Авторы: Пользователь.

Краткое описание: Пользователь может экспортировать любую таблицу (список участников, список пар, рейтинг-лист) в различные форматы файлов (.csv, .xls, .doc, .pdf) или распечатать на принтере.

Триггер: Нажатие кнопки экспорта.

Основной поток (таблица 2.16):

Таблица 2.16 – Прецедент "Экспорт данных"

Действия акторов	Отклик системы
1. Пользователь открывает турнир из списка турниров.	2. Отображение окна турнира.
3. Пользователь нажимает на кнопку экспорта.	4. Отображение окна с предпросмотром печатаемого документа.
5. Пользователь выбирает тип экспорта (файл или печать).	6. Осуществление экспорта (Е1).

Альтернативные потоки:

Е1: Если выбран экспорт в файл, то пользователю откроется окно сохранения файла.

2.2 Описание активностей процесса «to-be»

Исходя из постановки задачи, описанной ранее, были определены следующие бизнес-процессы.

2.2.1 Создание турнира

Бизнес-процесс «Создание турнира» (см. рис. 2.4) описывает процессы взаимодействия пользователя с программой и сервером базы данных с целью создания турнира для дальнейшего добавления участников, команд.

Для получения прав доступа по созданию турниров, пользователю необходимо авторизоваться, после чего он может выбрать в меню пункт создания турнира.

Далее пользователю предоставляется форма заполнения параметров турнира, среди которых: вид турнира (личный, командный, лично-командный), количество туров, название турнира, место проведения, дата и время начала, продолжительность в часах, название организации, проводящей турнир, также пользователю необходимо выбрать, разрешить ли смешанные игры между игроками из разных групп.

После заполнения всех параметров, программа обрабатывает их и отправляет запрос на добавление турнира, если название турнира уникально среди других турниров пользователя, то турнир добавляется в базу данных для пользователя. Если турнир с таким названием уже создан этим пользователем, тогда выводится ошибка о необходимости поменять название турнира.

После создания турнира для пользователя открывается окно этого турнира, в котором доступны вкладки со списком участников, списком пар, рейтинг-листом, с которыми пользователь может взаимодействовать далее.

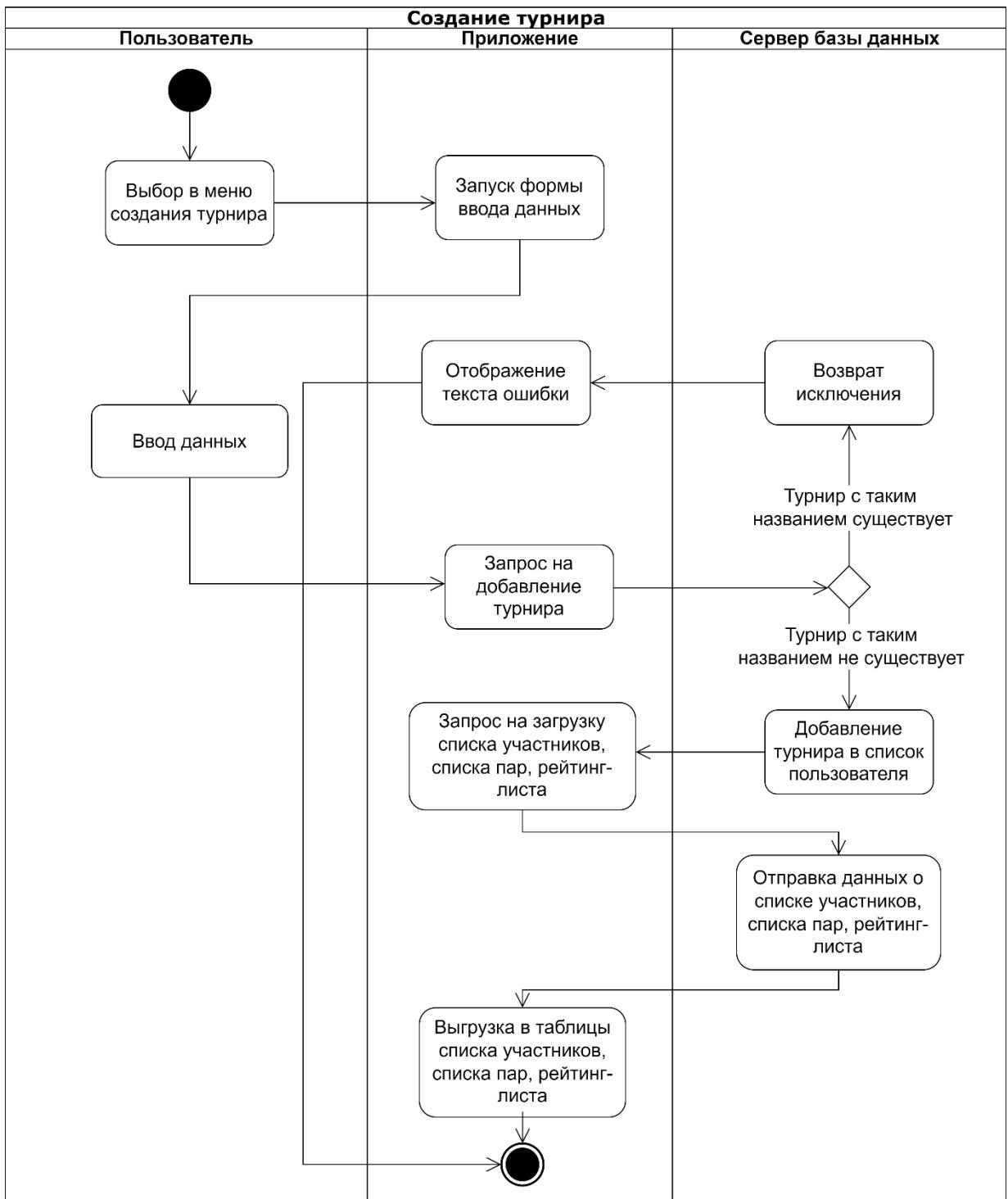


Рисунок 2.4 – Диаграмма активности «Создание турнира»

2.2.2 Заполнение данных турнира

Бизнес-процесс «Заполнение данных турнира» (см. рис. 2.5) описывает процессы взаимодействия пользователя с программой и сервером базы данных с целью добавления в списки участников, команд, для дальнейшего проведения жеребьевки турнира.

После авторизации и открытия или создания турнира, пользователю необходимо заполнить список участников или импортировать его, определить группы

игроков, также если турнир подразумевает наличие команд, пользователю необходимо заполнить и их.

После окончания заполнения данных турнира, программа отправляет запрос на добавление данных в базу, после чего данные сохраняются.

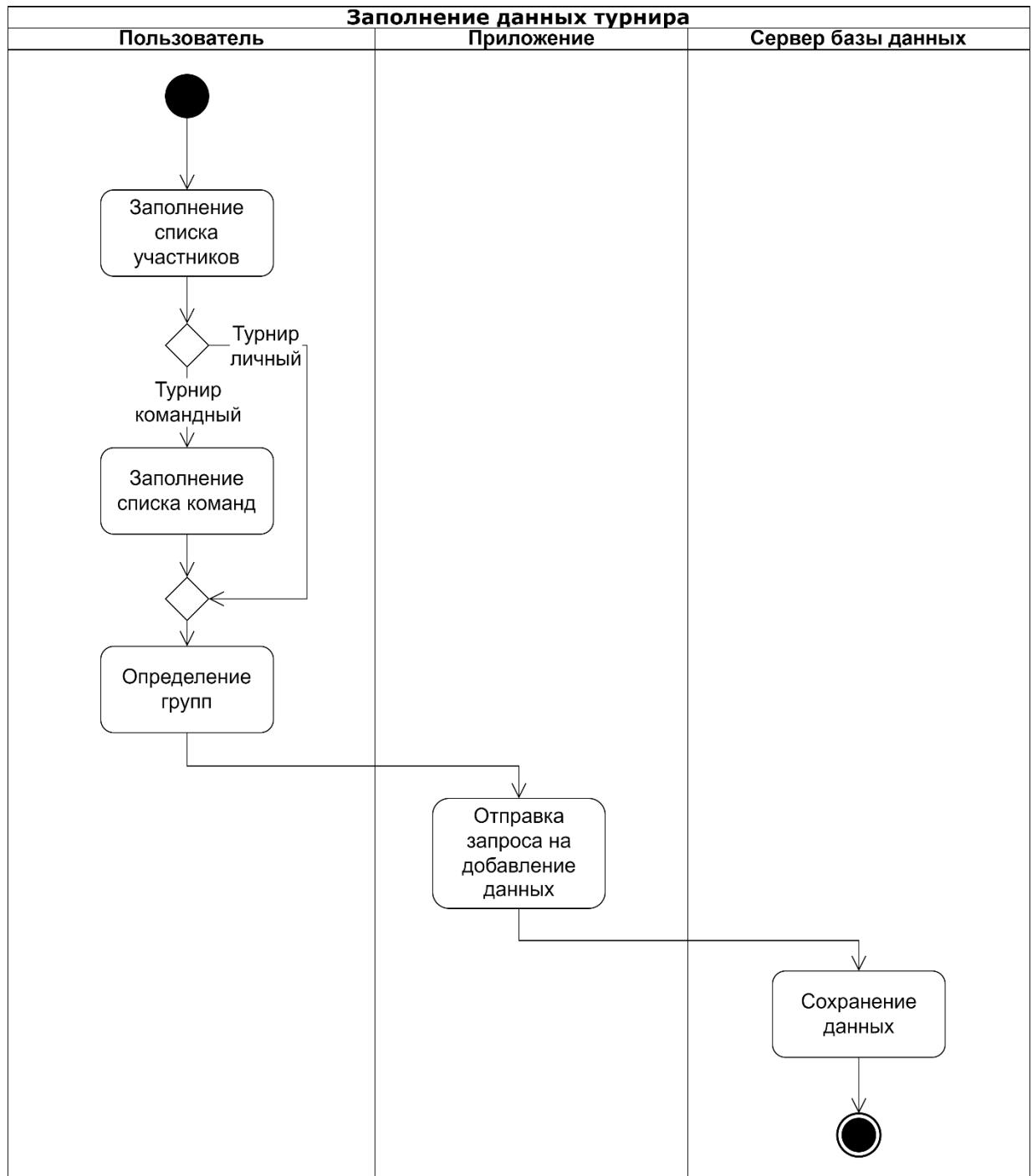


Рисунок 2.5 – Диаграмма активности «Заполнение данных турнира»

2.2.3 Проведение жеребьевки

После заполнения участников, команд, групп, пользователь может начать проводить жеребьевку первого тура. Описание этого бизнес-процесса приведено на рисунке 2.6.

Пользователь запускает алгоритм жеребьевки пар посредством нажатия на соответствующую кнопку в меню турнира.

Если участников меньше, чем количество турнов, то пользователю выводится сообщение о невозможности запуска жеребьевки. Если участников достаточное количество, то программа запускает алгоритм, по результатам которого, если разбиение на пары прошло успешно, то в разделе списков пар появится таблица с текущим номером тура, в которой будут находиться данные игроков, играющих друг с другом.

После того, как пользователь получает список пар, то он может распечатать его для заполнения результатов и перенесения их в программу. В качестве ввода результатов предполагается панель с выбором результата для каждой пары.

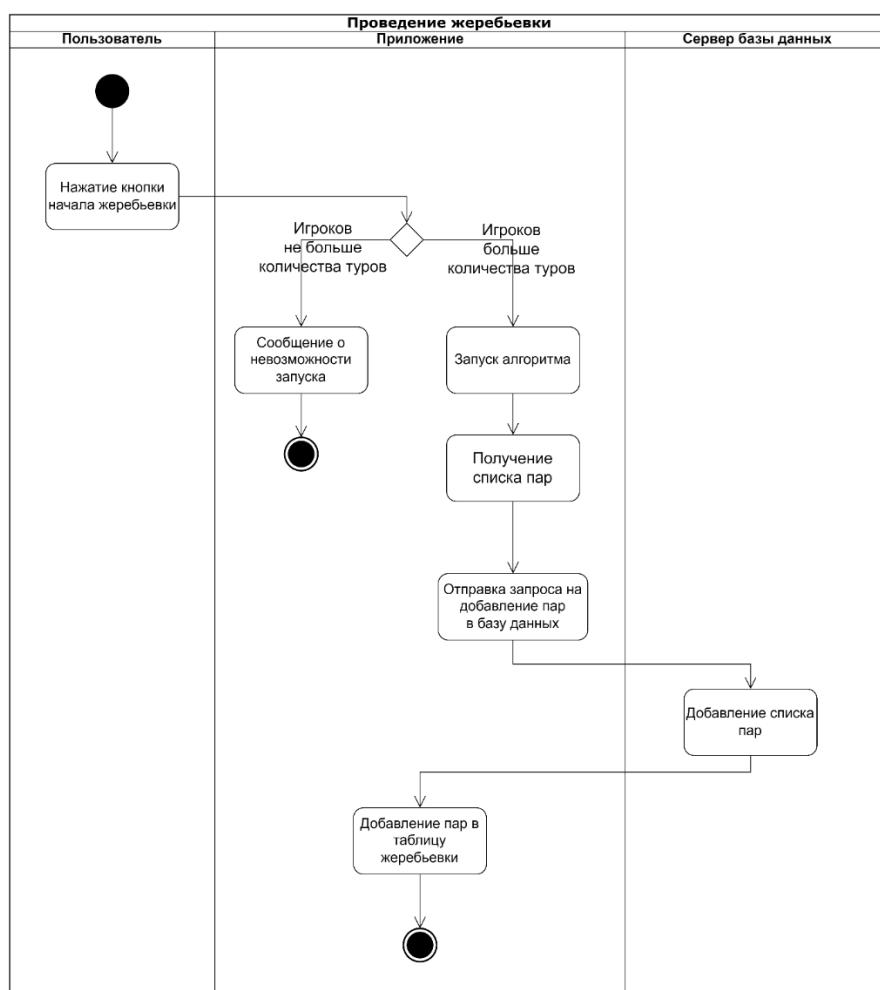


Рисунок 2.6 – Бизнес-процесс "Проведение жеребьевки"

2.3 Проектирование базы данных

На основе объектов и их атрибутов, определенных в главе 1 пункта 6, были выявлены требования к данным (см. ПРИЛОЖЕНИЕ Г, таблица Г1).

2.3.1 ER-диаграмма

Для графического отображения объектов, их атрибутов и связей между ними, была построена диаграмма «сущность-связь» в нотации Чена (см. рисунок 2.7).

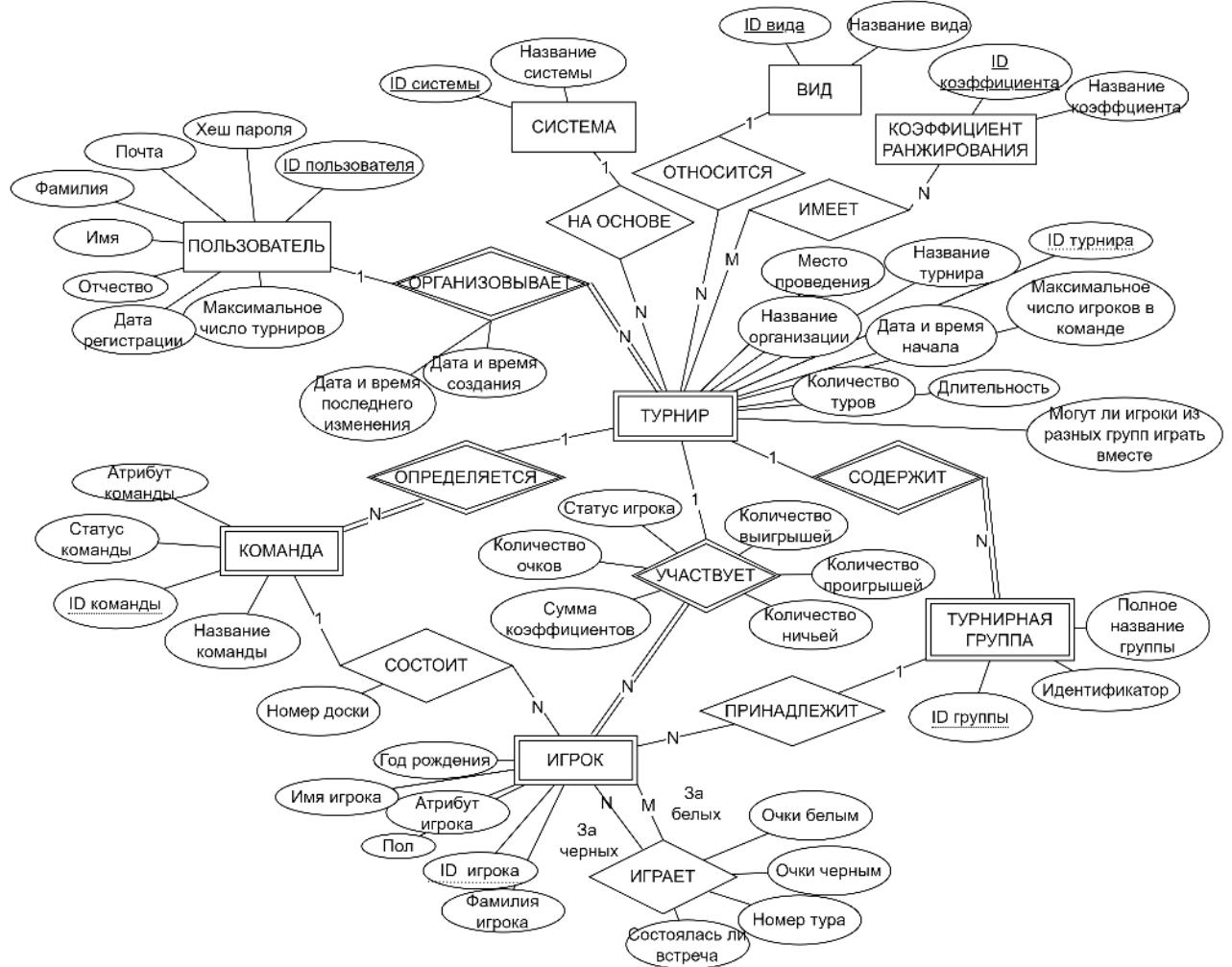


Рисунок 2.7 – ERD в нотации Чена

2.3.2 Определение таблиц

Исходя из данной диаграммы были выявлены следующие сущности:

- Пользователь: {[ID пользователя, Фамилия, Имя, Отчество, Почта, Хеш пароля, Лимит (максимальное число турниров, Дата регистрации]}.

2. Турнир: {[ID турнира, ID организатора (пользователя), Название турнира, Количество туров, Место проведения, Дата и время начала, Длительность, Максимальное число игроков в команде, Название организации, Могут ли игроки из разных групп играть вместе]}.

Данный случай показывает, что ID турнира разные только для 1 организатора, для другого организатора они могут повторяться.

3. Система: {[ID системы, Название системы]}.
4. Вид: {[ID вида, Название вида]}.
5. Игрок: {[ID игрока, ID турнира, ID организатора, Фамилия игрока, Имя игрока, Пол, Атрибут игрока, Год рождения]}.
6. Команда: {[ID команды, ID турнира, ID организатора, Название команды, Атрибут команды, Статус команды]}.
7. Коэффициент ранжирования: {[ID коэффициента, Название коэффициента]}.
8. Турнирная группа: {[ID группы, ID турнира, ID организатора, Идентификатор, Полное название группы]}.

Также были определены следующие связи между сущностями:

1. Организовывает: {[ID турнира, ID пользователя, Дата и время создания, Дата и время последнего изменения]}.
2. На основе: {[ID турнира, ID организатора, ID системы]}.
3. Относится: {[ID турнира, ID организатора, ID вида]}.
4. Участвует: {[ID игрока, ID турнира, ID организатора, Статус игрока, Количество очков, Сумма коэффициентов, Количество выигрышей, Количество проигрышей, Количество ничьей]}.
5. Определяется: {[ID команды, ID турнира, ID организатора]}.
6. Состоит: {[ID игрока, ID турнира, ID организатора, ID команды, Номер доски]}.
7. Играет: {[ID игрока белых, ID игрока черных, ID турнира, ID организатора, Очко белым, Очко черным, Состоялась ли Игра, Номер тура]}.
8. Турнир имеет коэффициенты: {[ID турнира, ID организатора, ID коэффициента]}.
9. Турнир содержит группы: {[ID турнира, ID организатора, ID группы]}.

10. Игрок принадлежит группе: {[ID игрока, ID турнира, ID организатора, ID группы]}.

На следующем этапе производится слияние сущностей и связей с одинаковым набором ключевых атрибутов:

1. (Турнир и Организовывает) Турнир: {[ID турнира, ID организатора, Название турнира, Количество туров, Место проведения, Дата и время начала, Длительность, Максимальное число игроков в команде, Название организации, Могут ли игроки из разных групп играть вместе, Дата и время создания, Дата и время последнего изменения]}.
2. (Турнир и На основе) Турнир: {[ID турнира, ID организатора, Название турнира, Количество туров, Место проведения, Дата и время начала, Длительность, Максимальное число игроков в команде, Название организации, Могут ли игроки из разных групп играть вместе, Дата и время создания, Дата и время последнего изменения, ID системы]}.
3. (Турнир и Относится) Турнир: {[ID турнира, ID организатора, Название турнира, Количество туров, Место проведения, Дата и время начала, Длительность, Максимальное число игроков в команде, Название организации, Могут ли игроки из разных групп играть вместе, Дата и время создания, Дата и время последнего изменения, ID системы, ID вида]}.
4. (Игрок и Участвует) Игрок: {[ID игрока, ID турнира, ID организатора, Фамилия игрока, Имя игрока, Пол, Атрибут игрока, Год рождения, Статус игрока, Количество очков, Сумма коэффициентов, Количество выигрышней, Количество проигрышней, Количество ничьей]}.
5. (Игрок и Состоит) Игрок: {[ID игрока, ID турнира, ID организатора, Фамилия игрока, Имя игрока, Пол, Атрибут игрока, Год рождения, Статус игрока, ID команды, Количество очков, Сумма коэффициентов, Количество выигрышней, Количество проигрышней, Количество ничьей, Номер доски]}.

6. (Игрок и Принадлежит группе) Игрок: {[ID игрока, ID турнира, ID организатора, Фамилия игрока, Имя игрока, Пол, Атрибут игрока, Год рождения, Статус игрока, ID команды, Количество очков, Сумма коэффициентов, Количество выигрышей, Количество проигрышней, Количество ничьей, Номер доски, ID группы]}.
7. (Команда и Определяется): Команда: {[ID команды, ID турнира, ID организатора, Название команды, Атрибут команды, Статус команды]}.
8. (Турнирная группа и Турнир содержит группы) Турнирная группа: {[ID группы, ID турнира, ID организатора, Идентификатор, Полное название группы]}.
9. Сущность Пользователь остается без изменения.
10. Сущность Система остается без изменения.
11. Сущность Вид остается без изменения.
12. Сущность Коэффициент ранжирования остается без изменения.
13. Связь Играет (Игра) остается без изменения.
14. Связь Турнир имеет коэффициенты (Коэффициенты турнира) остаётся без изменения.

Итог — 10 сущностей и связей в схеме:

1. Турнир: {[ID турнира, ID организатора, Название турнира, Количество турнов, Место проведения, Дата и время начала, Длительность, Максимальное число игроков в команде, Название организации, Могут ли игроки из разных групп играть вместе, Дата и время создания, Дата и время последнего изменения, ID системы, ID вида]}.
2. Игрок: {[ID игрока, ID турнира, ID организатора, Фамилия игрока, Имя игрока, Пол, Атрибут игрока, Год рождения, Статус игрока, ID команды, Количество очков, Сумма коэффициентов, Количество выигрышней, Количество проигрышней, Количество ничьей, Номер доски, ID группы]}.
3. Команда: {[ID команды, ID турнира, ID организатора, Название команды, Атрибут команды, Статус команды]}.

4. Пользователь: {[ID пользователя, Фамилия, Имя, Отчество, Почта, Хеш пароля, Лимит (максимальное число) турниров, Дата регистрации]}.
5. Система: {[ID системы, Название системы]}.
6. Вид: {[ID вида, Название вида]}.
7. Коэффициент ранжирования: {[ID коэффициента, Название коэффициента]}.
8. Игра: {[ID игрока белых, ID игрока черных, ID турнира, ID организатора, Очко белым, Очко черным, Состоялась ли Игра, Номер тура]}
9. Коэффициент турнира: {[ID турнира, ID организатора, ID коэффициента
- 10. Турнирная группа: {[ID группы, ID турнира, ID организатора, Идентификатор, Полное название группы]}.

2.3.3 Спецификация объектов

На основании требований к данным, а также проведенного проектирования схемы данных, можно выделить следующие объекты, их атрибуты и первичные ключи.

Таблица 2.17 – Спецификация объектов

Объект	Атрибуты	Первичный ключ
ПОЛЬЗОВАТЕЛЬ	1) ID пользователя; 2) Электронная почта; 3) Фамилия пользователя; 4) Имя пользователя; 5) Отчество пользователя; 6) Хеш пароля; 7) Лимит турниров; 8) Дата регистрации.	Идентификационный Описательный Описательный Описательный Описательный Описательный Описательный Описательный

Объект	Атрибуты	Первичный ключ
ТУРНИР	1) ID турнира; 2) ID организатора; 3) Название турнира; 4) Количество туров; 5) Место проведения; 6) Дата и время начала; 7) Длительность; 8) Максимальное число игроков в команде; 9) Название организации; 10) Могут ли игроки из разных групп играть вместе; 11) Дата и время создания; 12) Дата и время последнего изменения; 13) ID системы; 14) ID вида.	Идентификационный Идентификационный Описательный Описательный Описательный Описательный Описательный Описательный Описательный Описательный Описательный Описательный Описательный Описательный Описательный
СИСТЕМА	1) ID системы; 2) Название системы.	Идентификационный Описательный
ВИД	1) ID вида; 2) Название вида.	Идентификационный Описательный
ИГРОК	1) ID игрока; 2) ID турнира; 3) ID организатора; 4) Фамилия игрока; 5) Имя игрока; 6) Пол; 7) Атрибут игрока; 8) Год рождения; 9) Статус игрока; 10) ID команды; 11) Количество очков; 12) Сумма коэффициентов; 13) Количество выигрышей; 14) Количество проигрышей; 15) Количество ничьей; 16) Номер доски; 17) ID группы.	Идентификационный Идентификационный Идентификационный Описательный Описательный Описательный Описательный Описательный Описательный Описательный Описательный Описательный Описательный Описательный Описательный Описательный Описательный
ИГРА	1) ID игрока за белых; 2) ID игрока за черных; 3) ID турнира; 4) ID организатора; 5) Очки белым; 6) Очки черным; 7) Состоялась ли Игра; 8) Номер тура.	Идентификационный Идентификационный Идентификационный Идентификационный Описательный Описательный Описательный Описательный

Объект	Атрибуты	Первичный ключ
КОМАНДА	1) ID команды; 2) ID турнира; 3) ID организатора; 4) Название команды; 5) Атрибут команды; 6) Статус команды.	Идентификационный Идентификационный Идентификационный Описательный Описательный Описательный
КОЭФФИЦИЕНТ РАНЖИРОВАНИЯ	1) ID коэффициента; 2) Название коэффициента.	Идентификационный Описательный
КОЭФФИЦИЕНТ ТУРНИРА	1) ID турнира; 2) ID организатора; 3) ID коэффициента.	Идентификационный Идентификационный Идентификационный
ТУРНИРНАЯ ГРУППА	1) ID группы; 2) ID турнира; 3) ID организатора; 4) Идентификатор; 5) Полное название группы.	Идентификационный Идентификационный Идентификационный Описательный Описательный

На следующей таблице (см. таблица 2.18) представлены связи между объектами.

Таблица 2.18 – Связи между объектами

Наименование связи	Объекты	Показатель кардинальности	Степень участия
ОРГАНИЗОВЫВАЕТ	ПОЛЬЗОВАТЕЛЬ ТУРНИР	1:N	Частичная Полная
УЧАСТВУЕТ	ТУРНИР ИГРОК	1:N	Частичная Полная
НА ОСНОВЕ	ТУРНИР СИСТЕМА	N:1	Частичная Частичная
ОТНОСИТСЯ	ТУРНИР ВИД	N:1	Частичная Частичная
ИМЕЕТ	ТУРНИР КОЭФФИЦИЕНТ ТУРНИРА	1:N	Частичная Частичная
ВЗЯТ ИЗ	КОЭФФИЦИЕНТ ТУРНИРА КОЭФФИЦИЕНТ РАНЖИРОВАНИЯ	N:1	Частичная Частичная
СОСТОИТ	ИГРОК КОМАНДА	N:1	Частичная Полная
ОПРЕДЕЛЯЕТСЯ	КОМАНДА ТУРНИР	N:1	Полная Частичная

Наименование связи	Объекты	Показатель кардинальности	Степень участия
ИГРАЕТ	ИГРОК ИГРА	1:N	Частичная Полная
СОДЕРЖИТ	ТУРНИР ТУРНИРНАЯ ГРУППА	1:N	Частичная Полная
ПРИНАДЛЕЖИТ	ТУРНИРНАЯ ГРУППА ИГРОК	1:N	Частичная Частичная

2.3.4 Определение статуса таблиц

Чтобы создать бинарные связи между таблицами при помощи механизма первичных и внешних ключей, сначала нужно определить, какая из двух таблиц связи будет родительской, а какая – дочерней. Для этого следует обратиться к таблице 2.19 «Связи между объектами» и с учетом типа связи и степени участия таблиц в связи определить статус таблиц.

Таблица 2.19 – Статус таблиц

Наименование связи	Объекты	Показатель кардинальности	Степень участия	Статус таблиц
ОРГАНИЗОВЫВАЕТ	ПОЛЬЗОВАТЕЛЬ ТУРНИР	1:N	Частичная Полная	Родитель-ская Дочерняя
УЧАСТВУЕТ	ТУРНИР ИГРОК	1:N	Частичная Полная	Родитель-ская Дочерняя
НА ОСНОВЕ	ТУРНИР СИСТЕМА	N:1	Частичная Частичная	Родитель-ская Дочерняя
ОТНОСИТСЯ	ТУРНИР ВИД	N:1	Частичная Частичная	Родитель-ская Дочерняя
ИМЕЕТ	ТУРНИР КОЭФФИЦИЕНТ ТУРНИРА	1:N	Частичная Частичная	Дочерняя Родитель-ская
ВЗЯТ ИЗ	КОЭФФИЦИЕНТ ТУРНИРА КОЭФФИЦИЕНТ РАНЖИРОВАНИЯ	N:1	Частичная Частичная	Родитель-ская Дочерняя

Наименование связи	Объекты	Показатель кардинальности	Степень участия	Статус таблиц
СОСТОИТ	ИГРОК КОМАНДА	N:1	Частичная Полная	Дочерняя Родительская
ОПРЕДЕЛЯ- ЕТСЯ	КОМАНДА ТУРНИР	N:1	Полная Частичная	Дочерняя Родительская
ИГРАЕТ	ИГРОК ИГРА	1:N	Частичная Полная	Дочерняя Родительская
СОДЕРЖИТ	ТУРНИР ТУРНИРНАЯ ГРУППА	1:N	Частичная Полная	Родительская Дочерняя
ПРИНАДЛЕЖИТ	ТУРНИРНАЯ ГРУППА ИГРОК	1:N	Частичная Частичная	Родительская Дочерняя

2.3.5 Создание реляционных связей

После того, как в бинарной связи определена дочерняя и родительская таблицы, в дочерней таблице следует создать внешний ключ – копию первичного ключа родительской таблицы и связать таблицы по одноименному полю. В таблице 2.20 представлены реляционные связи проектируемой базы данных.

Таблица 2.20 – Реляционные связи

Наименование связи	Объекты	Статус таблиц	Ключи
ОРГАНИЗОВЫВАЕТ	ПОЛЬЗОВАТЕЛЬ ТУРНИР	Родительская Дочерняя	ID пользователя (ПК) (ID турнира, ID организатора) (ПК) ID пользователя (ВК)
УЧАСТВУЕТ	ТУРНИР ИГРОК	Родительская Дочерняя	(ID турнира, ID организатора) (ПК) (ID игрока, ID турнира, ID организатора) (ПК) (ID турнира, ID организатора) (ВК)
НА ОСНОВЕ	ТУРНИР СИСТЕМА	Дочерняя Родительская	(ID турнира, ID организатора) (ПК) ID системы (ПК) ID системы (ВК)

Наименование связи	Объекты	Статус таблиц	Ключи
ОТНОСИТСЯ	ТУРНИР ВИД	Дочерняя Родительская	(ID турнира, ID организатора) (ПК) ID вида (ПК) ID вида (ВК)
ИМЕЕТ	ТУРНИР КОЭФФИЦИЕНТ ТУРНИРА	Родительская Дочерняя	(ID турнира, ID организатора) (ПК) (ID турнира, ID организатора, ID коэффициента) (ПК) (ID турнира, ID организатора) (ВК)
ВЗЯТ ИЗ	КОЭФФИЦИЕНТ ТУРНИРА КОЭФФИЦИЕНТ РАНЖИРОВАНИЯ	Дочерняя Родительская	(ID турнира, ID организатора, ID коэффициента) (ПК) (ID коэффициента) (ПК) ID коэффициента (ВК)
СОСТОИТ	ИГРОК КОМАНДА	Дочерняя Родительская	(ID игрока, ID турнира, ID организатора) (ПК) (ID команды, ID турнира, ID организатора) (ПК) (ID команды, ID турнира, ID организатора) (ВК)
ОПРЕДЕЛЯЕТСЯ	КОМАНДА ТУРНИР	Дочерняя Родительская	(ID команды, ID турнира, ID организатора) (ПК) (ID турнира, ID организатора) (ПК) (ID турнира, ID организатора) (ВК)
ИГРАЕТ	ИГРОК ИГРА	Родительская Дочерняя	(ID игрока, ID турнира, ID организатора) (ПК) (ID игрока белых, ID игрока черных, ID турнира, ID организатора) (ПК) (ID игрока, ID турнира, ID организатора) (ВК)
СОДЕРЖИТ	ТУРНИР ТУРНИРНАЯ ГРУППА	Родительская Дочерняя	(ID турнира, ID организатора) (ПК) (ID группы, ID турнира, ID организатора) (ПК) (ID турнира, ID организатора) (ВК)

Наименование связи	Объекты	Статус таблиц	Ключи
ПРИНАДЛЕЖИТ	ТУРНИРНАЯ ГРУППА ИГРОК	Родительская Дочерняя	(ID группы, ID турнира, ID организатора) (ПК) (ID игрока, ID турнира, ID организатора) (ПК) (ID группы, ID турнира, ID организатора) (ВК)

2.3.6 Выбор СУБД

Для реализации спроектированной реляционной схемы была выбрана СУБД PostgreSQL.

Причины выбора данной СУБД:

1. Не требует оплаты лицензии для использования.
2. Широкое русскоязычное сообщество – большое количество информации на русском языке.
3. Свободное ПО (не зависит от политики конкретной компании).

2.3.7 Физическая модель базы данных

Исходя из проведенного проектирования схемы реляционной базы данных, а также с учетом выбора PostgreSQL в качестве СУБД, была спроектирована следующая схема физической модели базы данных (см. рис. 2.8), выполненная в нотации диаграммы «сущность-связь» UML.

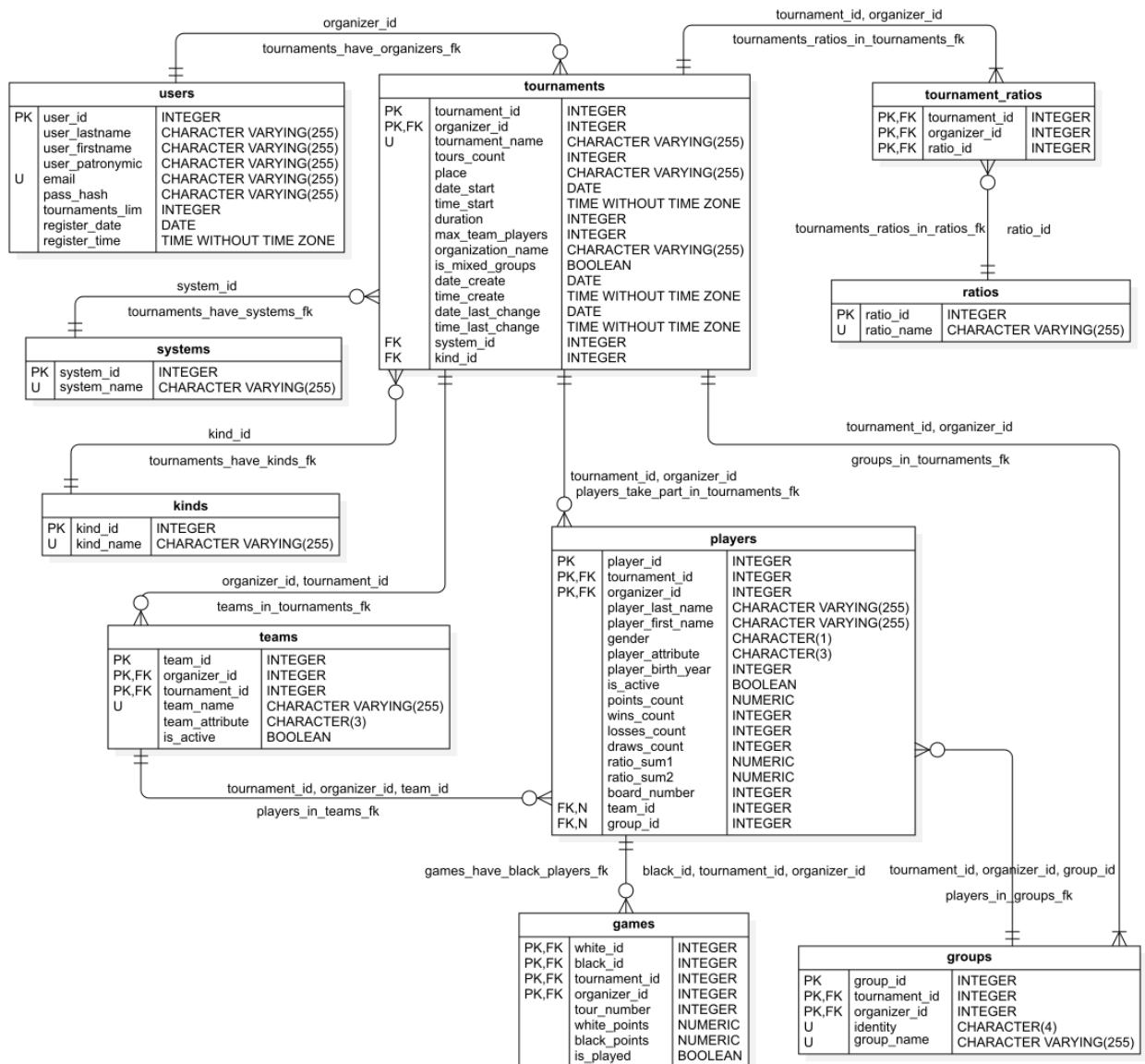


Рисунок 2.8 – Схема физической модели базы данных

2.3.8 Встроенные процедуры

Несмотря на то, что во время реализации планируется использовать Entity Framework, встроенные процедуры полезны тогда, когда нет доступа к приложению, а написание запросов на языке SQL может быть трудоемко.

Также эти процедуры могут быть полезны, если будет создано другое приложение, не использующее Entity Framework.

2.3.8.1 Процедура добавления игрока

Для добавления игрока определена процедура (см. ПРИЛОЖЕНИЕ Д, рисунок Д1), принимающая в качестве параметров данные об игроке, некоторые из которых необязательны и могут быть заполнены значениями по умолчанию.

Далее проверяются введенные параметры. Проверяется существование организатора с переданным id, проверяется существование турнира у этого организатора, проверяется существование команды и уникальность игрока, корректность номера доски игрока при командном турнире.

Если проверки успешны, то игрок добавляется в базу, иначе вызываются исключения, соответствующие не пройденной проверке.

2.3.8.2 Процедура добавления команды

По полной аналогии с процедурой добавления игрока, организована процедура добавления команды (см. ПРИЛОЖЕНИЕ Д, рисунок Д2).

Вначале организована проверка введенных параметров, после чего если данные корректны, то выполняется запрос, иначе – вызываются исключения.

2.3.8.3 Процедура добавления турнира

Таким же образом организована процедура добавления турнира (см. ПРИЛОЖЕНИЕ Д, рисунок Д3). Однако в ней проверок всего 2: проверка на существование организатора, проверка на уникальность названия турнира.

2.3.8.4 Процедура добавления пользователя

В данной процедуре (см. ПРИЛОЖЕНИЕ Д, рисунок Д4) реализована одна проверка – на отсутствие пользователя с такими же данными, ключевым данным на уникальность был принят email (логин).

2.3.8.5 Процедура удаления игрока

При удалении игрока (см. ПРИЛОЖЕНИЕ Д, рисунок Д5), помимо проверки на корректность данных, выполняется проверка на то, что этот игрок не состоит в играх, потому как нельзя удалить такого игрока, так как потеряется информация о встрече, вместо этого пользователю предлагается сделать игрока неактивным.

2.3.8.6 Процедура удаления команды

При удалении команды (см. ПРИЛОЖЕНИЕ Д, рисунок Д6), по аналогии с удалением игрока, производится проверка на то, что игроки данной команды не участвуют в играх. Если существуют записи в играх с данной командой, то удаление отклоняется во избежание потери информации.

2.3.8.7 Процедура удаления турнира

При удалении турнира (см. ПРИЛОЖЕНИЕ Д, рисунок Д7), проверяются параметры, передаваемы в процедуру. Также проверяется, есть ли в данном турнире игры, если они есть, то пользователю выдается предупреждение, что эти игры также будут удалены.

2.3.8.8 Процедура удаления пользователя

При удалении пользователя (см. ПРИЛОЖЕНИЕ Д, рисунок Д8), производится проверка на существование пользователя с введенными данными, если пользователь существует, то запись о нем удаляется из базы.

Необходимости удалять для пользователя его турниры, игроков и прочее нет, так как при создании таблиц было задано каскадное удаление этих записей.

2.3.8.9 Примеры вызова процедур

На следующем рисунке (см. рисунок 2.9) представлены примеры вызова процедур.

```
CALL add_user(p_last_name := 'Иванов',
              p_first_name := 'Александр',
              p_patronymic := 'Петрович',
              p_email := 'ivanov_alex@gmail.com',
              p_password_hash := 'asldnviedij4yufdadso3p24');

CALL add_tournament(p_organizer_id := 9,
                     p_tournament_name := 'Турнир по шахматам',
                     p_system_id := 1,
                     p_kind_id := 1);

CALL add_team(p_tournament_id := 7
              , p_organizer_id := 9
              , p_team_name := 'ОСШ 4');

CALL add_team(p_tournament_id := 7, p_organizer_id := 9, p_team_name := 'ОСШ 4');

CALL add_player(p_tournament_id := 7,
                p_organizer_id := 9,
                p_player_last_name := 'Филиппов',
                p_player_first_name := 'Иван',
                p_sex := 'M',
                p_player_attribute := '4MS',
                p_team_name := 'ОСШ 4',
                p_birth_year := 2007,
                p_board_number := 1);

CALL add_player(p_tournament_id := 7,
                p_organizer_id := 9,
                p_player_last_name := 'Филиппов',
                p_player_first_name := 'Андрей',
                p_team_name := 'ОСШ 4',
                p_sex := 'M',
                p_player_attribute := '2MS',
                p_birth_year := 2004,
                p_board_number := 2);

CALL delete_user(p_user_id := 9);
```

Рисунок 2.9 – Примеры вызовов процедур

2.3.9 Триггеры

Для частичной реализации бизнес-логики на стороне СУБД для того, чтобы эта бизнес-логика выполнялась независимо от способа доступа к данным, были реализованы следующие триггеры.

2.3.9.1 Триггер обновления очков

Для автоматизации обновления очков игроков по мере проведения игр и сыгранных партий, был создан триггер (см. ПРИЛОЖЕНИЕ Е, рисунок Е1), обновляющий данные очков игроков по мере добавления записей результатов их игр.

2.3.9.2 Триггер обновления результатов игроков

Для автоматизации обновления количества выигранных, проигранных партий, а также ничей, был создан триггер (см. ПРИЛОЖЕНИЕ Е, рисунок Е2), обновляющий эти данные для каждого игрока в зависимости от результатов сыгранных партий.

2.3.10 Представления

Представления были созданы исходя из тех же целей, что и процедуры – для упрощения доступа к данным, независимо от выбранной технологии доступа к ним.

2.3.10.1 Представление списка игроков

Для того, чтобы при запросе списка игроков была предоставлена только важная информация, было определено представление списка игроков (см. ПРИЛОЖЕНИЕ Ж, рисунок Ж1).

Благодаря данному представлению можно получить одним запросом всю информацию об играх, включая информацию об их группах и командах, без необходимости написания вложенных запросов.

2.3.10.2 Представление списка команд

Для просмотра информации о командах и количестве игроков в них, было создано представление (см. ПРИЛОЖЕНИЕ Ж, рисунок Ж2).

2.3.10.3 Представление команды

Для просмотра информации о конкретной команде, включая список игроков, входящих в ее состав, было создано представление (см. ПРИЛОЖЕНИЕ Ж, рисунок Ж3).

2.3.10.4 Представление одиночного рейтинг-листа

Одиночный рейтинг лист представлен в виде представления (см. ПРИЛОЖЕНИЕ Ж, рисунок Ж4) и организован в виде таблицы с рейтингом, полученным с помощью оконной функции DENSE_RANK, сортировки по убыванию количества очков и коэффициентов.

2.3.10.5 Представление командного рейтинг-листа

По аналогии с личным рейтинг-листом, было определено представление командного рейтинг-листа (см. ПРИЛОЖЕНИЕ Ж, рисунок Ж5), в котором дополнительно высчитывается сумма очков и коэффициентов каждого игрока.

2.3.10.6 Пример представления игроков

На следующем рисунке (см. рисунок 2.10) представлен пример структуры представления списка игроков.

player_last_name	player_first_name	gender	player_attribute	group_ident	team_name	player_birth_year	is_act
Иванов	Иван	М	-	M25	<null>	1999	• true
Сергеев	Сергей	М	-	M25	<null>	2000	• true
Иванов	Иван	М	-	<null>	<null>	2000	• true
Федоров	Федор	М	-	<null>	<null>	2000	• true
Носков	Александр	М	3MS	<null>	Медведь	2006	• true
Некрасов	Александр	М	1MS	<null>	Медведь	2003	• true
Филиппов	Артём	М	1MS	<null>	Медведь	2004	• true
Баяндина	Вероника	Ж	2FS	<null>	Медведь	2008	• true
Копытов	Илья	М	1MS	<null>	ОСШ 1	2003	• true
Некрасов	Александр	М	1MS	<null>	<null>	2003	• true
Носков	Александр	М	2MS	<null>	<null>	2006	• true
Носкова	Вероника	Ж	3FS	Д13	ОСШ 1	2010	• true

Рисунок 2.10 – Пример представления списка игроков

2.4 Проектирование информационной системы

Общая архитектура системы представлена на следующей диаграмме (см. рисунок 2.11), исходя из которой производится разделение системы на следующие уровни:

1. Уровень доступа к данным – DataAccess – организация классов-запросов к данным, взаимодействующих с ORM EntityFramework.
2. Уровень бизнес-логики (алгоритмы) – Domain – алгоритмы бизнес-логики (жеребьевка).
3. Уровень приложения (интерфейс и взаимодействие пользователя с системой) – WPF – организация пользовательского интерфейса с помощью паттерна MVVM (Model-View-ViewModel).

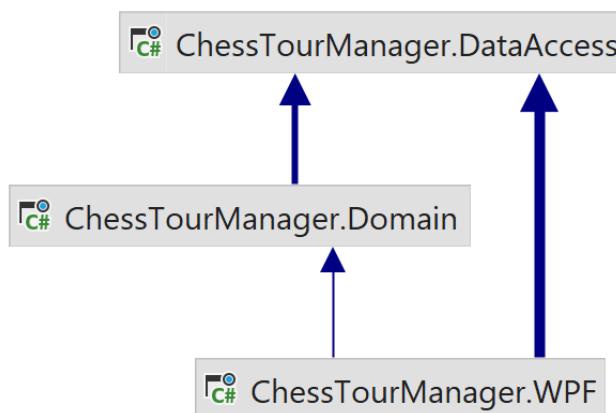


Рисунок 2.11 – Архитектура системы

2.4.1 Проектирование пользовательского интерфейса

Для успешного проектирования структуры пользовательского интерфейса были построены диаграммы последовательностей для структуризации взаимодействия пользователя с системой, затем были спроектированы макеты интерфейса с помощью интегрированной среды разработки Microsoft Visual Studio.

2.4.1.1 Последовательность создания турнира

После того как пользователь авторизуется, система открывает главное окно со списком турниров, доступных пользователю.

Для того, чтобы создать турнир, пользователь нажимает соответствующую кнопку в меню приложения. После чего открывается окно для заполнения параметров турнира.

Если параметры корректны и турнир с таким же именем в базе пользователя не существует, то этот турнир добавляется в базу данных и открывается, иначе пользователю предлагается изменить параметры турнира.



Рисунок 2.12 – Диаграмма последовательности "Создание турнира"

2.4.1.2 Последовательность добавления данных турнира

Для начала работы с турниром, пользователь открывает его нажатием соответствующей кнопки. После чего отправляется запрос в базу о получении данных турнира, открывается вкладка с данными этого турнира.

Далее пользователь может заполнять данные: участники, группы, команды, по мере заполнения данных, отправляются запросы на обновление и добавление данных в базу.

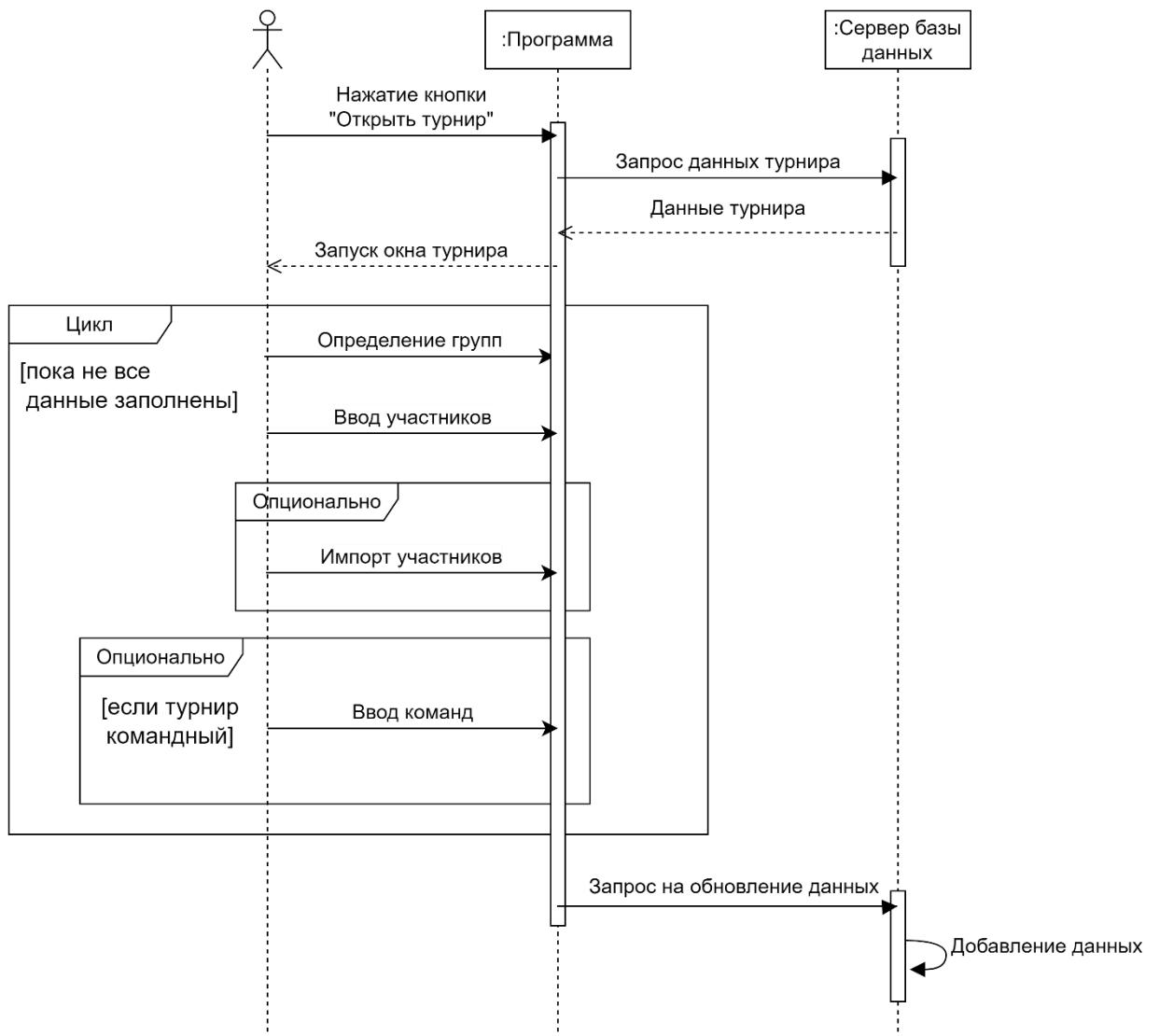


Рисунок 2.13 – Диаграмма последовательности "Добавление данных турнира"

2.4.1.3 Последовательность проведения жеребьевки турнира

Для начала проведения жеребьевки, пользователь, находясь на странице списка пар, нажимает на кнопку создания нового тура, после чего запускается алгоритм жеребьевки. Если алгоритм отработал успешно, то в базу записываются пары игроков, играющие в новом туре, обновляется таблица списка пар.

После добавления нового тура пользователь может изменять результаты встреч игроков, по мере заполнения которых, отправляются запросы в базу на обновление, обновляются рейтинг-листы.

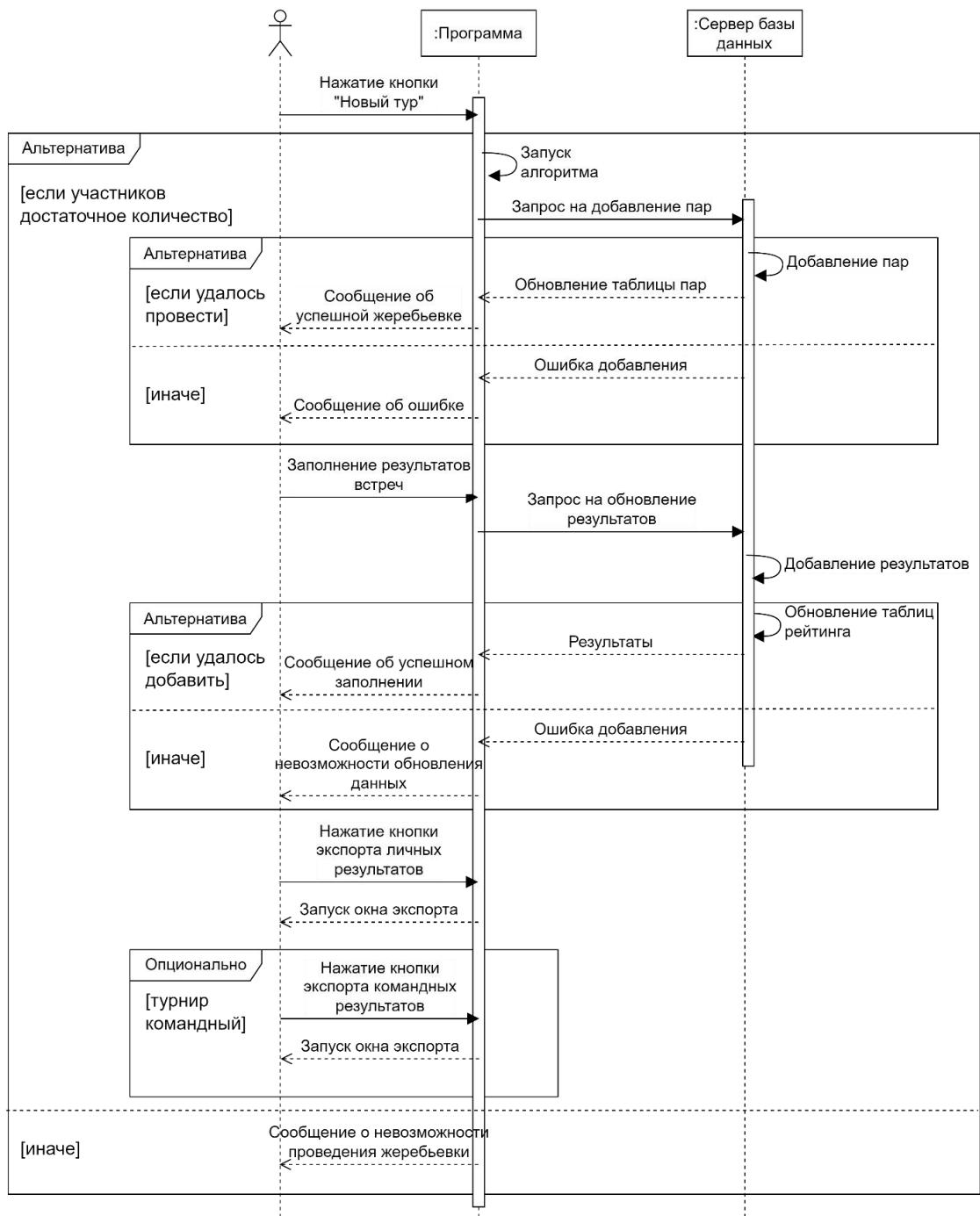


Рисунок 2.14 – Диаграмма последовательности "Проведение жеребьевки"

Так как было принято решение вести разработку приложения для настольных компьютеров, и по требованию заказчика о необходимости работы приложения на операционной системе Windows 10, то наилучшим решением было принято взять за основу язык программирования C# и фреймворк для разработки настольных приложений WPF.

Исходя из данного выбора, пользовательский интерфейс проектировался в среде разработки Microsoft Visual Studio.

2.4.1.4 Главное окно

Исходя из спроектированных диаграмм последовательностей, было спроектировано главное окно приложения, содержащее список турниров, доступных пользователю (см. рисунок 2.15).

Помимо списка турниров, пользователю сразу доступны и другие вкладки:

1. Профиль – страница с персональной информацией о пользователе (см. рисунок 2.16).
2. Дерево турниров – древовидное представление списка турниров, позволяющее редактировать содержимое турниров без необходимости их открытия (см. рисунок 2.17).

Создать турнир Печать Справка							
Профиль	Список турниров	Дерево турниров					
	Название	Дата	Туры	Место	Дата создания	Дата изменения	
Открыть	Рождественские встречи	3/5/2023	7	-	2/22/2023	3/11/2023	Редактировать Удалить
Открыть	Новогодний командный	2/22/2023	7	-	2/22/2023	2/22/2023	Редактировать Удалить
Открыть	Сельские игры	12/12/2023	7	Пермь	2/22/2023	3/3/2023	Редактировать Удалить
Открыть	Чемпионат России по быстрым шахматам	4/4/2023	9	Россия, Московская область, Москва	2/22/2023	3/4/2023	Редактировать Удалить

Рисунок 2.15 – Главное окно

На странице профиля пользователь, при необходимости, может отредактировать свои данные, если это необходимо, посмотреть информацию о дате регистрации, о количестве свободных мест для турниров.

Добро пожаловать, Пользователь

Здесь вы можете просмотреть ваши данные и изменить их при необходимости.

Фамилия

Имя

Отчество

Email

Свободных мест для турниров: 50

Дата регистрации: 11.11.2011

Рисунок 2.16 – Страница профиля пользователя

На странице древовидного списка турниров пользователь может, не открывая турниры, посмотреть на команды и их участников, при необходимости также отредактировать их посредством вызова контекстного меню вершины дерева.

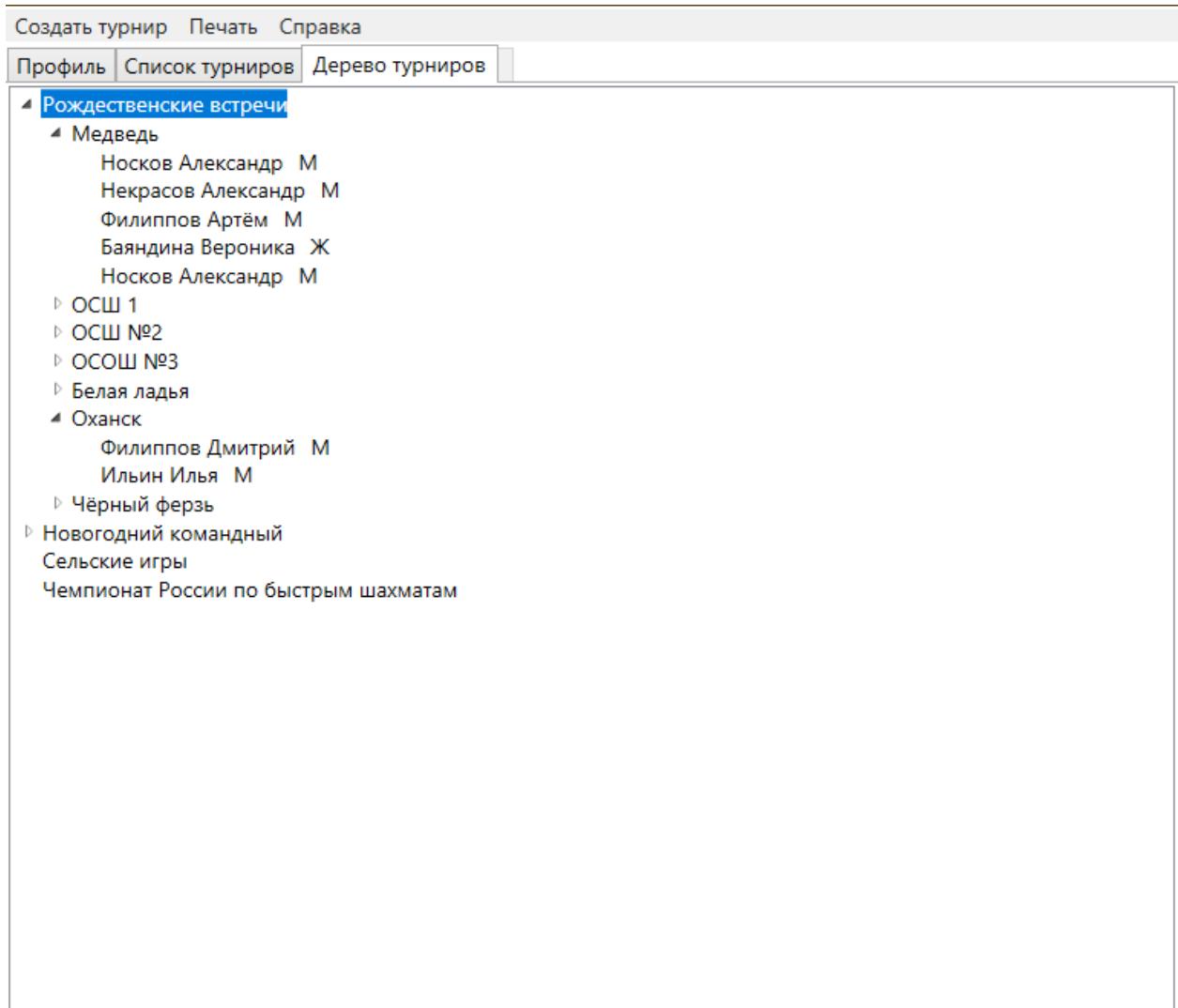


Рисунок 2.17 – Древовидный список турниров

Также доступна панель меню, содержащая следующие элементы:

1. Создать турнир – при нажатии вызывает окно для создания нового турнира (см. рисунок 2.18).
2. Печать – при нажатии вызывает выпадающий список с пунктами, позволяющими выбрать окно для печати.
3. Справка – при нажатии вызывает окно информации о приложении.

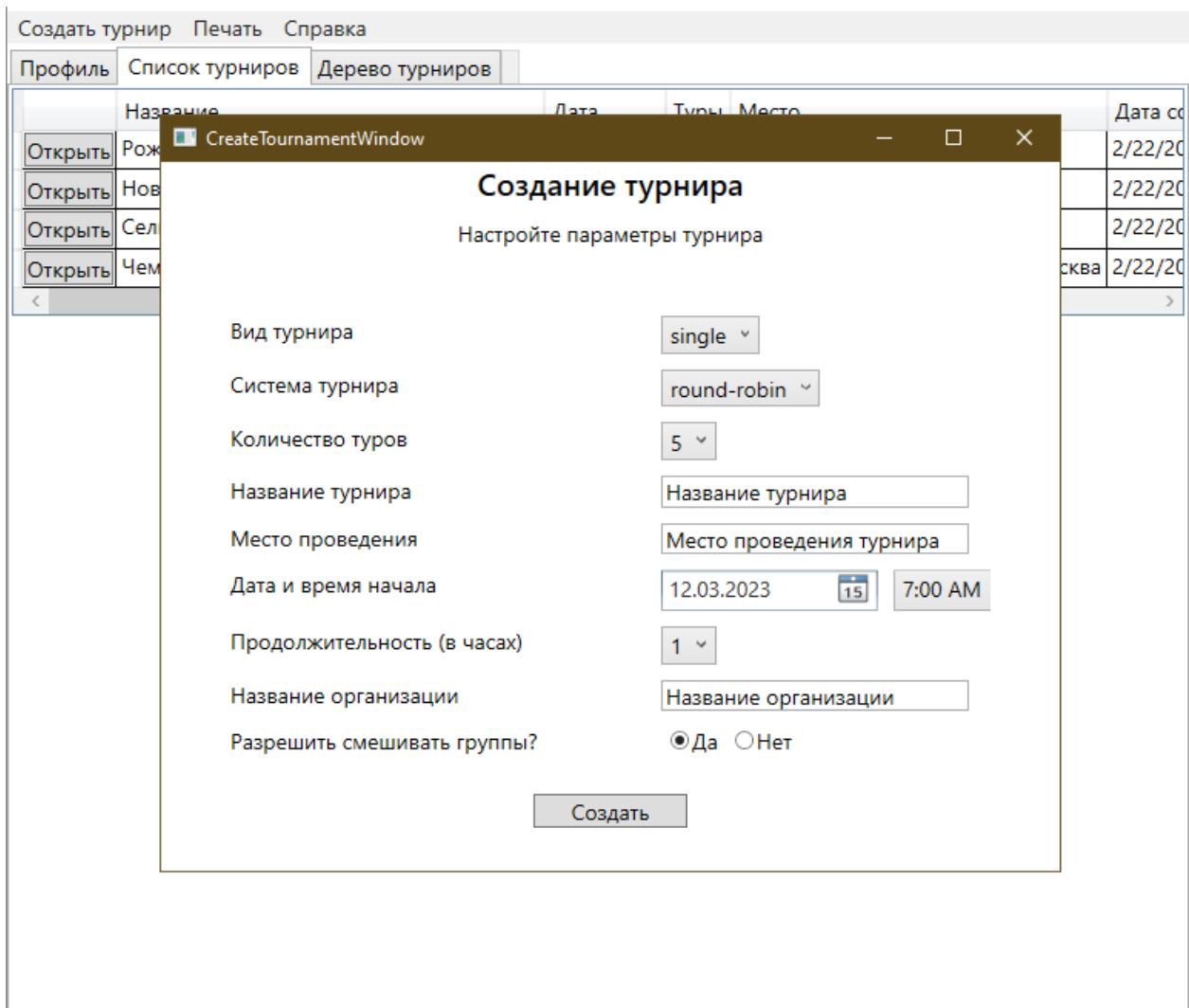


Рисунок 2.18 – Окно создания турнира

2.4.1.5 Работа с турниром

Для того, чтобы начать работу с турниром, пользователь может создать турнир или открыть существующий, содержащийся в списке турниров. После открытия турнира, пользователь перенаправляется на новую вкладку с этим турниром (см. рисунок 2.19).

В этой вкладке становятся доступными другие вкладки, относящиеся к открытому турниру:

1. Список участников – вкладка с таблицей участников турнира (см. рисунок 2.19).
2. Список команд – древовидный список команд и их участников (см. рисунок 2.20).
3. Список групп – древовидный список групп и их участников (см. рисунок 2.21).
4. Список пар – таблица, представляющая собой пары игроков, играющие между собой в турнире для каждого тура (см. рисунок 2.22).
5. Рейтинг-лист – таблица, представляющая собой список участников, отсортированный по убыванию количества очков и коэффициентов (см. рисунок 2.23).

2.4.1.6 Управление участниками

Управление участниками производится на странице списка участников. Для добавления игрока присутствует соответствующая кнопка, по нажатии которой, создается пустая запись в таблице, которую может отредактировать пользователь.

В строке, соответствующей участнику, содержатся следующие данные:

1. Фамилия – текстовое поле.
2. Имя – текстовое поле.
3. Пол – выпадающий список.
4. Атрибут (для выделения участника среди остальных) – текстовое поле.
5. Команда – выпадающий список.
6. Группа – выпадающий список.
7. Год рождения – выпадающий список.
8. Активен – поле с отметкой.
9. Кнопка удаления.

The screenshot shows a software window titled 'Создать турнир' (Create Tournament). The top menu bar includes 'Создать турнир', 'Печать' (Print), and 'Справка' (Help). Below the menu is a navigation bar with tabs: 'Профиль' (Profile), 'Список турниров' (List of Tournaments), 'Дерево турниров' (Tournament Tree), and 'Рождественские встречи' (Christmas Meetings). The current tab is 'Список участников' (List of Participants). A sub-menu bar below it contains 'Добавить игрока' (Add Player), 'Список команд' (List of Teams), 'Список групп' (List of Groups), 'Список пар' (List of Pairs), and 'Рейтинг-лист' (Rating List). The main area is a table with the following data:

	Фамилия	Имя	Пол	Атрибут	Команда	Группа	Год рождения	Активен	Удалить
1	Носков	Александр	М	3MS	Медведь		2006	<input checked="" type="checkbox"/>	Удалить
2	Некрасов	Александр	М	1MS	Медведь		2003	<input checked="" type="checkbox"/>	Удалить
3	Филиппов	Артём	М	1MS	Медведь		2004	<input checked="" type="checkbox"/>	Удалить
4	Баяндина	Вероника	Ж	2FS	Медведь		2008	<input checked="" type="checkbox"/>	Удалить
5	Носков	Александр	М	2MS	Медведь		2008	<input checked="" type="checkbox"/>	Удалить
6	Копытов	Илья	М	1MS	ОСШ 1		2003	<input checked="" type="checkbox"/>	Удалить
7	Носкова	Вероника	Ж	3FS	ОСШ 1	D13	2010	<input checked="" type="checkbox"/>	Удалить
8	Баяндин	Дмитрий	М	3MS	ОСШ 1	M13	2009	<input checked="" type="checkbox"/>	Удалить
9	Филиппов	Евгений	М	1MS	ОСШ 1		2005	<input checked="" type="checkbox"/>	Удалить
10	Бояршинов	Дмитрий	М	1MS	ОСШ 1		2005	<input checked="" type="checkbox"/>	Удалить
11	Бояршинов	Илья	М	1MS	ОСШ №2		2004	<input checked="" type="checkbox"/>	Удалить
12	Иванов	Дмитрий	М	1MS	ОСШ №2		2005	<input checked="" type="checkbox"/>	Удалить
13	Копытов	Александр	М	1MS	ОСОШ №3		2004	<input checked="" type="checkbox"/>	Удалить
14	Носков	Илья	М	2MS	ОСОШ №3		2005	<input checked="" type="checkbox"/>	Удалить
15	Филиппов	Филипп	М	1MS	ОСОШ №3		2006	<input checked="" type="checkbox"/>	Удалить
16	Иванов	Иван	М	1MS	Белая ладья		2003	<input checked="" type="checkbox"/>	Удалить
17	Петров	Петр	М	2MS	Белая ладья		2004	<input checked="" type="checkbox"/>	Удалить
18	Сидоров	Сидор	М	2MS	Белая ладья		2005	<input checked="" type="checkbox"/>	Удалить
19	Смирнов	Сергей	М	1MS	Белая ладья		2006	<input checked="" type="checkbox"/>	Удалить
20	Смирнова	Светлана	Ж	2FS	Белая ладья	D16	2007	<input checked="" type="checkbox"/>	Удалить
21	Смирнова	Анна	Ж	2FS	Белая ладья		2008	<input checked="" type="checkbox"/>	Удалить

Рисунок 2.19 – Страница турнира (список участников)

2.4.1.7 Управление командами

На странице списка команд пользователю предоставляется возможность управления командами (добавление, удаление, редактирование, просмотр членов команд).

Древовидная структура позволяет удобно просмотреть не только команды, но также и их участников, без необходимости постоянного переключения между вкладками.

The screenshot shows a web-based application interface for managing tournaments. At the top, there's a menu bar with links like 'Создать турнир' (Create tournament), 'Печать' (Print), 'Справка' (Help), 'Профиль' (Profile), 'Список турниров' (List of tournaments), 'Дерево турниров' (Tournament tree), and 'Рождественские встречи' (Christmas meetings). Below the menu, a navigation bar includes tabs for 'Список участников' (List of participants), 'Список команд' (List of teams), 'Список групп' (List of groups), 'Список пар' (List of pairs), and 'Рейтинг-лист' (Rating list). A prominent button 'Добавить команду' (Add team) is located on the left. The main content area displays a hierarchical list of teams:

- ▲ Медведь [Удалить] [Редактировать]
 - Носков Александр
 - Некрасов Александр
 - Филиппов Артём
 - Баяндина Вероника
 - Носков Александр
- ▲ ОСШ 1 [Удалить] [Редактировать]
 - Копытов Илья
 - Носкова Вероника
 - Баяндин Дмитрий
 - Филиппов Евгений
 - Бояршинов Дмитрий
- ▷ ОСШ №2 [Удалить] [Редактировать]
- ▷ Оханск [Удалить] [Редактировать]
- ▷ ОСОШ №3 [Удалить] [Редактировать]
- ▷ Белая ладья [Удалить] [Редактировать]
- ▷ Чёрный ферзь [Удалить] [Редактировать]

Рисунок 2.20 – Список команд

Добавление команды

Название команды:

Рисунок 2.21 – Окно добавления команды

2.4.1.8 Управление группами

Также как и для команд, управление группами осуществляется посредством древовидного списка. Так как в одной команде могут быть участники из разных групп, то совмещение списков древовидной структуры невозможно, но оно и не критично, так как ситуации, когда необходимо одновременно редактировать и группу, и команду, редки.

При добавлении группы пользователю предлагается заполнить следующие поля: полное название группы, идентификатор группы – сокращение названия. В основном используется идентификатор группы, при этом во время выбора группы в списке участников, этот идентификатор заменяется на полное название (см. рисунок 27).

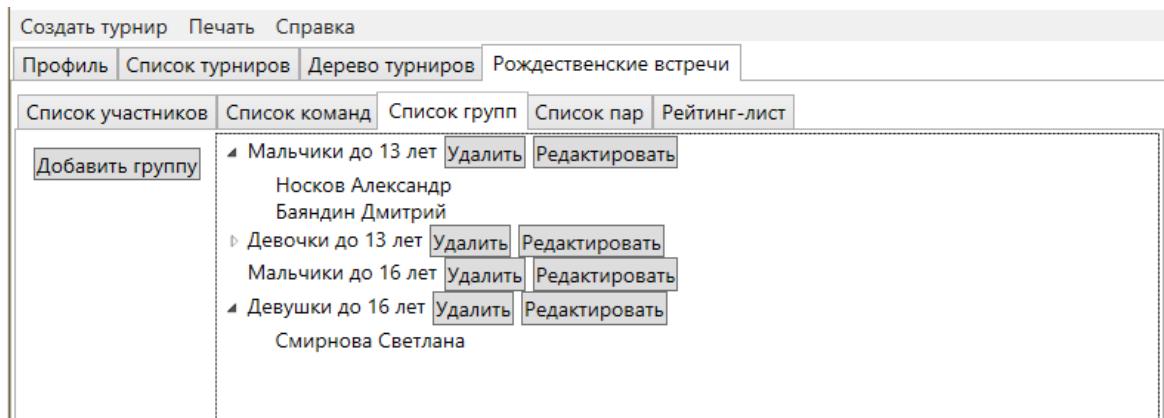


Рисунок 2.22 – Список групп

Добавление группы

Полное название группы:

Идентификатор группы:

Добавить

Рисунок 2.23 – Окно добавления группы

Группа	Г
Д13	
Мальчики до 13 лет	Д13
Мальчики до 13 лет	M13
Девочки до 13 лет	
Мальчики до 16 лет	
Девушки до 16 лет	

Рисунок 2.24 – Пример использования названия и идентификатора группы

2.4.1.9 Управление списком пар

На данной странице пользователю доступны данные встреч игроков в рамках турнира. При необходимости пользователь может посмотреть и отредактировать предыдущие туры.

По нажатию кнопки начала нового тура, появятся пары нового тура, текущий тур изменится на новый.

The screenshot shows a software interface for managing chess tournaments. At the top, there are menu options: 'Создать турнир' (Create Tournament), 'Печать' (Print), and 'Справка' (Help). Below the menu is a navigation bar with tabs: 'Профиль' (Profile), 'Список турниров' (List of Tournaments), 'Дерево турниров' (Tournament Tree), and 'Рождественские встречи' (Christmas Meetings). The 'Рождественские встречи' tab is selected. Underneath the navigation bar is another row of tabs: 'Список участников' (List of Participants), 'Список команд' (List of Teams), 'Список групп' (List of Groups), 'Список пар' (List of Pairs), and 'Рейтинг-лист' (Rating List). The 'Список пар' tab is selected. A message 'Выбранный тур: 4, текущий: 4' (Selected round: 4, current: 4) is displayed. On the left, there are three buttons: 'Начать новый' (Start new), 'Предыдущий' (Previous), and 'Следующий' (Next). To the right is a table showing the list of pairs for the current round (Round 4). The table has columns: 'Белые' (White), 'Очки' (Points), 'Результат' (Result), 'Черные' (Black), and 'Очки' (Points). The data is as follows:

	Белые	Очки	Результат	Черные	Очки
1	Баяндина Вероника	0	0 – 0	Филиппов Артём	0
2	Носков Александр	1	0 – 0	Некрасов Александр	2
3	Копытов Илья	1	0 – 0	Носков Александр	0
4	Носкова Вероника	0.5	0 – 0	Ильин Илья	0
5	Филиппов Евгений	0	0 – 0	Петров Петр	0.5
6	Бояршинов Илья	0	0 – 0	Бояршинов Роберт	0
7	Иванов Дмитрий	0.5	0 – 0	Смирнова Светлана	0
8	Носков Илья	1	0 – 0	Сидоров Сидор	0.5
9	Филиппов Филипп	0	0 – 0	Петров Петр	1
10	Бояршинов Дмитрий	1	0 – 0	Иванов Иван	0
11	Баяндин Дмитрий	1	0 – 1	Сидор Сидор	1
12	Филиппов Дмитрий	0.5	0.5 – 0.5	Смирнова Анна	1.5
13	Копытов Александр	0	0 – 1	Смирнов Сергей	1

Рисунок 2.25 – Список пар

2.4.1.10 Просмотр рейтинг-листа

На этой вкладке пользователь не может ничего редактировать, так как это вкладка-отчет, представляющий собой таблицу с информацией об участниках, которые отсортированы по очкам и коэффициентам по убыванию, образуя тем самым рейтинговую таблицу, где самый верхний участник занимает первое место.

The screenshot shows a software interface for managing chess tournaments. At the top, there are menu options: Создать турнир (Create Tournament), Печать (Print), Справка (Help). Below the menu is a navigation bar with tabs: Профиль (Profile), Список турниров (List of Tournaments), Дерево турниров (Tournament Tree), Рождественские встречи (Christmas Meetings). The current tab is Рейтинг-лист (Rating List). Underneath the tabs are sub-tabs: Список участников (List of Participants), Список команд (List of Teams), Список групп (List of Groups), Список пар (List of Pairs), and Рейтинг-лист (Rating List). The main content area is titled "Рейтинг-лист после 4 тура" (Rating List after 4 rounds). It displays a table with 20 rows, each representing a participant's performance. The columns are: № (Rank), Имя (Name), Команда (Team), Победы (Wins), Ничьи (Draws), Проигрыши (Losses), Очки (Points), Коэффициент 1 (Coefficient 1), and Коэффициент 2 (Coefficient 2). The table shows that the top performer is Nekrasov Alexander from Medvedy with 4 wins, 0 draws, and 1 loss, totaling 7 points. The bottom performer is Boyarshevich Ilya from OCSH №2 with 0 wins, 0 draws, and 4 losses, totaling 0 points.

№	Имя	Команда	Победы	Ничьи	Проигрыши	Очки	Коэффициент 1	Коэффициент 2
1	Некрасов Александр	Медведь	4	0	1	2	0.00	0.00
2	Смирнова Анна	Белая ладья	4	1	0	1.5	0.00	0.00
3	Носков Александр	Медведь	1	0	1	1	0.00	0.00
4	Копытов Илья	ОСШ 1	1	0	3	1	0.00	0.00
5	Баяндин Дмитрий	ОСШ 1	1	0	4	1	0.00	0.00
6	Носков Илья	ОСОШ №3	1	0	1	1	0.00	0.00
7	Бояршинов Дмитрий	ОСШ 1	3	0	1	1	0.00	0.00
8	Петров Петр	Белая ладья	4	0	0	1	0.00	0.00
9	Смирнов Сергей	Белая ладья	4	0	0	1	0.00	0.00
10	Сидо Сидор	Чёрный ферзь	4	0	2	1	0.00	0.00
11	Носкова Вероника	ОСШ 1	0	1	3	0.5	0.00	0.00
12	Филиппов Дмитрий	Оханская	0	1	3	0.5	0.00	0.00
13	Иванов Дмитрий	ОСШ №2	0	1	3	0.5	0.00	0.00
14	Сидоров Сидор	Белая ладья	3	1	0	0.5	0.00	0.00
15	Петров Петр	Чёрный ферзь	3	1	0	0.5	0.00	0.00
16	Носков Александр	Медведь	1	0	1	0	0.00	0.00
17	Филиппов Артём	Медведь	1	0	3	0	0.00	0.00
18	Баяндина Вероника	Медведь	1	0	1	0	0.00	0.00
19	Филиппов Евгений	ОСШ 1	0	0	4	0	0.00	0.00
20	Бояршинов Илья	ОСШ №2	0	0	1	0	0.00	0.00

Рисунок 2.26 – Рейтинг-лист

2.4.2 Проектирование классов

2.4.2.1 Сущности

Проектирование сущностей предметной области было проведено исходя из проектированной базы данных. Выполнено проектирование в нотации UML (см. рисунок 2.27).

Каждая сущность сопоставлена с соответствующей ей таблице и служит в качестве DTO (data transfer object) – объектом, переносящим данные между процессами.

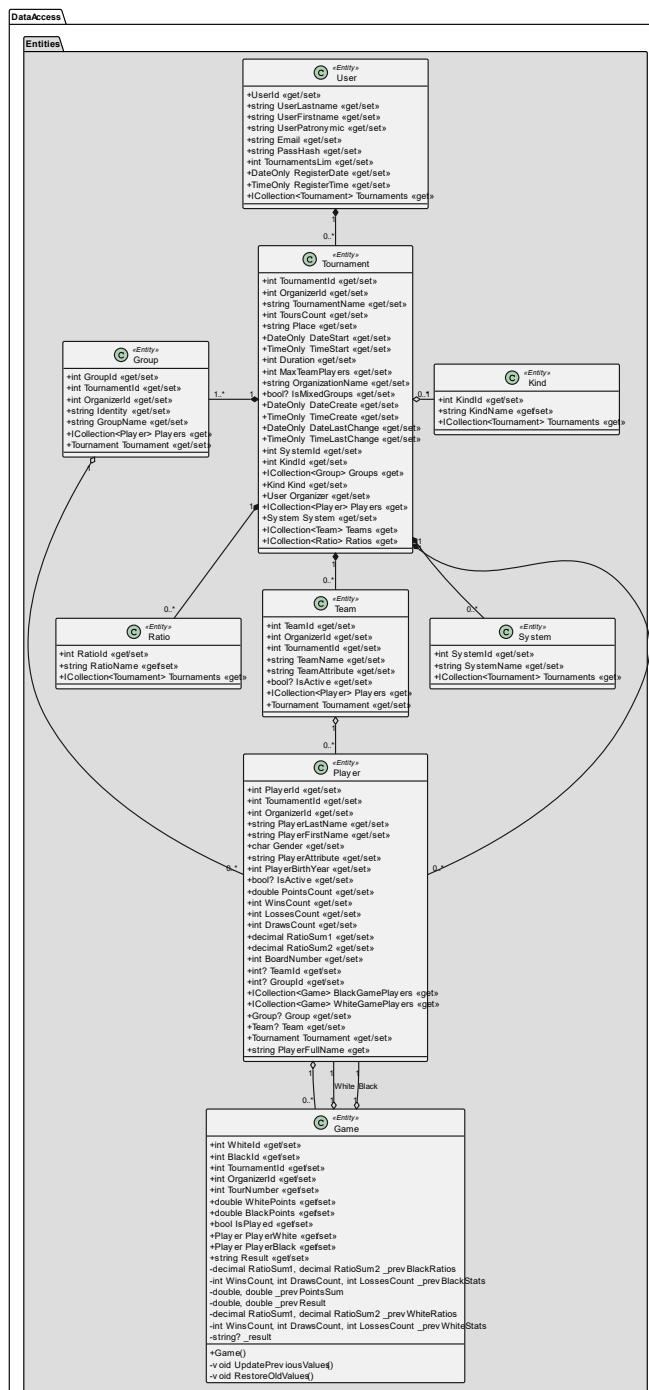


Рисунок 2.27 – Сущности предметной области

2.4.2.2 Взаимодействие с данными

Так как было принято решение использовать в качестве связи с базой данных ORM Entity Framework, то структура проекта организации данных выполнена в виде реализации паттерна «Репозиторий», так как доступ к этим сущностям предоставляется с помощью класса ChessTourContext, наследуемого от DbContext (см. рисунок 2.28).

Каждая коллекция типа DbSet служит хранилищем данных таблицы, соответствующей сущности, содержащейся в коллекции.

Также в этом классе содержатся приватные методы конфигурации сущностей, в которых происходит настройка связи между сущностями и соответствующим им таблицам в базе данных.



Рисунок 2.28 – Контекст данных и сущности

2.4.2.3 Получение данных

Для организации получения данных из базы необходимо реализовать соответствующий интерфейс и класс, реализующий его.

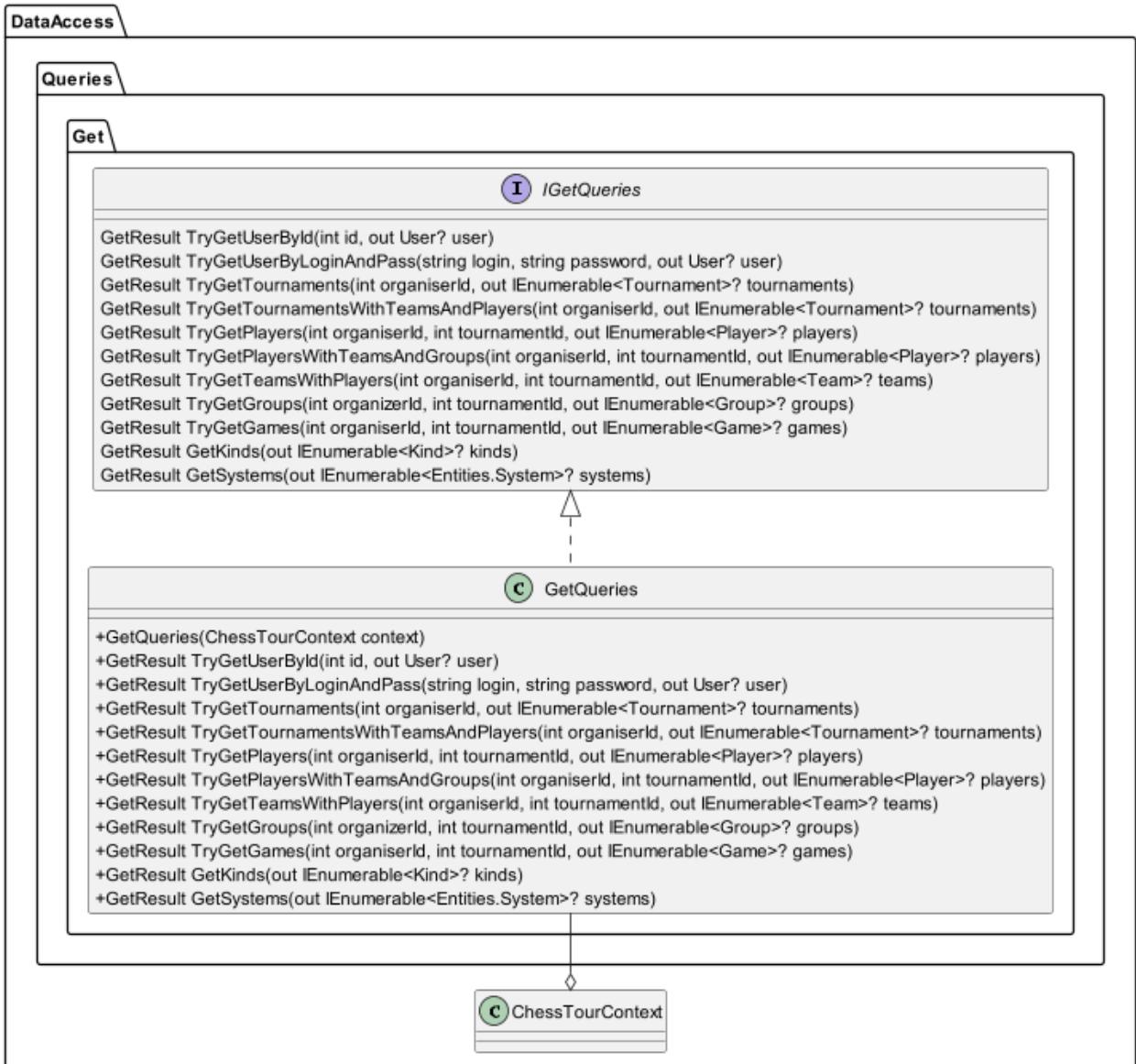


Рисунок 2.29 – Интерфейс и реализация получения данных

2.4.2.4 Вставка данных

Для организации вставки данных в базу необходимо реализовать соответствующий интерфейс и класс, реализующий его.

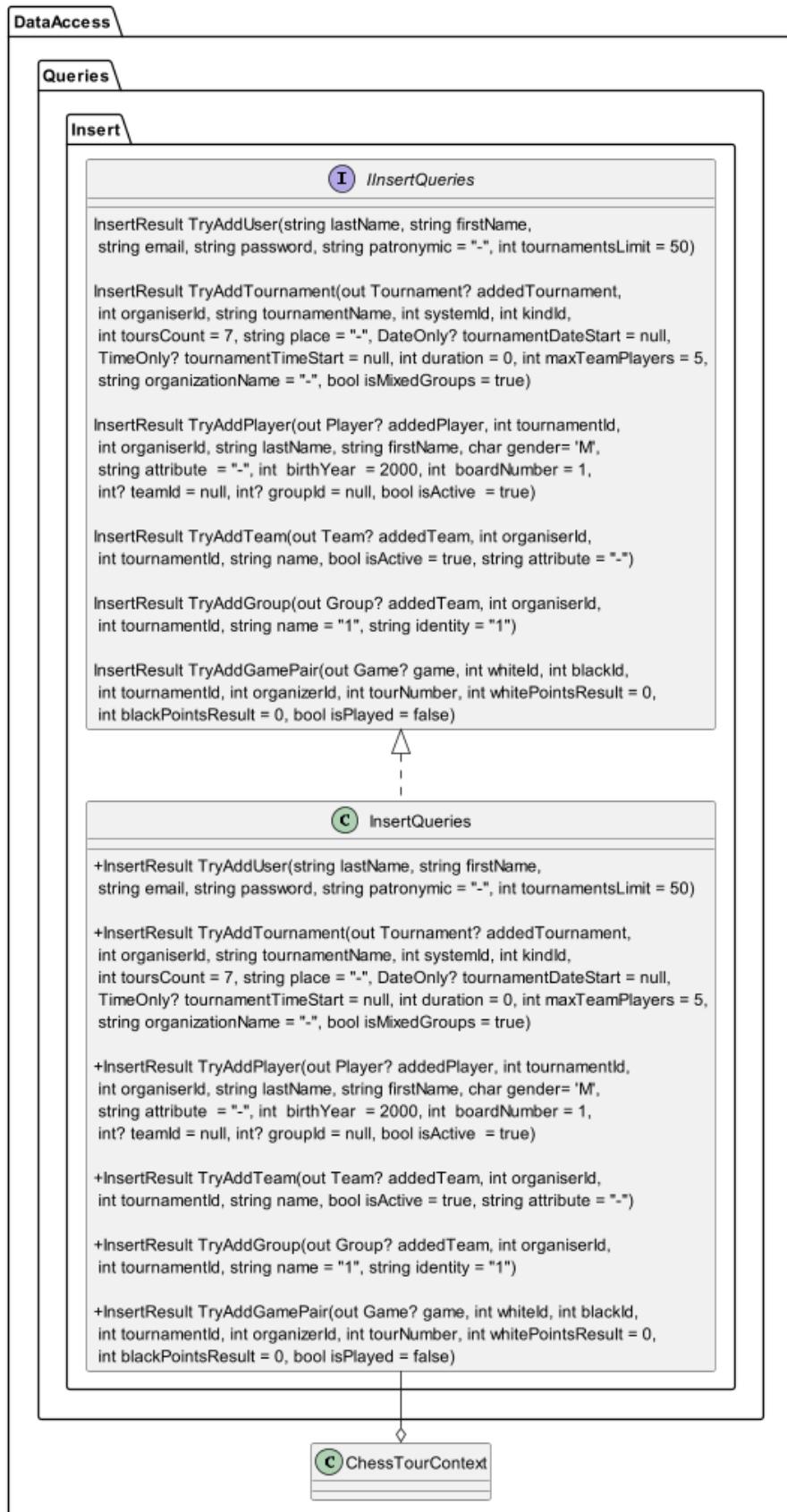


Рисунок 2.30 – Интерфейс и реализация вставки данных

2.4.2.5 Удаление данных

Для организации удаления данных из базы необходимо реализовать соответствующий интерфейс и класс, реализующий его.

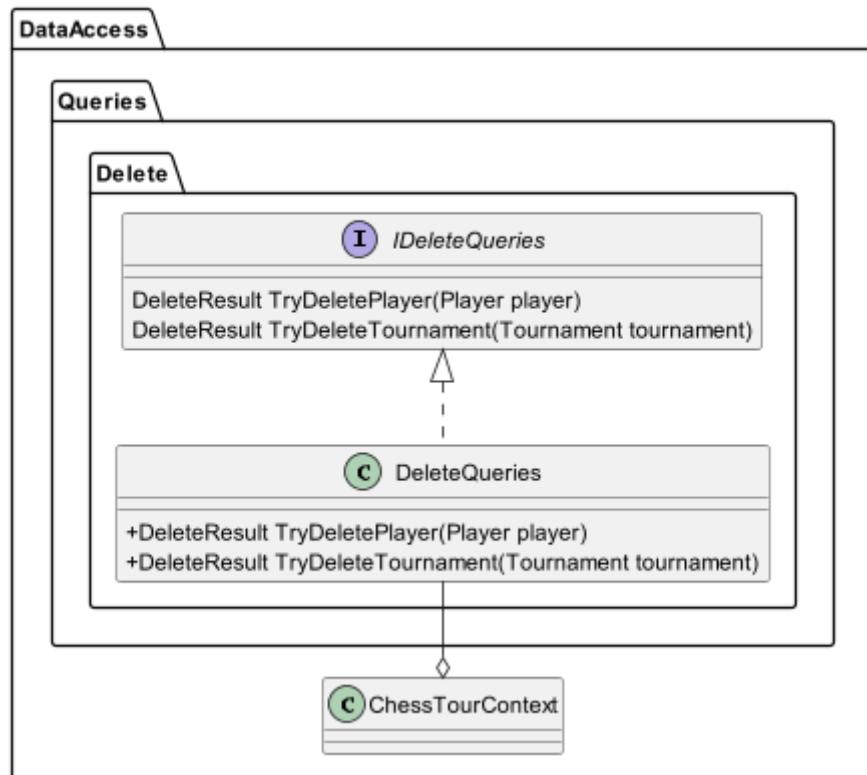


Рисунок 2.31 – Интерфейс и реализация удаления данных

2.4.2.6 Структура классов алгоритмов

Для реализации алгоритмов необходимо описать интерфейс и реализовать его. Этот интерфейс должен описывать абстракцию над алгоритмом жеребьевки турниров.

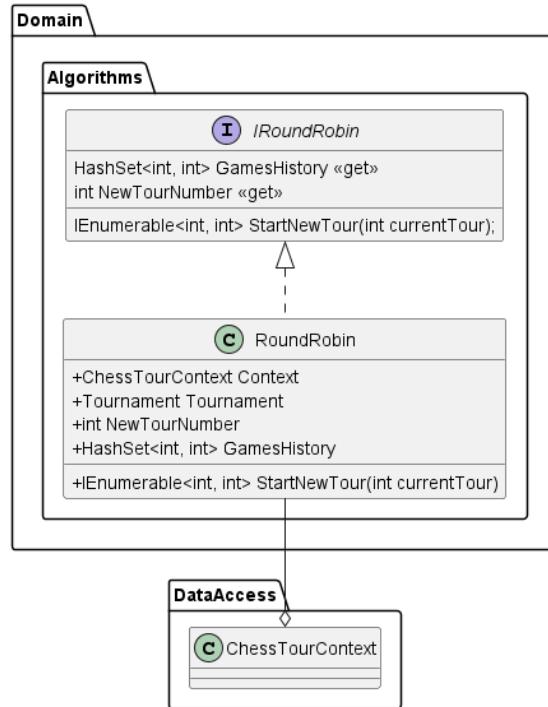


Рисунок 2.32 – Структура классов алгоритмов

2.4.2.7 Структура классов аутентификации

Для реализации аутентификации необходимо реализовать окна регистрации и входа в аккаунт, наследуемые от класса окна. Так как для пользователя должна быть возможность при запуске окна входа зарегистрироваться, то, таким образом, окно регистрации может быть запущено из окна входа, тем самым, оно зависит от окна входа, что показано агрегацией на диаграмме (см. рисунок 2.33).

Для каждого окна должны быть свои модели представления (view model), содержащие данные полей окон (логин, пароль и др.), а также команды для выполнения входа или регистрации.

Классы моделей представления должны быть унаследованы от базового абстрактного класса модели представления, делегирующего реализацию интерфейса `INotifyPropertyChanged`.

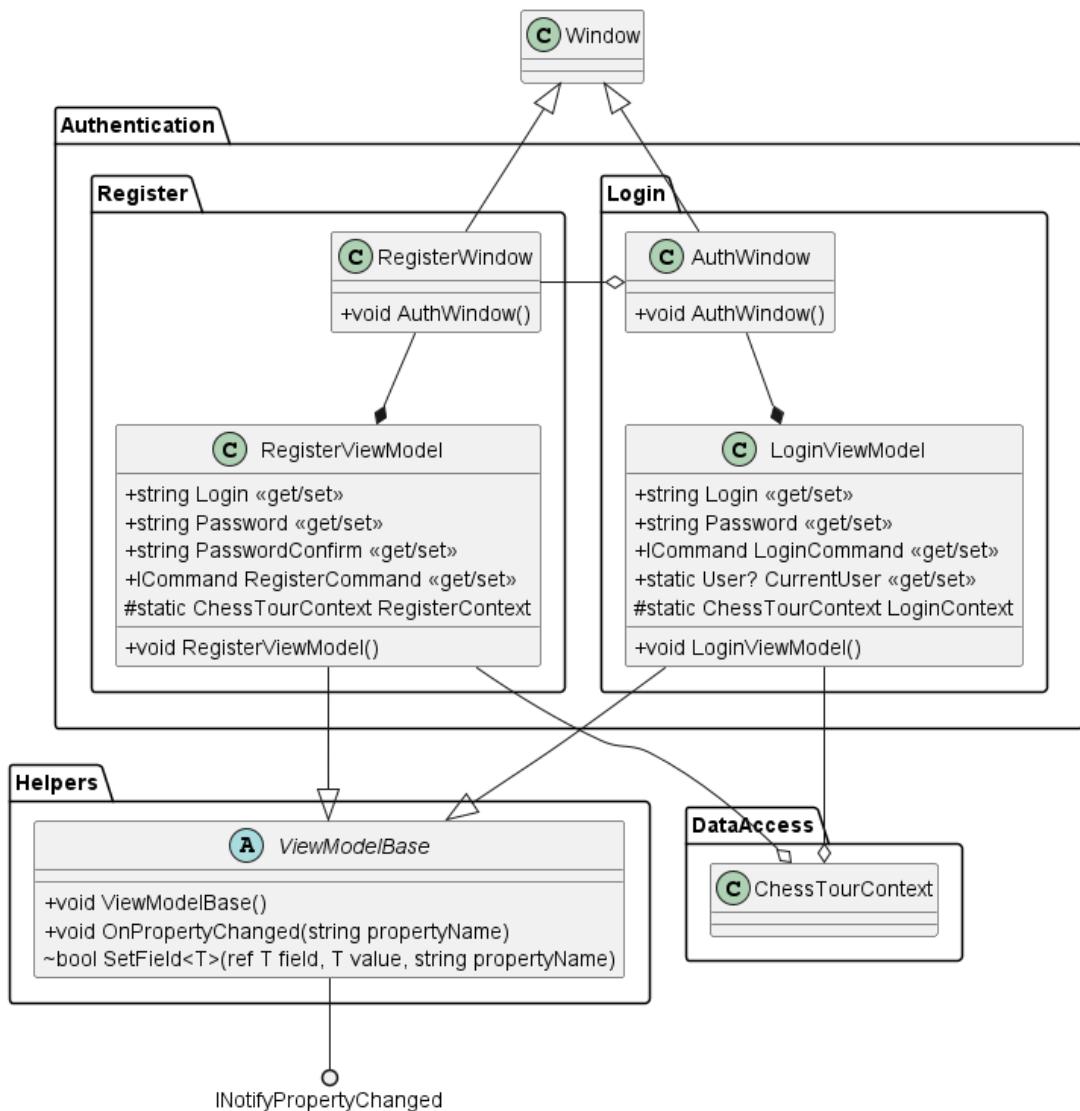


Рисунок 2.33 – Структура классов аутентификации

2.4.2.8 Структура классов управления турнирами

Для реализации управления турнирами необходимо реализовать главное окно, содержащее в себе список турниров и имеющее модель представления, содержащую информацию об этих турнирах.

Дополнительно необходимо реализовать окно для создания нового турнира со своей моделью представления и окно для редактирования турнира.

Дополнительно для удаления турнира окно реализовывать нет необходимости, так как предполагается, что по нажатию кнопки удаления в строке турнира, этот турнир удалится.

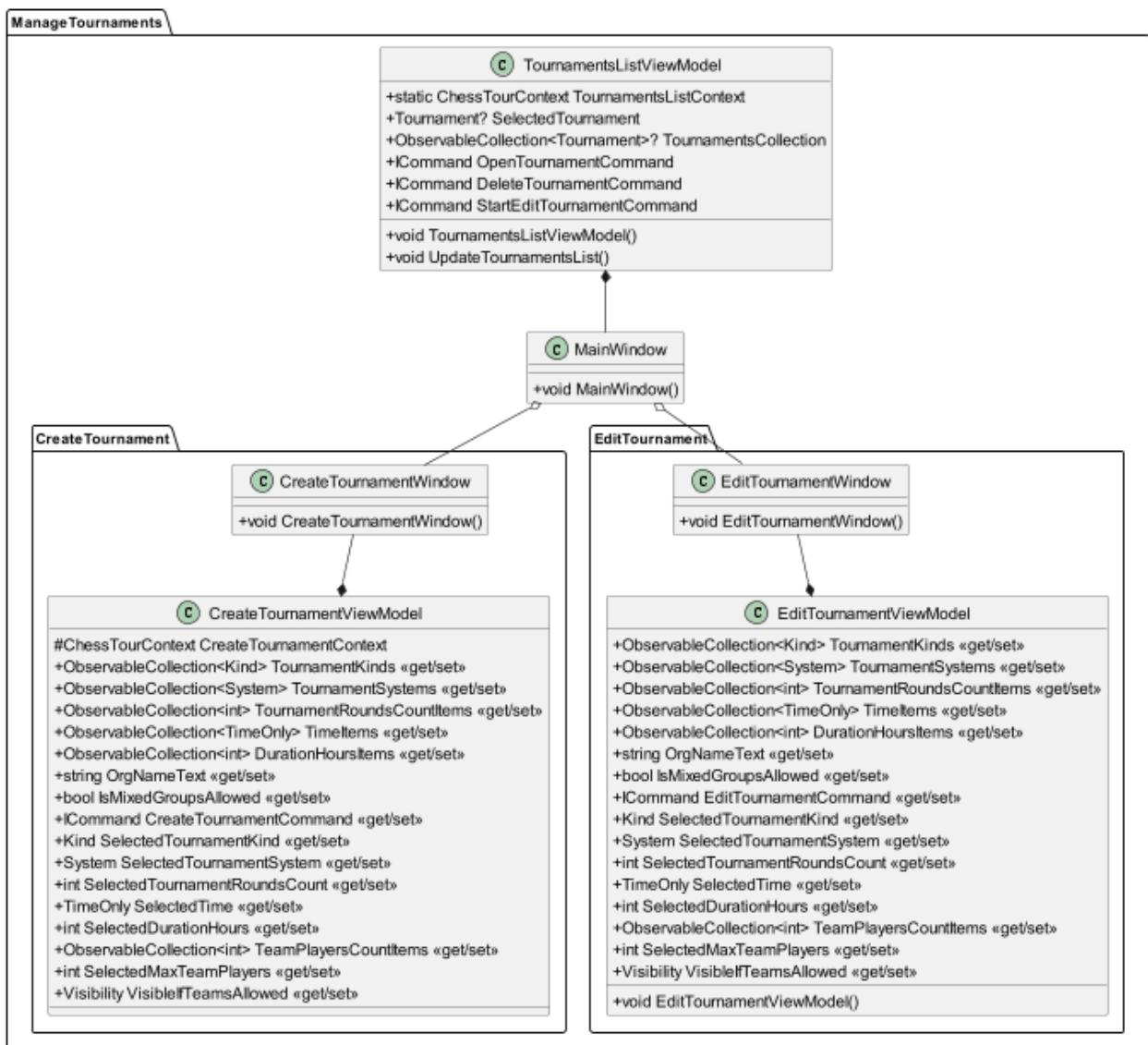


Рисунок 2.34 – Структура классов управления турнирами

2.4.2.9 Структура классов управления игроками

Для реализации управления игроками необходимо реализовать управляющий элемент, представляющий собой таблицу с данными об игроках.

Дополнительно необходимо реализовать окно для добавления и редактирования игрока.

Дополнительно для удаления игрока окно реализовывать нет необходимости, так как предполагается, что по нажатию кнопки удаления в строке игрока, этот игрок удалится.

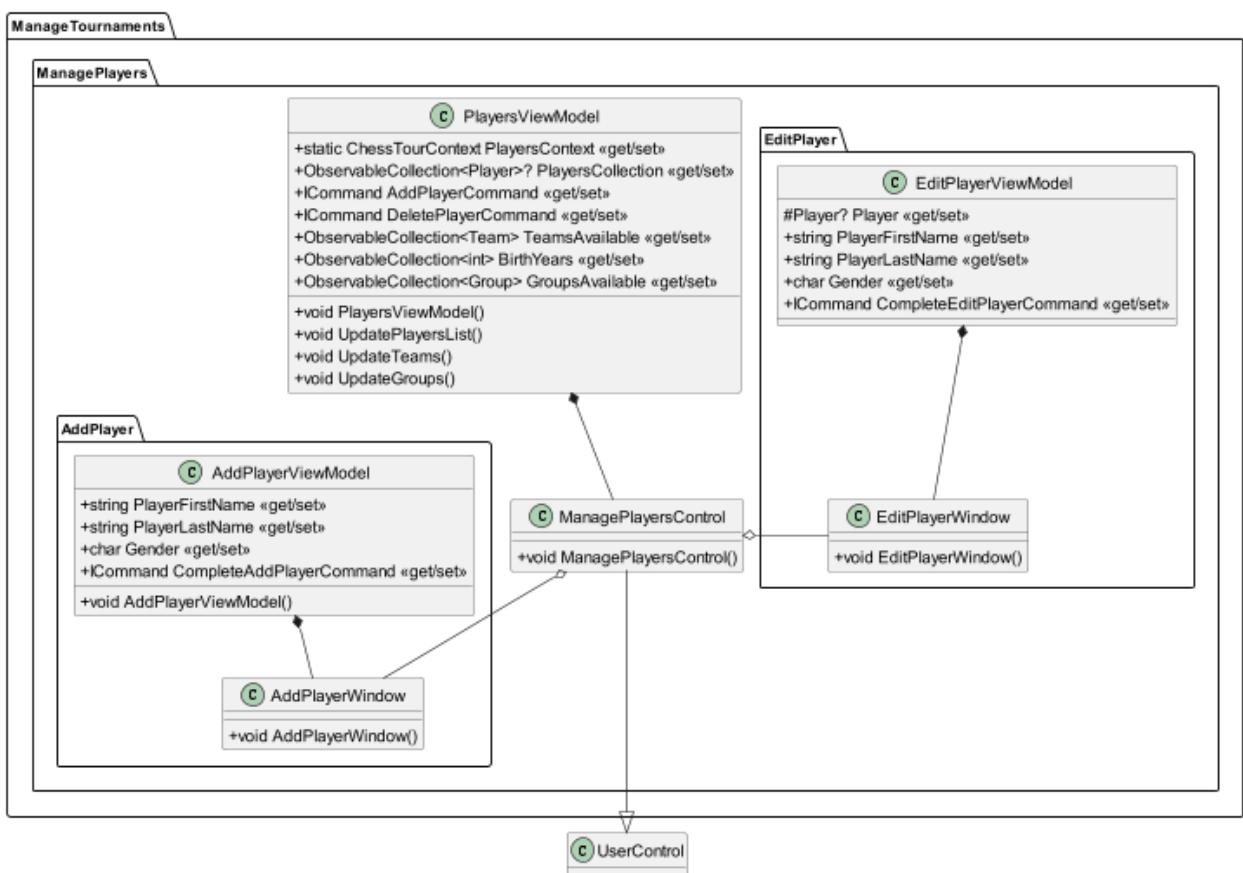


Рисунок 2.35 – Структура классов управления игроками

2.4.2.10 Структура классов просмотра рейтинга

Так как представление рейтинг-листа не редактируется, то кроме самого представления и его модели ничего не требуется реализовать.

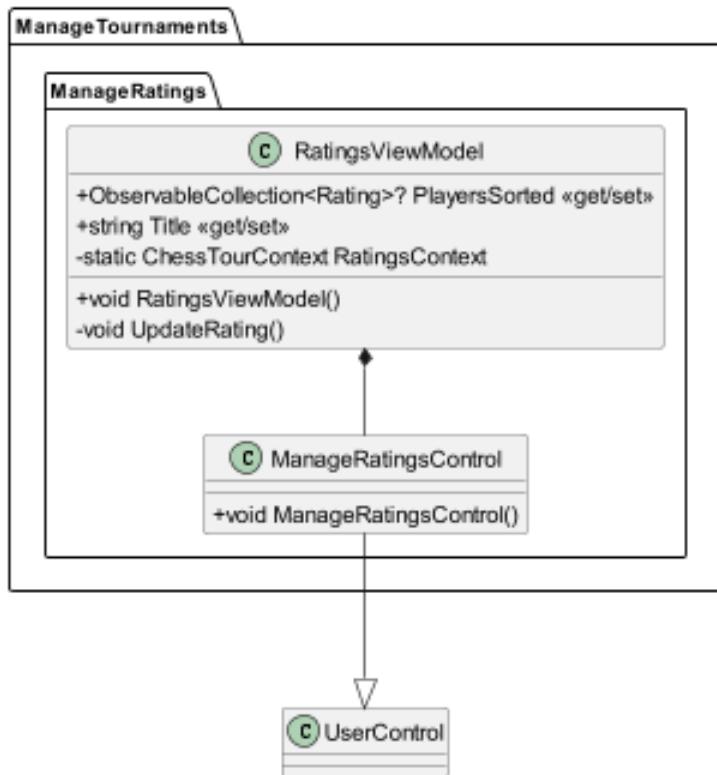


Рисунок 2.36 – Структура классов просмотра рейтинга

2.4.2.11 Структура классов управления командами

Для реализации управления командами необходимо реализовать элемент управления, который связан со своей моделью представления. Также с этой моделью представления допускается связать окно добавления команды.

Для редактирования команды было принято решение о создании отдельной модели представления, так как необходимо хранить и саму команду тоже, помимо редактируемых данных.

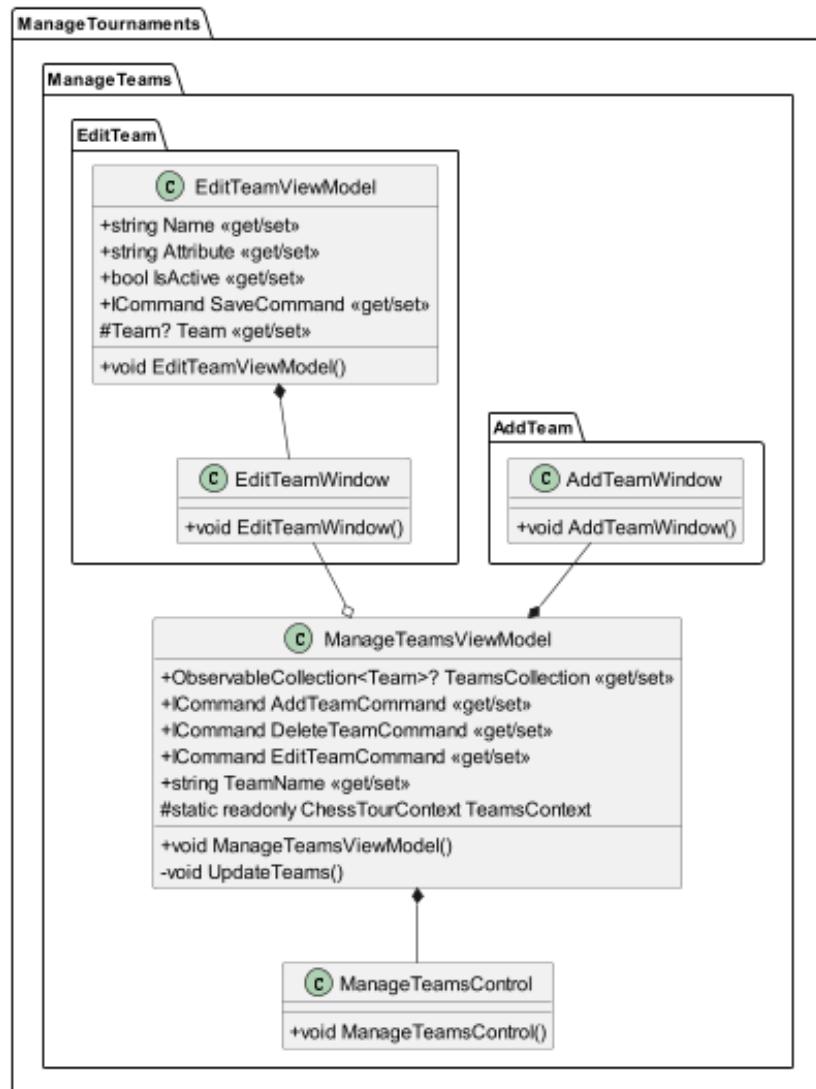


Рисунок 2.37 – Структура классов управления командами

2.4.2.12 Структура классов управления играми

Так как список пар игроков представляет собой таблицу, в которой предполагается только одно редактируемое поле – результат встречи, то было решено не создавать дополнительные окна и модели представления.

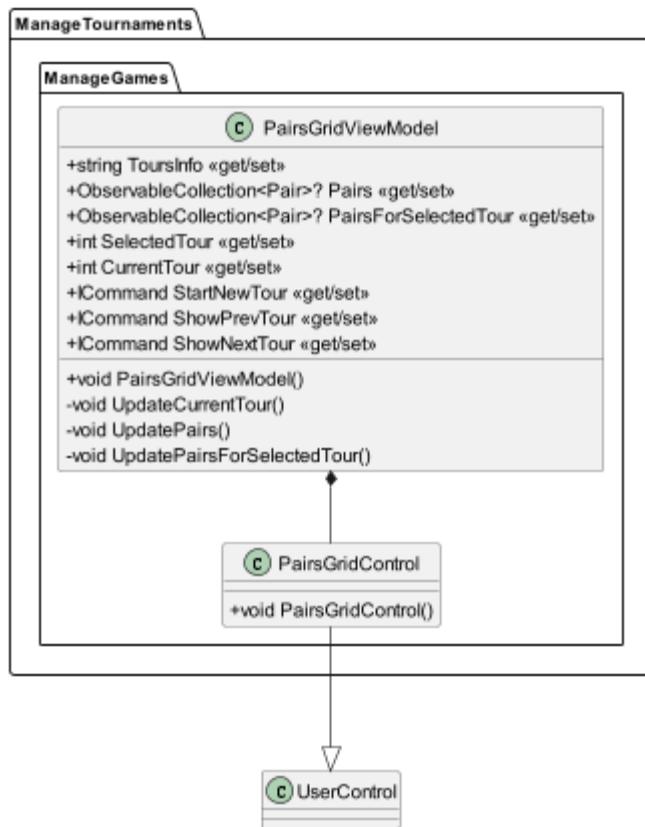


Рисунок 2.38 – Структура классов управления играми

2.4.2.13 Структура классов управления группами

Проектирование данных классов произведено по полной аналогии с проектированием классов управления командами.

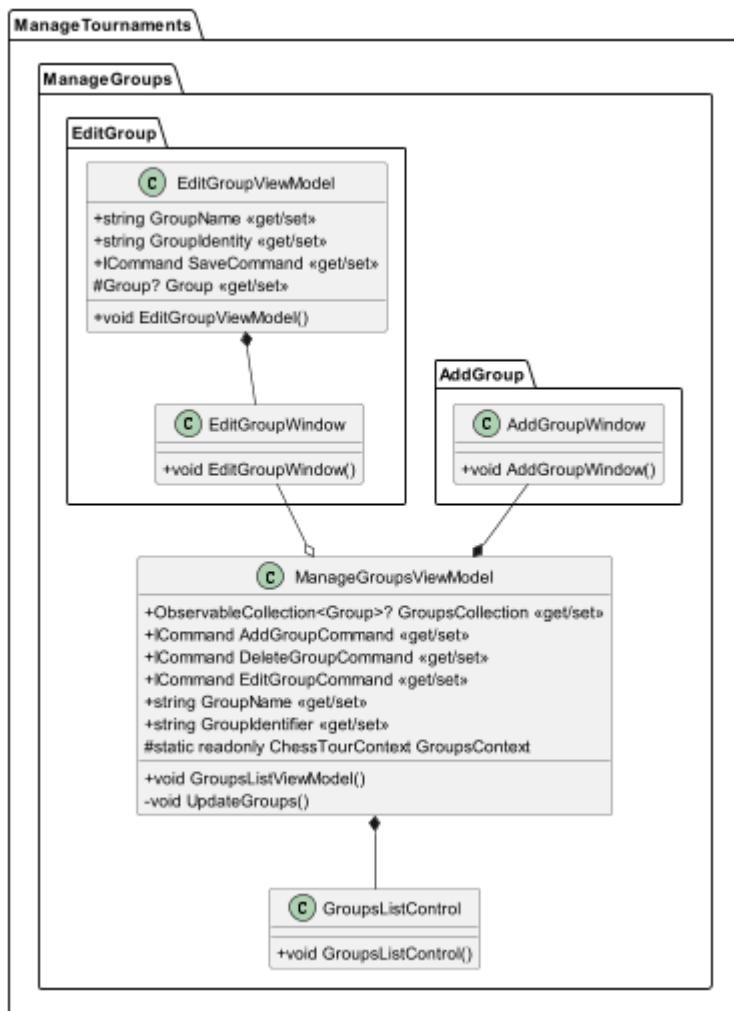


Рисунок 2.39 – Структура классов управления группами

2.4.2.14 Структура управления списком турниров

Управление древовидным списком предполагается реализовывать после реализации всех предыдущих классов, таким образом, для управления данными в этом дереве можно воспользоваться спроектированными командами, так что реализовывать дополнительно ничего не придется.

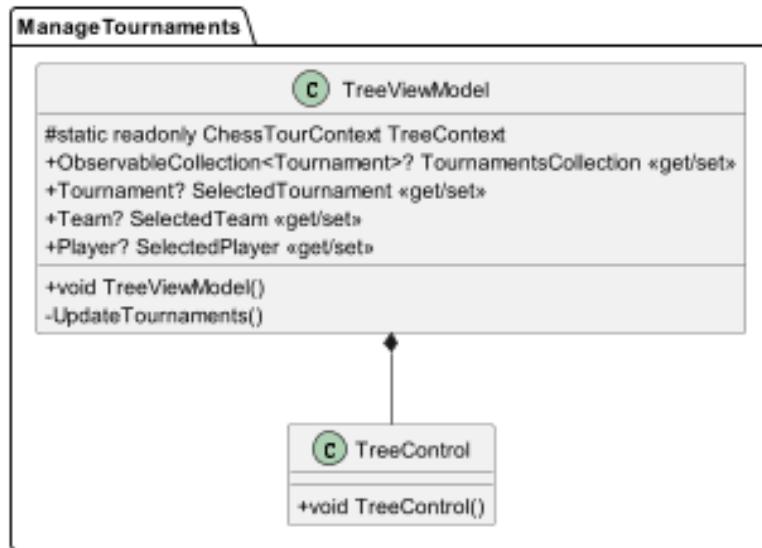


Рисунок 2.40 – Структура управления турниром

2.4.3 Проектирование алгоритмов жеребьевки

2.4.3.1 Проектирование кругового алгоритма

Круговой алгоритм заключается в том, чтобы организовать встречи игроков таким образом, что каждый игрок играет с каждым ровно один раз, при этом должно соблюдаться чередование цветов, которыми играют игроки, например, не должно быть так, что один игрок играл только черным или только белым цветом, в идеальном случае количество партий за белых и за черных должны чередоваться и быть в равном количестве.

Далее приведена диаграмма активности, описывающая данный алгоритм.

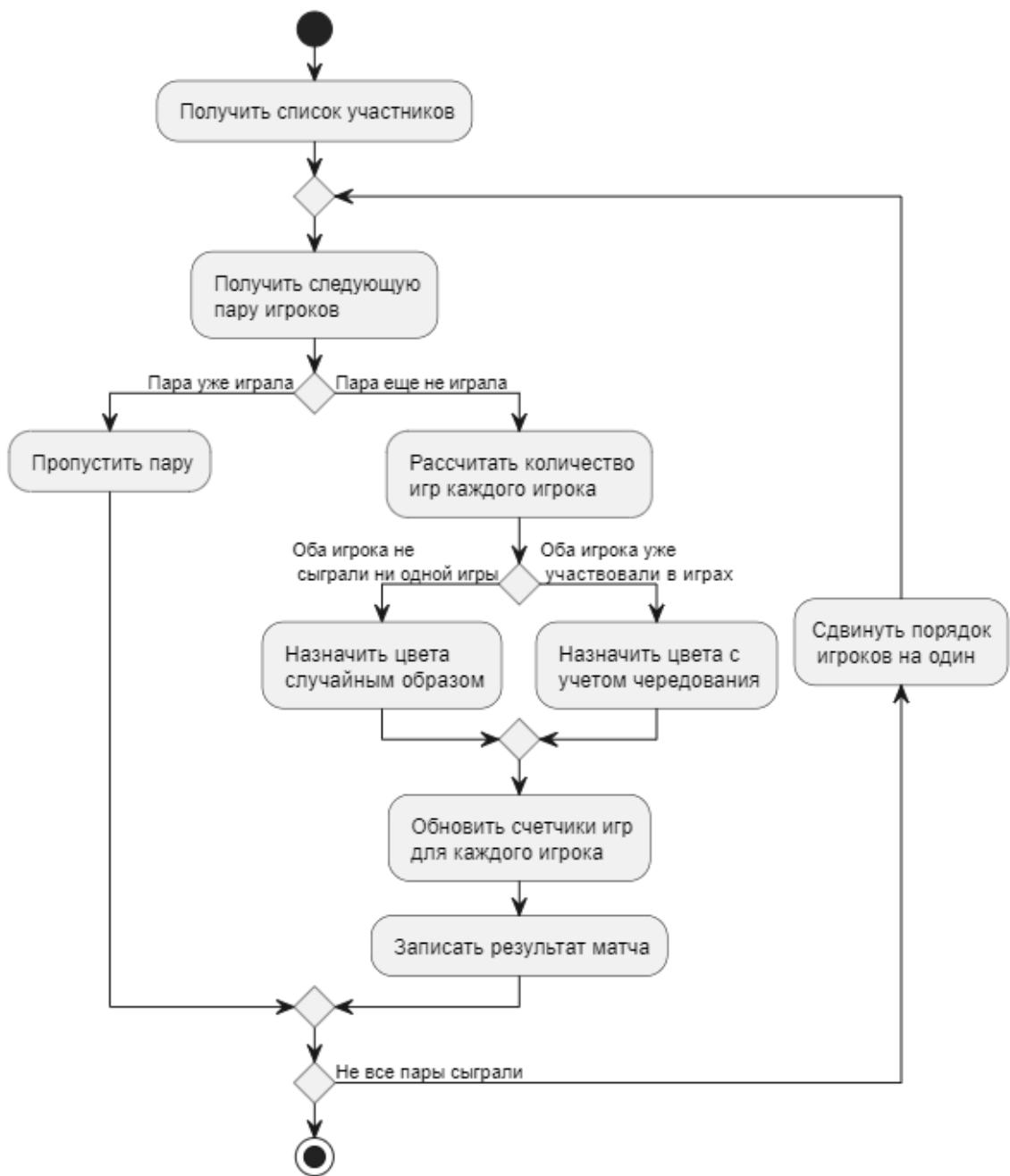


Рисунок 2.41 – Диаграмма активности алгоритма круговой жеребьевки

Глава 3 Реализация информационной системы

3.1 Реализация взаимодействия классов модуля интерфейса

Исходя из спроектированных диаграмм классов, классы, ответственные за представление пользовательского интерфейса (view), обеспечивают взаимодействие пользователя с бизнес-логикой с помощью классов модели представления (view model). Представления выводят ту информацию, которую получают из моделей представления, привязываясь к их свойствам посредством механизма привязки (binding). Обработка пользовательского ввода происходит в классах моделей представления.

После того, как пользователь ввел данные в форму представления, данные измененных свойств в модели представления также изменяются, так как модели представления наследуют интерфейс `INotifyPropertyChanged` от базового класса модели представления.

После завершения ввода пользователь нажимает на кнопку сохранения изменений, после чего по нажатию запускается команда, определенная дополнительно в классе команды.

По завершении действия генерируется событие, определенное в отдельном классе, позволяющее осуществить подписку на него из любого другого класса.

3.1.1 Взаимодействие классов входа в систему

Далее приведена диаграмма зависимостей типов, где сплошными синими линиями указаны зависимости использования (вызов методов), пунктирными зелеными – агрегационные зависимости (использование экземпляров класса), желтая линия композиции – передача класса в конструкторе.

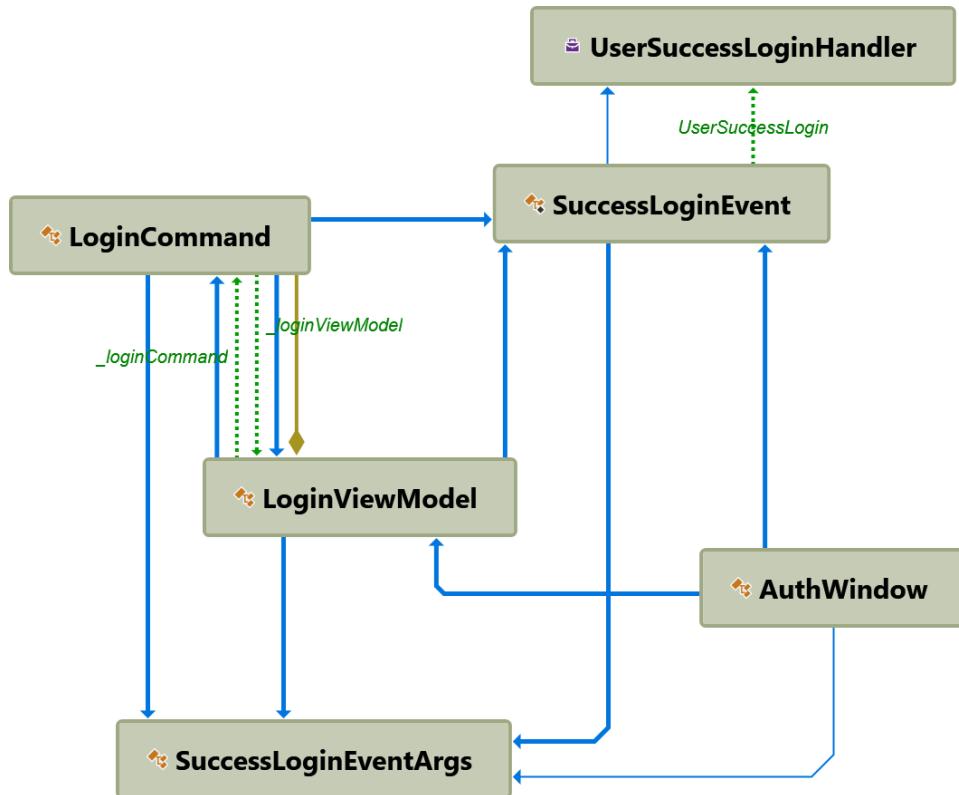


Рисунок 3.1 – Диаграмма зависимостей типов функции входа в систему

Исходя из данной диаграммы на примере реализации функционала для входа в систему видно, что представление (AuthWindow) зависит от модели представления (LoginViewModel), также оно зависит от аргументов события успешного входа (SuccessLoginEvent и SuccessLoginEventArgs), т.к. при возникновении события SuccessLoginEvent, данное окно должно закрыться, а откроется главное окно.

Модель представления также зависит от возникновения события успешного входа, т.к. при успешном входе, пользователь запоминается в данном классе, чтобы передать о нем данные для следующих классов, что, вероятно, не самая лучшая реализация, так как можно было бы вызвать конструктор следующего класса со встраиванием пользователя в качестве параметра, однако, это одна из простейших реализаций такого способа сохранения информации о пользователе пока запущено приложение.

В модели представления также создается экземпляр класса команды LoginCommand, содержащей метод запуска. Этот класс также зависит от LoginViewModel, так как встраиванием этого класса в конструктор получает необходимые поля для выполнения команды.

При успешном выполнении команды, она вызывает событие успешного выполнения действия.

3.1.2 Взаимодействие классов редактирования турнира

Далее приведена диаграмма зависимостей типов, где сплошными синими линиями указаны зависимости использования (вызов методов), пунктирными зелеными – агрегационные зависимости (использование экземпляров класса).

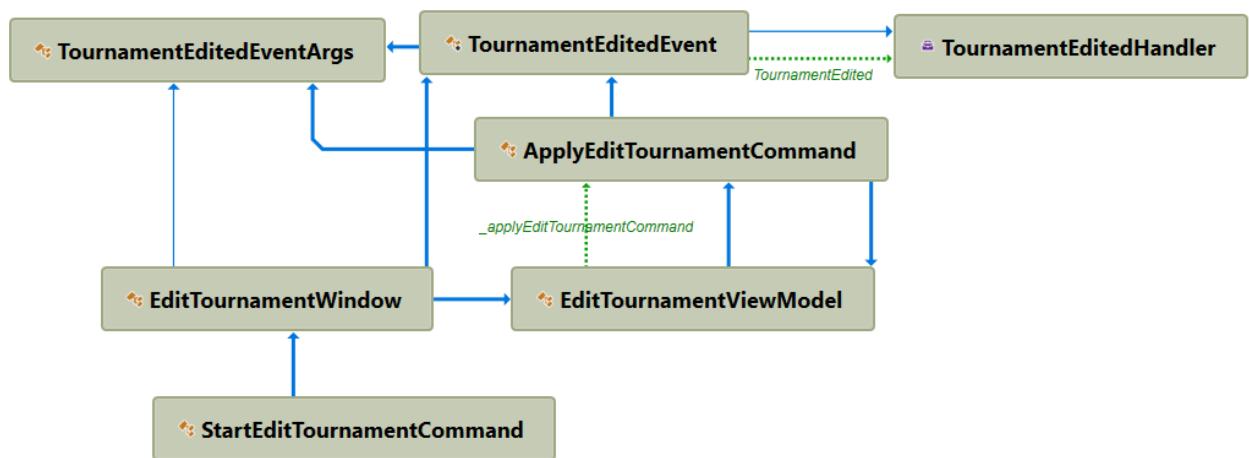


Рисунок 3.2 – Диаграмма зависимостей типов редактирования турнира

Исходя из данной диаграммы видно, что взаимодействие начинается с запуска команды начала редактирования турнира, при запуске которой происходит запуск окна редактирования турнира (EditTournamentWindow).

Окно редактирования привязано к модели представления окна редактирования с тем, чтобы обновлять поля ввода и передавать их команде применения изменений (ApplyEditTournamentCommand), которая отсылает данные турнира в базу и вызывает событие редактирования турнира (TournamentEditedEvent).

3.1.3 Взаимодействие классов создания турнира

Далее приведена диаграмма зависимостей типов, где сплошными синими линиями указаны зависимости использования (вызов методов), пунктирными зелеными – агрегационные зависимости (использование экземпляров класса).

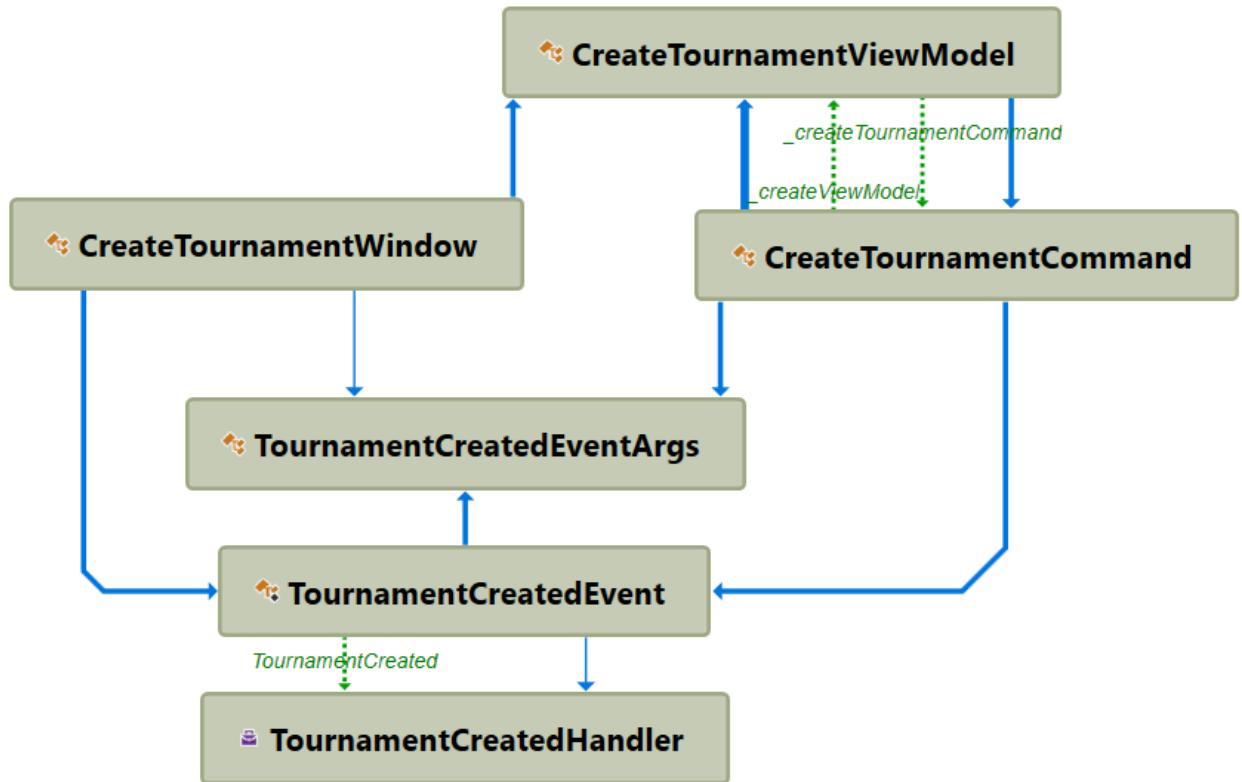


Рисунок 3.3 – Диаграмма зависимостей типов создания турнира

Исходя из данной диаграммы видно, что окно создания турнира (CreateTournamentWindow), как и все другие представления, зависит от модели представления (CreateTournamentViewModel), используя систему привязки данных.

При вызове команды создания турнира (CreateTournamentCommand), в базу отправляется запрос на добавление турнира. При удачном его исполнении вызывается событие создания турнира (TournamentCreatedEvent).

3.1.4 Взаимодействие классов управления играми

Далее приведена диаграмма зависимостей типов, где сплошными синими линиями указаны зависимости использования (вызов методов), пунктирными зелеными – агрегационные зависимости (использование экземпляров класса). Серыми стрелками обозначены связи с классами, лежащими на уровень ниже, которые будут описаны далее.

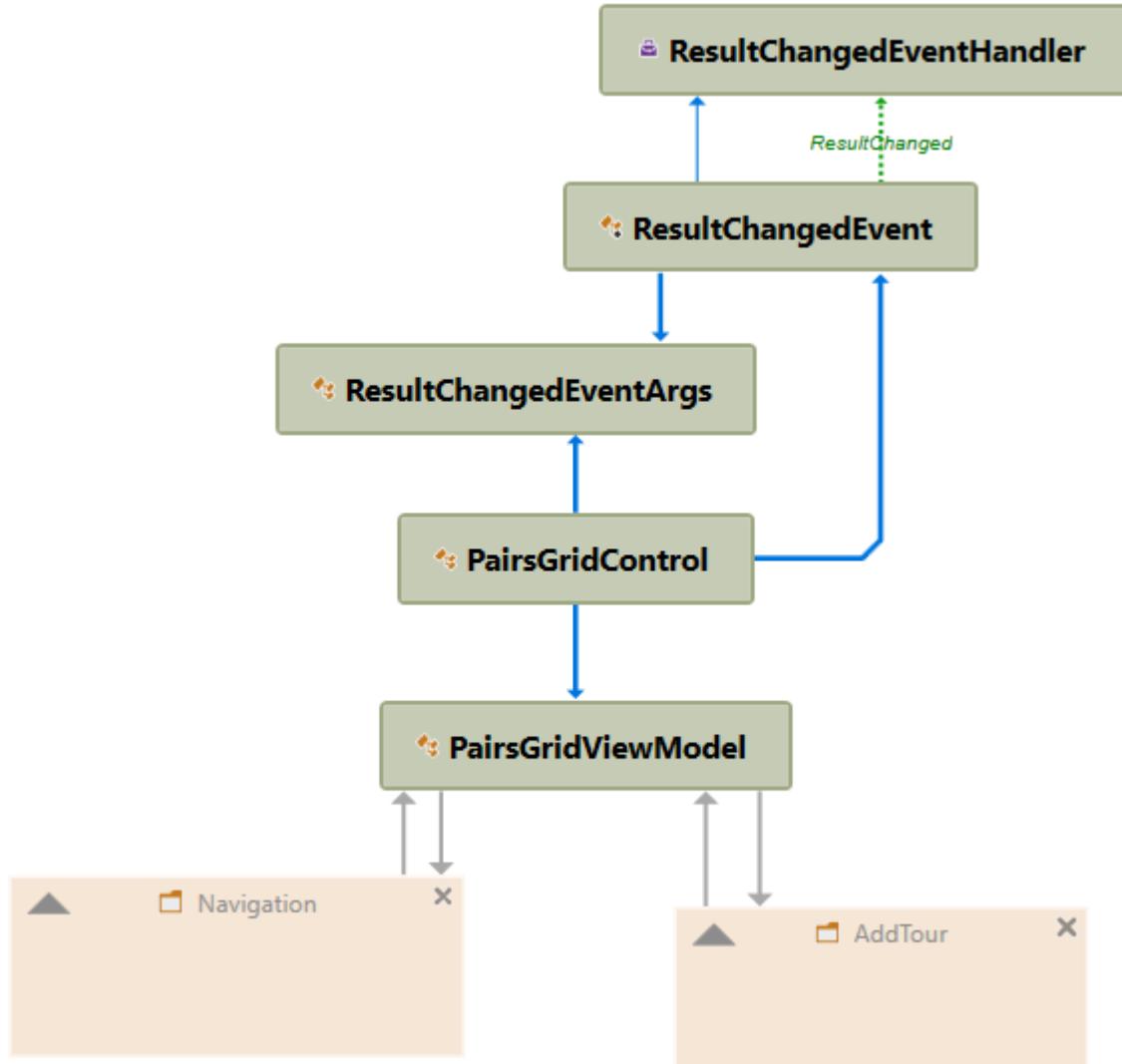


Рисунок 3.4 – Диаграмма зависимостей типов управления турнирами

Исходя из данной диаграммы видно, что представление, представленное в виде пользовательского элемента управления, выполненного в виде таблицы (PairsGridControl), зависит от события изменения результата, но не имеет команд, так как все изменения сохраняются автоматически (после каждого изменения данных в таблице, отсылается запрос на обновление данных).

Также сама модель представления имеет связи с классами навигации по турам и классами, связанными с добавлением нового тура.

3.1.4.1 Навигация по турам

Навигация по турам (см. рисунок 3.5) представляет собой два класса команд, вызываемых из модели представления (PairsGridViewModel).

Команда показа предыдущего тура (ShowPrevTourCommand) принимает данные модели представления и обновляет их в соответствии с данными предыдущего тура.

Аналогично работает и команда показа следующего тура (ShowNextTourCommand).

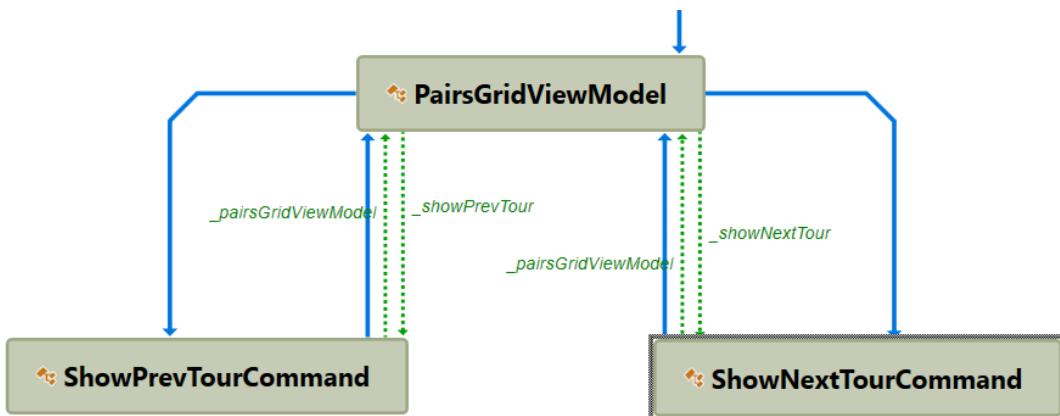


Рисунок 3.5 – Диаграмма зависимостей типов навигации по турам

3.1.4.2 Добавление нового тура

Добавление нового тура (см. рисунок 3.6) происходит при вызове команды StartNewTourCommand, которая вызывается из модели представления (PairsGridViewModel).

При вызове команды происходит также вызов алгоритма жеребьевки, порезультатам которого получается список пар нового тура.

При добавлении каждой пары и при добавлении всего тура вызываются соответствующие события.

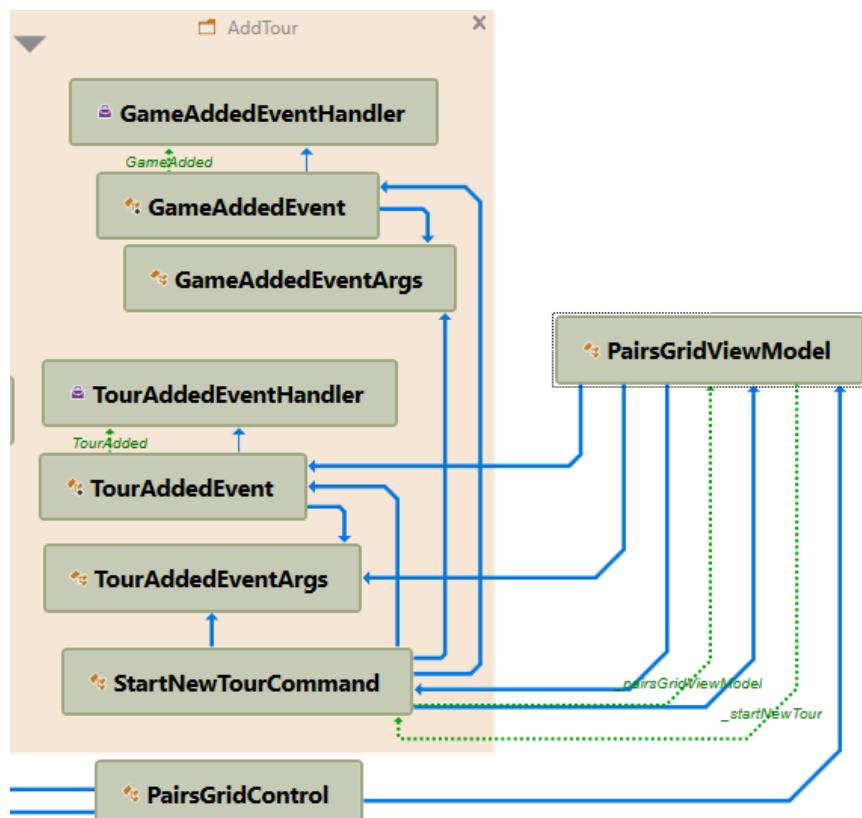


Рисунок 3.6 – Диаграмма зависимостей типов добавления нового тура

3.1.5 Взаимодействие классов управления командами и группами

Так как группы и команды представляют собой схожий функционал (в них состоят игроки), то далее будет описание взаимодействия классов как для команд, так и для групп.

Далее приведены диаграммы зависимостей типов, где сплошными синими линиями указаны зависимости использования (вызов методов), пунктирными зелеными – агрегационные зависимости (использование экземпляров класса). Серыми стрелками обозначены связи с классами, лежащими на уровень ниже, которые будут описаны далее.

Исходя из следующих диаграмм (см. рисунок 3.7 и рисунок 3.8), представления команд (TeamsListControl) и групп (GroupsListControl) зависят от моделей представления и классов, связанных с редактированием этих представлений (используется подписка на событие редактирования коллекций элементов).

Модели представления, в свою очередь, связаны со всеми классами (редактирования, добавления, удаления)

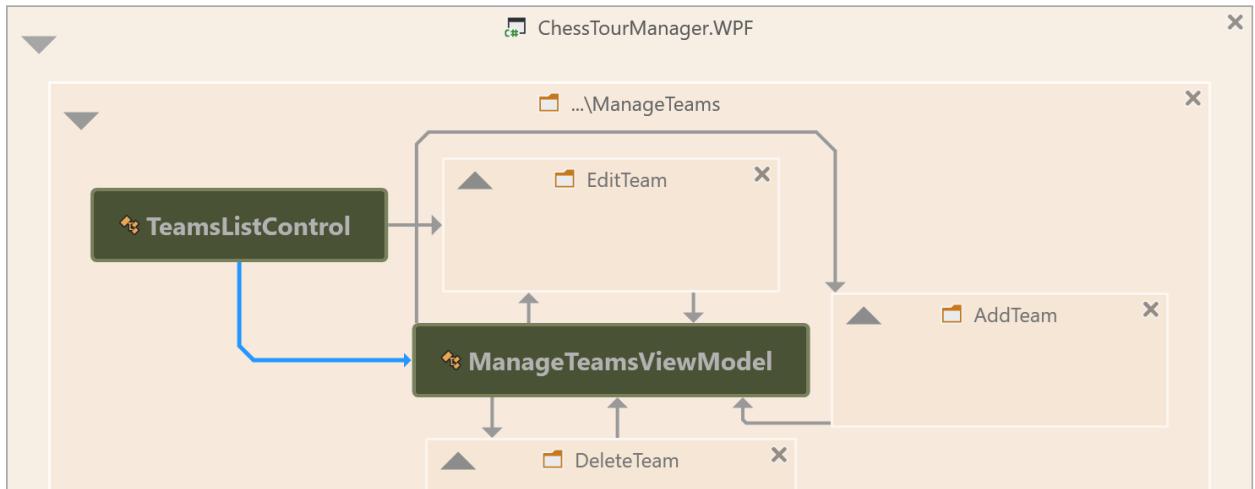


Рисунок 3.7 – Диаграмма взаимодействия классов управления командами

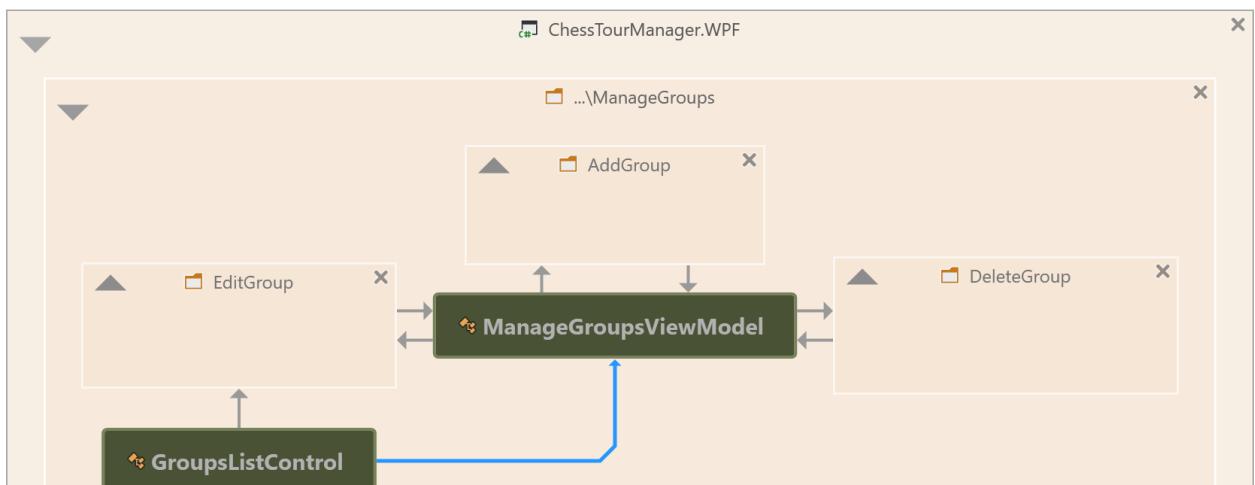


Рисунок 3.8 – Диаграмма взаимодействия классов управления группами

3.1.5.1 Добавление

Добавление команд и групп основано на вызове соответствующей команды из модели представления, которая запускает новое окно для ввода названия команды или группы. Для этого окна не создается отдельная модель представления, так как ввод предполагает только одно значение, поэтому моделью представления данных окон является основная модель представления (ManageTeamsViewModel или ManageGroupsViewModel).

После ввода данных при нажатии на кнопку сохранения происходит вызов команды завершения добавления (CompleteAddTeamCommand или CompleteAddGroupCommand).

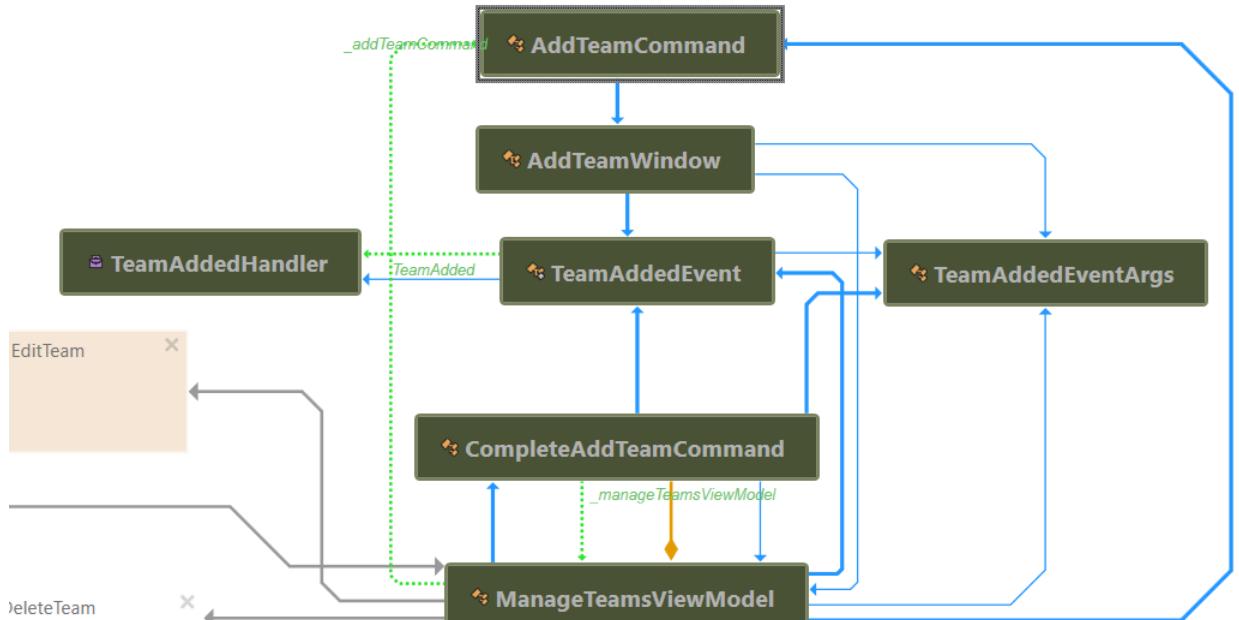


Рисунок 3.9 – Диаграмма взаимодействия классов добавления команды

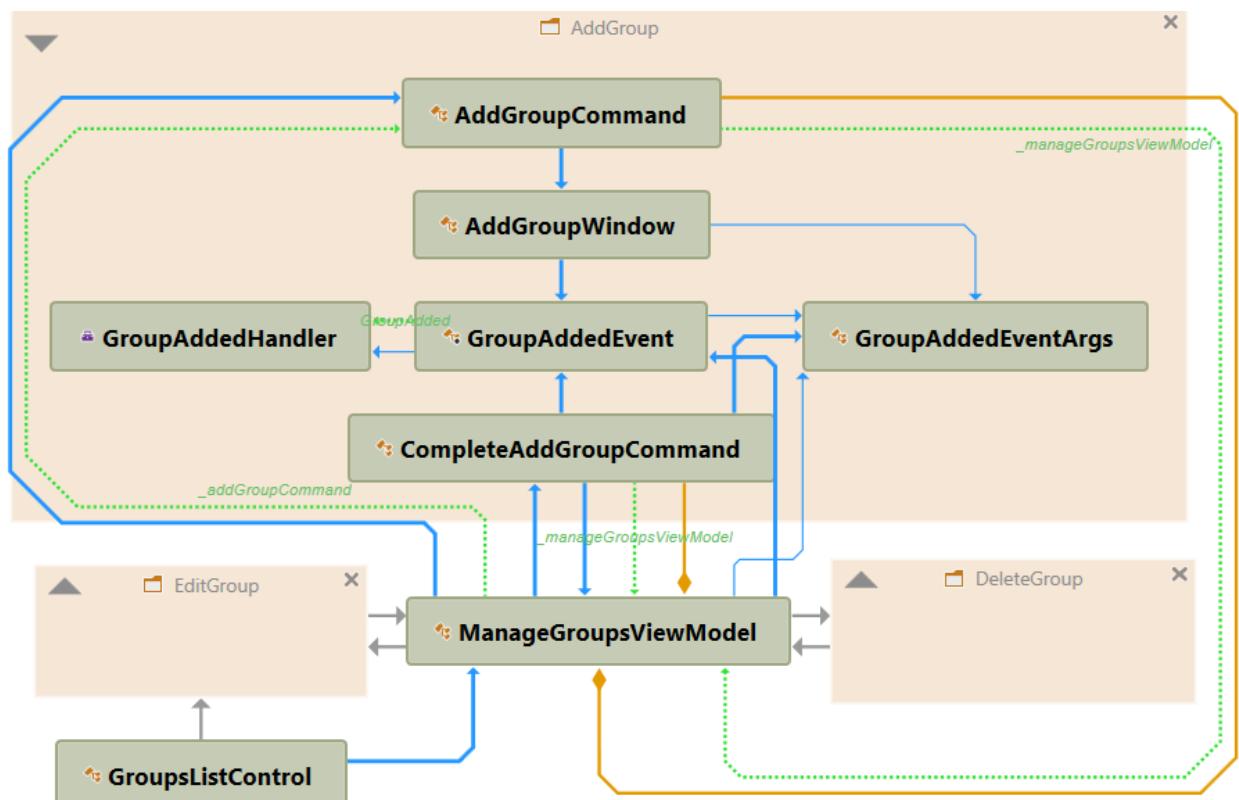


Рисунок 3.10 – Диаграмма взаимодействия классов добавления группы

3.1.5.2 Редактирование

Редактирование организовано схожим образом с добавлением, но в этом случае для окна редактирования предназначен дополнительный класс модели представления, так как в редактировании команды могут быть изменены параметры, задаваемые при создании по умолчанию.

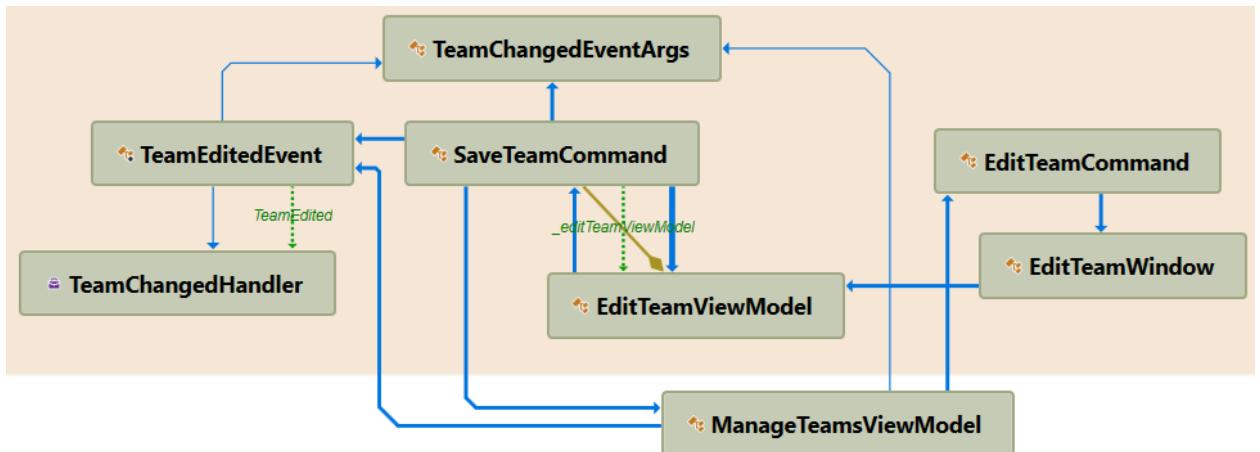


Рисунок 3.11 – Диаграмма взаимодействия классов редактирования команды

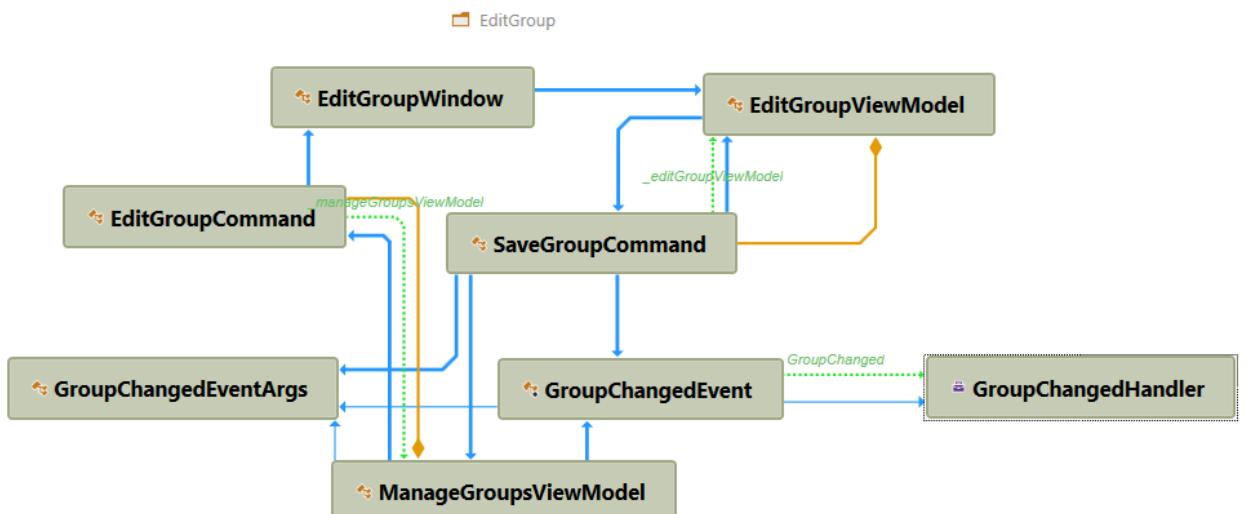


Рисунок 3.12 – Диаграмма взаимодействия классов редактирования группы

3.1.5.3 Удаление

Удаление команды или группы происходит при вызове соответствующей команды. Если удаление происходит успешно, то вызывается соответствующее событие.

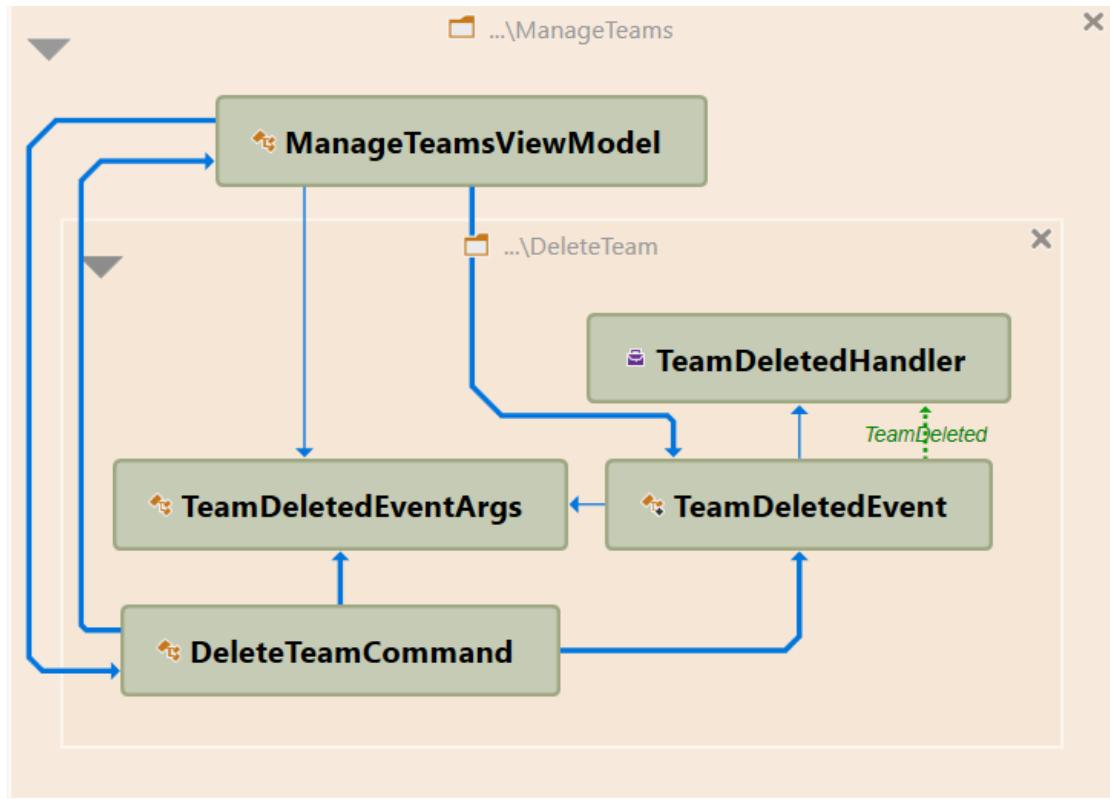


Рисунок 3.13 – Диаграмма взаимодействия классов удаления команды

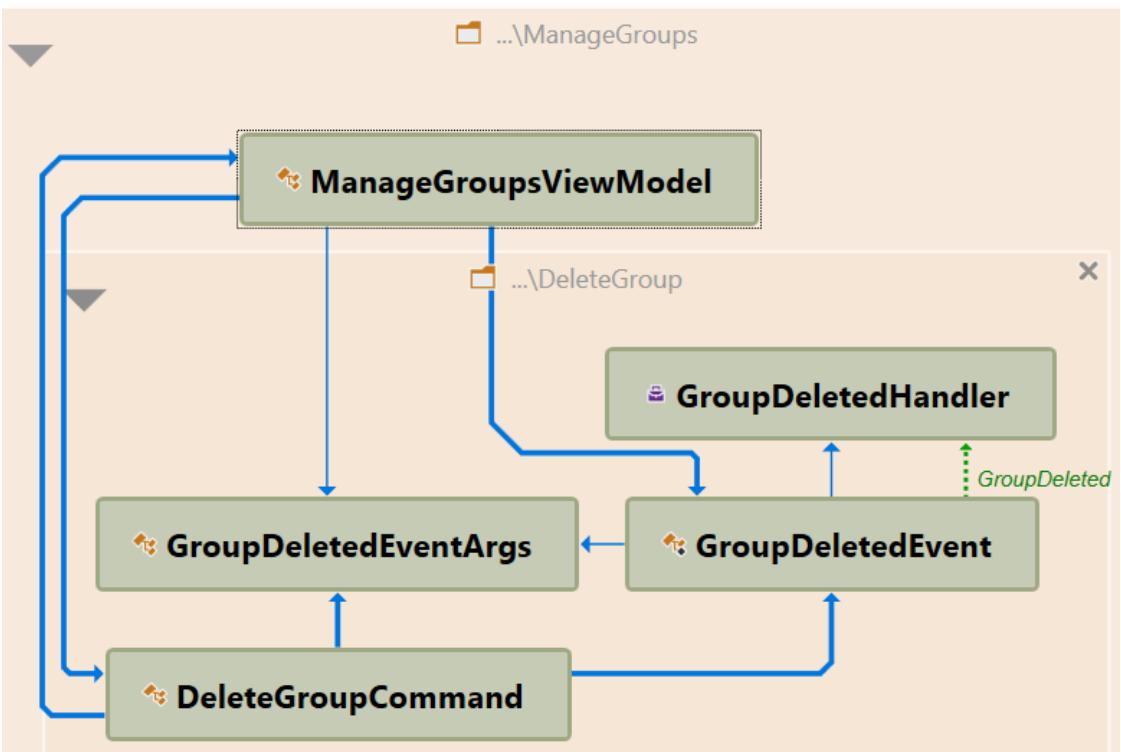


Рисунок 3.14 – Диаграмма взаимодействия классов удаления группы

3.2 Реализация алгоритмов модуля бизнес-логики

Далее приведено описание реализации алгоритма жеребьевки участников турнира.

3.2.1 Круговой алгоритм

Для реализации кругового алгоритма был создан специальный класс, в конструкторе которого вызывается метод запроса списка игр, который затем записывается в коллекцию истории игр, созданной для того, чтобы отслеживать сыгранные пары игроков (см. рисунок 3.15).

Структура истории игр представляет собой множество пар целочисленных значений, состоящих из ID игроков. Множество позволяет обеспечить уникальность пар и не допустить их повторных встреч.

Далее записываются ID всех игроков в виде списка для того, чтобы потом проводить жеребьевку, храня при этом не самих игроков, а их ID, что позволяет сэкономить на памяти и времени выполнения алгоритма.

Затем вызывается метод настройки списка пар всех туров.

```
public RoundRobin(ChessTourContext context, Tournament tournament)
{
    IGetQueries.CreateInstance(context) // IGetQueries
        .TryGetGames(organiserId: tournament.OrganizerId,
                     tournament.TournamentId,
                     out IEnumerable<Game>? games);
    if (games is not null)
    {
        GamesHistory = games.Select(g:Game => (g.WhiteId, g.BlackId)).ToHashSet();
    }

    IGetQueries.CreateInstance(context) // IGetQueries
        .TryGetPlayers(organiserId: tournament.OrganizerId, tournament.TournamentId,
                      out IEnumerable<Player>? players);

    PlayersIds = players?.Select(p:Player => p.PlayerId).ToList() ?? new List<int>();
    ConfigureTours();
}
```

Рисунок 3.15 – Инициализация класса кругового алгоритма

Метод настройки списка пар всех туров представляет собой цикл, проходящий по номерам всех туров турнира и вызывающий метод настройки списка пар тура. Далее этот список добавляется в коллекцию пар тура, представляющую собой словарь, ключами которого являются номера тура, а значениями – список пар для этого тура (см. рисунок 3.16).

Для обеспечения сменяемости пар происходит сдвиг игроков.

```
private void ConfigureTours()
{
    PlayersIds.Sort();
    for (var tourNumber = 1; tourNumber <= PlayersIds.Count / 2; tourNumber++)
    {
        _pairsForTour.TryAdd(tourNumber, ConfigurePairs());

        // Rotate players, so that the first player is always the same
        PlayersIds.Add(PlayersIds[0]);
        PlayersIds.RemoveAt(index: 0);
    }
}
```

Рисунок 3.16 – Жеребьевка пар всего турнира

Метод настройки списка пар тура представляет собой разбиение игроков на две половины, у последней из которых обращается порядок следования (см. рисунок 3.17).

Также на каждом четном номере тура происходит смена половин списка игроков, что позволяет обеспечить чередование цвета игроков (порядка начала партии).

Полученные последовательности соединяются в одну, образующую множество пар. Это множество и возвращается в качестве списка пар тура.

```
private HashSet<(int, int)> ConfigurePairs()
{
    IEnumerable<int> whiteIds;
    IEnumerable<int> blackIds;

    // Swap colors for even tours.
    if (NewTourNumber % 2 == 0)
    {
        whiteIds = _playersIds.Take(_playersIds.Count / 2);
        blackIds = _playersIds.Skip(_playersIds.Count / 2).Reverse();
    }
    else
    {
        whiteIds = _playersIds.Take(_playersIds.Count / 2).Reverse();
        blackIds = _playersIds.Skip(_playersIds.Count / 2);
    }

    return new HashSet<(int, int)>(collection: whiteIds.Zip(blackIds, (w:int, b:int) => (w, b)));
}
```

Рисунок 3.17 – Жеребьевка пар тура

Также реализуется дополнительный метод начала нового тура, который принимает в качестве параметра номер текущего тура, список пар относительно которого необходимо получить.

Для каждой пары игроков, определенной для этого тура, проверяется отсутствие пары в истории игр. Если пара присутствует в истории, то есть игра между этими игроками уже проводилась, то будут проверяться пары, определенные для следующего тура. Это полезно тогда, когда турнир уже начался, сыграно несколько туров.

Если пара не играла, то она возвращается с помощью итератора, обеспеченного интерфейсом `IEnumerable`.

```
public IEnumerable<(int, int)> StartNewTour(int currentTour)
{
    NewTourNumber = currentTour + 1;
    int tour = NewTourNumber;
    foreach ((int, int) pair in _pairsForTour[tour])
    {
        if (GamesHistory.Contains(pair))
        {
            tour++;
            continue;
        }

        yield return pair;
    }
}
```

Рисунок 3.18 – Получение пар тура

3.3 Реализация запросов LINQ модуля доступа к данным

Так как доступ к данным решено было организовать с помощью ORM Entity Framework, запросы были реализованы с помощью языка интегрированных запросов LINQ.

3.3.1 Получение данных

Для организации получения данных был создан специальный класс, методы которого описаны далее.

3.3.1.1 Получение пользователя

Далее предоставлен метод получения пользователя по его ID. Среди пользователей выбирается тот, который имеет ID, соответствующий переданному в функцию параметру.

Также помимо самого пользователя для него загружается список его турниров.

Если пользователь найден, то возвращается успешный результат, иначе – результат о том, пользователь не найден.

```
public GetResult TryGetUserById(int id, out User? user)
{
    IEnumerable<User> usersLocal = _context.Users.Include(navigationPropertyName: u:User => u.Tournaments);
    user = usersLocal.FirstOrDefault(u:User => u.UserId == id);

    return user is not null
        ? GetResult.Success
        : GetResult.UserNotFound;
}
```

Рисунок 3.19 – Получение пользователя по ID

Также создан метод получения пользователя по его логину и паролю. При этом используется специальный класс хеширования пароля.

Пароль, переданный в качестве параметра, хешируется и затем передается в качестве параметра поиска пользователя по его логину и хешу пароля.

```
public GetResult TryGetUserByLoginAndPass(string login, string password, out User? user)
{
    string hash = PasswordHasher.HashPassword(password);
    user = _context.Users.FirstOrDefault(u:User => u.Email == login && u.PassHash == hash);
    return user is not null
        ? GetResult.Success
        : GetResult.UserNotFound;
}
```

Рисунок 3.20 – Получение пользователя по логину и паролю

3.3.1.2 Получение турниров

Метод получения турниров пользователя основан на проверке существования пользователя, и если этот пользователь существует, то возвращаются его турниры, иначе – пустой список.

```

public GetResult TryGetTournaments(int organiserId, out IEnumerable<Tournament>? tournaments)
{
    if (TryGetUserById(organiserId, out User? organiser) = GetResult.Success)
    {
        tournaments = organiser!.Tournaments;
        return GetResult.Success;
    }

    tournaments = default(IEnumerable<Tournament>);
    return GetResult.UserNotFound;
}

```

Рисунок 3.21 – Получение списка турниров пользователя

Также создан метод, позволяющий получить турниры пользователя со вложенными коллекциями команд и игроков этих команд.

Сначала проверяется существование пользователя с переданным ID, затем отбираются те турниры, чьим организатором является этот пользователь. Далее к этим турнирам подключается коллекция команд, а к каждой команде – коллекция игроков.

```

public GetResult TryGetTournamentsWithTeamsAndPlayers(int organiserId, out IEnumerable<Tournament>? tournaments)
{
    if (TryGetUserById(organiserId, out User? _) = GetResult.Success)
    {
        tournaments = _context.Tournaments // DbSet<Tournament>
            .Where(t:Tournament => t.OrganizerId = organiserId) // IQueryable<Tournament>
            .Include(navigationPropertyName: t:Tournament => t.Teams) // IIncludableQueryable<Tournament, ICollection<_>>
            .ThenInclude(t:Team => t.Players); // IIncludableQueryable<Tournament, ICollection<_>>

        return GetResult.Success;
    }

    tournaments = default(IEnumerable<Tournament>);
    return GetResult.UserNotFound;
}

```

Рисунок 3.22 – Получение турнира с вложенными коллекциями команд и игроков

3.3.1.3 Получение игроков

Для получения списка игроков конкретного турнира сначала проверяется наличие организатора и турнира, игроков которого необходимо получить. Затем возвращаются игроки данного турнира.

```
public GetResult TryGetPlayers(int organiserId, int tournamentId, out IEnumerable<Player>? players)
{
    players = default(IEnumerable<Player>);
    if (TryGetUserById(organiserId, out User? user) == GetResult.UserNotFound)
    {
        return GetResult.UserNotFound;
    }

    if (user!.Tournaments.Count == 0)
    {
        return GetResult.NoTournaments;
    }

    IEnumerable<Tournament> tournaments = _context.Tournaments // DbSet<Tournament>
                                                .Where(t:Tournament => t.OrganizerId == organiserId) // :
                                                .Include(navigationPropertyName: t:Tournament => t.Players);

    Tournament? tournament = tournaments.FirstOrDefault(t:Tournament => t.TournamentId == tournamentId);
    if (tournament is null)
    {
        return GetResult.TournamentNotFound;
    }

    players = tournament.Players;

    return GetResult.Success;
}
```

Рисунок 3.23 – Получение игроков конкретного турнира

Также был создан метод, позволяющий получить список игроков вместе с командами и группами, в которых они состоят.

Сначала происходит фильтрация турниров среди тех, которые не принадлежат данному пользователю, затем происходит включение коллекций игроков в турнир, а к игрокам подключаются группы и команды, в которых они состоят.

```
public GetResult TryGetPlayersWithTeamsAndGroups(int organiserId, int tournamentId,  
                                                out IEnumerable<Player>? players)  
{  
    players = default(IEnumerable<Player>);  
    if (TryGetUserById(organiserId, out User? user) == GetResult.UserNotFound)  
    {  
        return GetResult.UserNotFound;  
    }  
  
    if (user!.Tournaments.Count == 0)  
    {  
        return GetResult.NoTournaments;  
    }  
  
    IQueryble<Tournament> tournaments = _context.Tournaments // DbSet<Tournament>  
                                         .Where(t:Tournament => t.OrganizerId == organiserId) // IQu  
                                         .Include(navigationPropertyName: t:Tournament => t.Players) //  
                                         .ThenInclude(p:Player => p.Group);  
    tournaments = tournaments.Include(navigationPropertyName: t:Tournament => t.Players) // IIIncludableQueryble<Tour  
                                         .ThenInclude(p:Player => p.Team); // IIIncludableQueryble<Tournament, Team?>  
  
    Tournament? tournament = tournaments.FirstOrDefault(t:Tournament => t.TournamentId == tournamentId);  
    if (tournament is null)  
    {  
        return GetResult.TournamentNotFound;  
    }  
  
    players = tournament.Players;  
  
    return GetResult.Success;  
}
```

Рисунок 3.24 – Получение игроков конкретного турнира с командами и группами

3.3.1.4 Получение команд

Далее приведена реализация метода получения команд с игроками, состоящими в ней.

По аналогии с предыдущими методами сначала происходит выбор турниров у пользователя с подключением команд и пользователей, затем возвращается турнир, с заданным ID.

```
public GetResult TryGetTeamsWithPlayers(int organiserId, int tournamentId, out IEnumerable<Team>? teams)
{
    teams = default(IEnumerable<Team>);
    if (TryGetUserById(organiserId, out User? user) == GetResult.UserNotFound)
    {
        return GetResult.UserNotFound;
    }

    if (user is { Tournaments.Count: 0 })
    {
        return GetResult.NoTournaments;
    }

    IEnumerable<Tournament> tournaments = _context.Tournaments // DbSet<Tournament>
        .Where(t:Tournament => t.OrganizerId == organiserId) // И
        .Include(navigationPropertyName: t:Tournament => t.Teams) // И
        .ThenInclude(t:Team => t.Players);

    Tournament? tournament = tournaments.FirstOrDefault(t:Tournament => t.TournamentId == tournamentId);
    if (tournament is null)
    {
        return GetResult.TournamentNotFound;
    }

    teams = tournament.Teams;

    return GetResult.Success;
}
```

Рисунок 3.25 – Получение команд с вложенными коллекциями игроков

3.3.1.5 Получение групп

По аналогии с получением команд организовано получение групп. При этом к каждой группе подключаются коллекции игроков, состоящих в ней.

```
public GetResult TryGetGroups(int organizerId, int tournamentId,
                               out IEnumerable<Group>? groups)
{
    groups = default(IEnumerable<Group>);
    if (TryGetUserId(organizerId, out User? user) == GetResult.UserNotFound)
    {
        return GetResult.UserNotFound;
    }

    if (user is { Tournaments.Count: 0 })
    {
        return GetResult.NoTournaments;
    }

    IEnumerable<Tournament> tournaments = _context.Tournaments.Where(g:Tournament => g.OrganizerId == organizerId);

    Tournament? tournament = tournaments.FirstOrDefault(t:Tournament => t.TournamentId == tournamentId);
    if (tournament is null)
    {
        return GetResult.TournamentNotFound;
    }

    groups = _context.Groups // DbSet<Group?>
        .Where(g:Group? => g.OrganizerId == organizerId && g.TournamentId == tournamentId) // IQueryable<Group?>
        .Include(navigationPropertyName: g:Group? => g.Players); // IIIncludableQueryable<Group?, ICollection<..>>
    return GetResult.Success;
}
```

Рисунок 3.26 – Получение групп с вложенными списками игроков

3.3.2 Добавление данных

Рассмотрим добавление данных в базу на примере добавления игрока.

В качестве параметров передаются параметры игрока, некоторые из них могут быть заданы по умолчанию. Далее создается экземпляр класса сущности игрока, который добавляется в коллекцию игроков, возвращается в качестве выходного параметра.

Если при добавлении произошла ошибка, то запускаются окна с сообщениями об ошибке, программа при этом не завершает работу, пользователь может исправить ошибки.

```
public InsertResult TryAddPlayer(out Player? addedPlayer, int tournamentId, int organiserId, string lastName,
                                string firstName,
                                char gender = 'M',
                                string attribute = "-",
                                int birthYear = 2000,
                                int boardNumber = 1,
                                int? teamId = null,
                                int? groupId = null,
                                bool isActive = true)

{
    try
    {
        addedPlayer = new Player
        {
            TournamentId = tournamentId,
            OrganizerId = organiserId,
            PlayerLastName = lastName,
            PlayerFirstName = firstName,
            Gender = gender,
            PlayerAttribute = attribute,
            PlayerBirthYear = birthYear,
            BoardNumber = boardNumber,
            TeamId = teamId,
            GroupId = groupId,
            IsActive = isActive
        };

        _context.Players.Add(addedPlayer);
        _context.SaveChanges();
        return InsertResult.Success;
    }
    catch (DbUpdateException)
    {
        MessageBox.Show(messageBoxText: "Ошибка в веденных данных! Возможно игрок с такими данными уже существует,"
                                         + " либо вы не заполнили важные данные", caption: "Ошибка добавлении игрока", MessageBoxButtons.OK,
                                         icon: MessageBoxIcon.Error);
        addedPlayer = null;
        return InsertResult.Fail;
    }
    catch (Exception e)
    {
        MessageBox.Show(e.InnerException?.Message ?? e.Message, caption: "Ошибка добавлении игрока", MessageBoxButtons.OK,
                                         icon: MessageBoxIcon.Error);
        addedPlayer = null;
        return InsertResult.Fail;
    }
}
```

Рисунок 3.27 – Добавление игрока

3.3.3 Удаление данных

Удаление данных также рассмотрим на при мере сущности игрока.

Метод принимает на вход параметр типа игрока, который затем удаляется из коллекции игроков. Если удаление прошло с ошибкой, то открывается окно с сообщением ошибки. Так, например, при попытке удалить игрока, который участвует в играх турнира, возникнет окно с сообщением о том, что данного игрока удалить нельзя.

```
public DeleteResult TryDeletePlayer(Player player)
{
    try
    {
        _context.Players.Remove(player);
        _context.SaveChanges();
        return DeleteResult.Success;
    }
    catch (DbUpdateException)
    {
        MessageBox.Show(messageBoxText: "Нельзя удалить игрока, который участвует в игре!",
                      caption: "Ошибка при удалении игрока",
                      MessageBoxButtons.OK, icon: MessageBoxIcon.Error);
        _context.Entry(player).State = EntityState.Unchanged;
        return DeleteResult.Failed;
    }
    catch (Exception e)
    {
        MessageBox.Show(e.InnerException?.Message ?? e.Message, caption: "Ошибка при удалении игрока",
                      MessageBoxButtons.OK, icon: MessageBoxIcon.Error);
        _context.Entry(player).State = EntityState.Unchanged;
        return DeleteResult.Failed;
    }
}
```

Рисунок 3.28 – Удаление игрока

3.3.4 Обновление данных

При обновлении свойств сущности вызывается метод применения изменений, позволяющий сохранить изменения для сущности, не пересоздавая ее.

```
EditTournamentViewModel.EditingTournament.DateLastChange = DateOnly.FromDateTime(DateTime.UtcNow);
EditTournamentViewModel.EditingTournament.TimeLastChange = TimeOnly.FromDateTime(DateTime.UtcNow);
TournamentsListViewModel.TournamentsListContext.SaveChanges();
```

Рисунок 3.29 – Пример изменения турнира

Глава 4 Тестирование информационной системы

4.1 Компонентное тестирование

Компонентное тестирование, также известное как unit-тестирование, основано на покрытии тестами тестируемого кода.

Для осуществления компонентного тестирования был создан отдельный проект, использующий фреймворк тестирования NUnit.

4.1.1 Тестирование методов запросов

Далее приведен план тестирования методов запросов, написанных с помощью LINQ.

4.1.1.1 Тестирование запросов получения данных

Для проверки качества запросов на получение данных необходимо проверить каждый метод, содержащий LINQ запрос на следующие случаи:

1. Объект поиска существует.
 - 1.1. Корректный ID.
 - 1.1.1. Объект не имеет вложенные коллекции.
 - 1.1.2. Объект имеет вложенные коллекций.
 - 1.2. Корректный логин и пароль (для пользователя).
2. Объект поиска не существует.
 - 2.1. Некорректный ID.
 - 2.2. Некорректный логин.
 - 2.3. Некорректный пароль.

Далее приведена демонстрация тестирования получения данных пользователя.

В таблице пользователей имеются следующие данные (см. рисунок 4.1).

	user_id	user_lastname	user_firstname	user_patronymic	email	pass_hash
1	4	Кузнецов	Кузьма	Кузьмич	kuzya@gmail.com	klsdf324klsdf32klsdf
2	5	Петрова	Анна	Петровна	annpe@g.com	klsdf324klsdf32klsdf
3	6	Владимиров	Владимир	-	vldmr@d.c	adsvc23mdfs23opdsop
4	3	Сидоров	Сидор	Сидорович	sedorr@ya.ru	F0CB589F5E948213306
5	2	Петров	Петр	Петрович	petre@live.com	F0CB589F5E948213306

Рисунок 4.1 – Таблица пользователей

Протестируем случай 2.1. Для этого попробуем найти пользователя с ID равным 1 (см. рисунок 4.2).

Как видно из результата теста, метод успешно вернул результат о том, что пользователь не существует с таким ID.

```
// 2. The object to be searched for does not exist.  
[Test]  
diamond  
public void TryGetUserById_WhenUserDoesNotExist_ReturnsUserNotFound()  
{  
    // Arrange.  
    var context = new ChessTourContext();  
    var queries = IGetQueries.CreateInstance(context);  
    const int id = 1;  
  
    // Act.  
    GetResult result = queries.TryGetUserById(id, out User? user);  
  
    // Assert.  
    Assert.AreEqual(expected: GetResult.UserNotFound, actual: result);  
    Assert.IsNull(user);  
}
```

The screenshot shows the Visual Studio Test Explorer window. The title bar says "RoundRobinTests". Below it, there's a toolbar with icons for filtering, sorting, and other test-related operations. The main area lists three test results:

- "ChessTourManager.UnitTests (3 tests) Failed: 2 tests failed" (red)
- "ChessTourManager.UnitTests (3 tests) Failed: 2 tests failed" (red)
- "GetQueriesTests (2 tests) Failed: 1 test failed" (red)
- "TryGetUserById_WhenUserDoesNotExist_ReturnsUserNotFound Success" (green checkmark)

The last item is highlighted with a blue selection bar.

Рисунок 4.2 – Тестирование получения несуществующего пользователя по ID

Теперь протестируем случай 2.2 (см. рисунок 4.3).

Как видно из результата теста, пользователя с таким логином не найдено, результат – UserNotFound.

```

// 2. The object to be searched for does not exist (incorrect login).
[Test]
◇
public void Try GetUserByLoginAndPass_WhenLoginIsNotCorrect_ReturnsUserNotFound()
{
    // Arrange.
    var context = new ChessTourContext();
    var queries = IGetQueries.CreateInstance(context);
    const string login = "incorrect";

    // Users with this password exist in the database.
    const string password = "123qwe";

    // Act.
    GetResult result = queries.Try GetUserByLoginAndPass(login, password, out User? user);

    // Assert.
    Assert.AreEqual(expected: GetResult.UserNotFound, actual: result);
    Assert.IsNull(user);
}

```



Рисунок 4.3 – Тестирование получения пользователя по неверному логину

Теперь протестируем случай 2.3 (см. рисунок 4.3).

Как видно из результата теста, пользователя с таким паролем не найдено, результат – `UserNotFound`.

```

// 2. The object to be searched for does not exist (incorrect password).
[Test]
◇
public void Try GetUserByLoginAndPass_WhenPasswordIsNotCorrect_ReturnsUserNotFound()
{
    // Arrange.
    var context = new ChessTourContext();
    var queries = IGetQueries.CreateInstance(context);
    const string password = "incorrect";

    // User with this login exists in the database.
    const string login = "petre@live.com";

    // Act.
    GetResult result = queries.Try GetUserByLoginAndPass(login, password, out User? user);

    // Assert.
    Assert.AreEqual(expected: GetResult.UserNotFound, actual: result);
    Assert.IsNull(user);
}

```

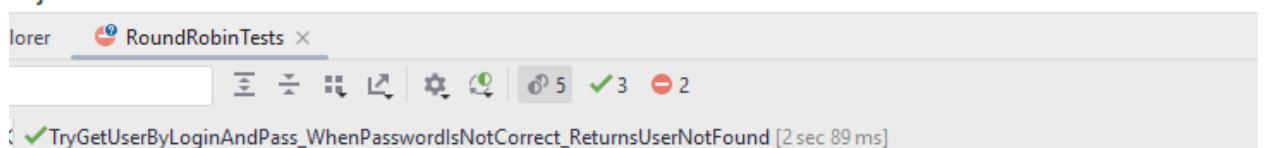


Рисунок 4.4 – Тестирование получения пользователя по неверному паролю

Теперь протестируем корректные случаи.

Протестируем случай 1.1.1 – пользователь существует, получаем без вложенных конструкций (см. рисунок 4.5).

Как видно из результата теста, пользователь с ID 2 существует, также корректен и полученный его email.

```
// 1.1.1. The object has no nested collections.
[Test]
◇
public void TryGetUserById_WhenUserExists_ReturnsSuccess()
{
    // Arrange.
    var context = new ChessTourContext();
    var queries = IGetQueries.CreateInstance(context);
    const int id = 2;

    // Act.
    GetResult result = queries.TryGetUserById(id, out User? user);

    // Assert.
    Assert.AreEqual(expected: GetResult.Success, actual: result);
    Assert.IsNotNull(user);
    Assert.AreEqual(expected: id, actual: user!.UserId);
    Assert.AreEqual(expected: "petre@live.com", actual: user.Email);
}

```



The screenshot shows a test runner window titled 'RoundRobinTests'. It displays a single test case: 'TryGetUserById_WhenUserExists_ReturnsSuccess' with a status of '1 sec 738 ms'. The status bar at the bottom indicates 5 total tests, 1 passed, 1 failed, and 3 pending.

Рисунок 4.5 – Тестирование получения пользователя по его ID (корректный случай)

Далее предоставлена таблицы турниров и команд (см. рисунки 4.6, 4.7).

	tournament_id	organizer_id	tournament_name	place
1	17	4	Турнир по шахматам	5 Москва
2	19	4	Чемпионат Пермского края	9 г. Пермь
3	21	3	Командный Чемпионат Пермского края по класси...	9 Россия, П
4	22	3	Командный Чемпионат Московской области по бы...	11 Россия, М
5	2	2	Рождественские встречи	7 -
6	3	2	Новогодний командный	7 -
7	18	2	Сельские игры	7 Пермь
8	20	2	Чемпионат России по быстрым шахматам	9 Россия, М
9	52	2	Название турнира	5 Место про

Рисунок 4.6 – Таблица турниров

	team_id	organizer_id	tournament_id	team_name
1	1	2	2	Медведь
2	2	2	2	ОСШ 1
3	3	2	2	ОСШ №2
4	4	2	2	ОСОШ №3
5	5	2	2	Белая ладья
6	6	2	2	Оханск
7	7	2	2	Чёрный ферзь
8	39	3	21	Team

Рисунок 4.7 – Таблица команд

Протестируем случай 1.1.2 – турнир существует, получаем с вложенной коллекцией (команды) (см. рисунок 4.8).

Как видно из таблиц, у пользователя с ID равным 2 – 5 турниров, при этом у турнира с ID равным 2 – 7 команд. Проверим, что метод корректно предоставляет эту информацию (см. рисунок 4.8).

Тест показал, что метод получения турнира также корректно возвращает и коллекцию команд, принадлежащую турниру: корректен результат, возвращаемый методом, корректно возвращается число турниров, количество команд для конкретного турнира, название турнира и название команды.

```

public void TryGetTournamentById_WhenTournamentExists_ReturnsSuccess()
{
    // Arrange.
    var context = new ChessTourContext();
    var queries = IGetQueries.CreateInstance(context);
    const int userId = 2;
    const int expectedTournamentsCount = 5;
    const int expectedTeamsCount = 7;
    const int expectedTournamentId = 2;
    const string expectedTournamentName = "Рождественские встречи";
    const int expectedTeamId = 1;
    const string expectedTeamName = "Медведь";

    // Act.
    GetResult result = queries.TryGetTournamentsWithTeamsAndPlayers(userId,
        out IEnumerable<Tournament> tournaments);
    Tournament tournament = tournaments!.Single(t => t.TournamentId == expectedTournamentId);
    Team team = tournament.Teams.Single(t => t.TeamId == expectedTeamId);

    // Assert.
    Assert.AreEqual(expected: GetResult.Success, actual: result);
    Assert.IsNotNull(tournaments);
    Assert.AreEqual(expectedTournamentsCount, actual: tournaments!.Count());
    Assert.AreEqual(expectedTeamsCount, actual: tournament.Teams.Count());
    Assert.AreEqual(expectedTournamentName, actual: tournament.TournamentName);
    Assert.AreEqual(expectedTeamName, actual: team.TeamName);
}

```



Рисунок 4.8 – Тестирование получения турнира с вложенной коллекцией

4.1.1.2 Тестирование запросов добавления данных

Для проверки качества запросов на получение данных необходимо проверить каждый метод, содержащий LINQ запрос на следующие случаи:

1. Некорректное добавление.
 - 1.1. Некорректный формат данных (null значения там, где их нельзя допускать).
 - 1.2. Объект с этими данными уже существует (пользователь с логином).
2. Корректное добавление.

Далее приведена демонстрация тестирования добавления данных пользователя (см. рисунок 4.9).

```

// 1.1. Incorrect data (nulls in required fields).
[Test]
◊
public void TryAddUser_WhenUserIsNull_ReturnsUserIsNull()
{
    // Arrange.
    var context = new ChessTourContext();
    var queries = IInsertQueries.CreateInstance(context);

    // Act.
    InsertResult result = queries.TryAddUser(email: null!,
                                              password: null!,
                                              lastName: null!,
                                              firstName: null!);

    // Assert.
    Assert.AreEqual(expected: InsertResult.Fail, actual: result);
}

```

RoundRobinTests x
7 ✓ 6 ✘ 1
GetQueriesTests (5 tests) [3 sec 108 ms]

Рисунок 4.9 – Тестирование некорректного добавления пользователя

4.1.2 Тестирование алгоритма жеребьевки

Для проверки качества алгоритма круговой жеребьевки необходимо проверить следующие случаи:

1. Жеребьевка первого тура.
 - 1.1. Активных игроков нет.
 - 1.1.1. Список игроков пуст.
 - 1.1.2. Только неактивные игроки (один).
 - 1.1.3. Только неактивные игроки (четное количество).
 - 1.1.4. Только неактивные игроки (нечетное количество).
- 1.2. Активный игрок один.
 - 1.2.1. Один активный.
 - 1.2.2. Один активный и один неактивный.
 - 1.2.3. Один активный и четное число неактивных.
 - 1.2.4. Один активный и нечетное число неактивных.
- 1.3. Активных игроков четное количество.
 - 1.3.1. Отсутствуют неактивные.
 - 1.3.2. Присутствует один неактивный.
 - 1.3.3. Присутствует четное число неактивных.
 - 1.3.4. Присутствует нечетное число неактивных.

1.4. Активных игроков нечетное количество.

1.4.1. Отсутствуют неактивные.

1.4.2. Присутствует один неактивный.

1.4.3. Присутствует четное число неактивных.

1.4.4. Присутствует нечетное число неактивных.

2. Жеребьевка не первого тура.

2.1. Отсутствуют активные игроки.

2.2. Активный игрок один.

2.3. Активных игроков четное количество.

2.4. Активных игроков нечетное количество.

Для каждого случая необходимо написать свой unit-test.

Для тестирования жеребьевки создадим пользователя и добавим турнир с игроками.

В методе предварительной настройки (SetUp) проинициализируем необходимые поля (см. рисунок 4.10).

```
private const int OrgId = 16;
private const int TourId = 53;
private readonly ChessTourContext _context = new();
private Tournament _tournament;
private IRoundRobin _roundAlgorithm;

[SetUp]
◇ 1 test OK aleks *
public void SetUp()
{
    IGetQueries.CreateInstance(_context) // IGetQueries
        .TryGetTournamentsWithTeamsAndPlayers(OrgId, out IEnumerable<Tournament>? tournaments);
    _tournament = tournaments.Single(t:Tournament => t.TournamentId = TourId);
    _roundAlgorithm = IRoundRobin.Initialize(_context, _tournament);
}
```

Рисунок 4.10 – SetUp-метод тестирования кругового алгоритма

Добавим метод добавления игроков из списка их имен и состояния активности.

```

private void AddPlayers((string, string)[] playerNames, bool isActive = true)
{
    foreach ((string, string) fullName in playerNames)
    {
        IInsertQueries.CreateInstance(_context) // IInsertQueries
            .TryAddPlayer(out Player _, 
                tournamentId: TourId,           organizerId: OrgId,
                lastName: fullName.Item1,      firstName: fullName.Item2,
                isActive: isActive);
    }
}

```

Рисунок 4.11 – Метод добавления игроков по их именам и активности

Из-за того, что очистка игроков тестового турнира происходит при вызове каждого метода, было принято решение вынести очистку игроков в отдельный метод в классе unit-тестирования (см. рисунок 4.16).

```

private void ClearPlayers()
{
    IGetQueries.CreateInstance(_context) // IGetQueries
        .TryGetPlayers(organizerId: OrgId, tournamentId: TourId,
            out IEnumerable<Player>? players);
    if (players is null)
    {
        Assert.Fail(message: "No players found.");
        return;
    }

    IEnumerable<Player> playersEnum = players.ToArray();
    for (var i = 0; i < playersEnum.Count(); i++)
    {
        Player player = playersEnum.ElementAt(i);
        IDeleteQueries.CreateInstance(_context).TryDeletePlayer(player);
    }
}

```

Рисунок 4.12 – Метод очистки игроков тестируемого турнира

4.1.2.1 Тестирование жеребьевки первого тура

Рассмотрим следующие случаи:

- 1.1. Активных игроков нет.
- 1.1.1. Список игроков пуст.
- 1.1.2. Только неактивные игроки (один).
- 1.1.3. Только неактивные игроки (четное количество).
- 1.1.4. Только неактивные игроки (нечетное количество).

Протестируем случай 1.1.1 (список игроков пуст). Для этого убедимся, что в турнире нет игроков, и попробуем провести жеребьевку (см. рисунок 4.11).

В данном тесте мы пытаемся провести жеребьевку первого тура без игроков, ожидая, что нам вернется пустой список пар.

```
// 1.1.1 The list of players is empty.  
[Test]  
◇  aleks*  
public void DrawFirstRound_WhenNoPlayers_ReturnsEmptyList()  
{  
    IGetQueries.CreateInstance(_context) // IGetQueries  
        .TryGetTournamentsWithTeamsAndPlayers(OrgId,  
                                              out IEnumerable<Tournament>? tournaments);  
    _tournament = tournaments.Single(t:Tournament => t.TournamentId == TourId);  
    // Arrange.  
    ClearPlayers();  
    _roundAlgorithm = IRoundRobin.Initialize(_context, _tournament);  
  
    // Act.  
    IEnumerable<(int, int)> result = _roundAlgorithm.StartNewTour(0);  
  
    // Assert.  
    Assert.IsEmpty(result);  
  
    ClearPlayers();  
}
```

Рисунок 4.13 – Тест случая 1.1.1

В результате теста была выявлена ошибка – код алгоритма не был готов к данной ситуации, из-за чего его пришлось модифицировать (см. рисунок 4.12).

Теперь он имеет проверку на то, чтобы номер нового тура был меньше количества определенных для турнира туров.

```
◆  0+2 usages  1 test OK  aleks *
public IList<(int, int)> StartNewTour(int currentTour)
{
    NewTourNumber = currentTour + 1;
    int tour = NewTourNumber;
    if (tour >= _pairsForTour.Count)
    {
        return new List<(int, int)>();
    }

    List<(int, int)> result = new();
    foreach ((int, int) pair in _pairsForTour[tour])
    {
        if (GamesHistory.Contains(pair))
        {
            tour++;
            continue;
        }

        result.Add(pair);
    }

    return result;
}
```

Рисунок 4.14 – Модифицированный метод жеребьевки нового тура

Протестируем случай 1.1.2 (только неактивные игроки (один)). Для этого убедимся, что в турнире только один игрок и он неактивный, и попробуем провести жеребьевку (см. рисунок 4.13).

В данном тесте мы пытаемся провести жеребьевку первого тура без игроков, ожидая, что нам вернется пустой список пар.

Тест не выявил ошибок.

```
// 1.1.2. Only inactive players (one).
[Test]
◇ new *
public void DrawFirstRound_WhenOnlyOneInactivePlayer_ReturnsEmptyList()
{
    IGetQueries.CreateInstance(_context) // IGetQueries
        .TryGetTournamentsWithTeamsAndPlayers(OrgId, out IEnumerable<Tournament>? tournaments);
    _tournament = tournaments.Single(t:Tournament => t.TournamentId == TourId);
    // Arrange.
    ClearPlayers();

    IIInsertQueries.CreateInstance(_context) // IIInsertQueries
        .TryAddPlayer(out Player _, 
            tournamentId: TourId,   organiserId: OrgId,
            lastName: "Петров",   firstName: "Петр",
            isActive: false);
    _roundAlgorithm = IRoundRobin.Initialize(_context, _tournament);

    // Act.
    IEnumerable<int, int> result = _roundAlgorithm.StartNewTour(0);

    // Assert.
    Assert.AreEqual(expected: 1, actual: _tournament.Players.Count);
    Assert.AreEqual(expected: 0, actual: _tournament.Players.Count(p:Player => p.IsActive ?? false));
    Assert.IsEmpty(result);

    ClearPlayers();
}
```

Рисунок 4.15 – Тестирование случая 1.1.2

Протестируем случай 1.1.3 (только неактивные игроки (четное количество)). Для этого убедимся, что в турнире четыре игрока и они неактивные, и попробуем провести жеребьевку (см. рисунок 4.14).

В данном тесте мы пытаемся провести жеребьевку первого тура без игроков, ожидая, что нам вернется пустой список пар.

```
// 1.1.3. Only inactive players (even number).
[Test]
◇ new *
public void DrawFirstRound_WhenOnlyEvenNumberOfInactivePlayers_ReturnsEmptyList()
{
    IGetQueries.CreateInstance(_context) // IGetQueries
        .TryGetTournamentsWithTeamsAndPlayers(OrgId,
            out IEnumerable<Tournament>? tournaments);
    _tournament = tournaments.Single(t:Tournament => t.TournamentId == TourId);
    // Arrange.
    ClearPlayers();

    (string, string)[] players = { ("Петров", "Петр"), ("Иванов", "Иван"),
        ("Сидоров", "Сидор"), ("Смирнов", "Сергей") };
    AddPlayers(players, isActive: false);

    _roundAlgorithm = IRoundRobin.Initialize(_context, _tournament);

    // Act.
    IEnumerable<(int, int)> result = _roundAlgorithm.StartNewTour(0);

    // Assert.
    Assert.AreEqual(expected: 4, actual: _tournament.Players.Count);
    Assert.AreEqual(expected: 0, actual: _tournament.Players.Count(p:Player => p.IsActive ?? false));
    Assert.IsEmpty(result);

    ClearPlayers();
}
```

Рисунок 4.16 – Тест случая 1.1.3

В результате теста была выявлена ошибка: алгоритм игнорирует неактивных игроков и продолжает добавлять их в жеребьевку.

Для исправления этой ошибки необходимо в методе инициализации алгоритма добавить фильтрацию на состояние активности игроков (см. рисунок 4.15).

```

public RoundRobin(ChessTourContext context, Tournament tournament)
{
    IGetQueries.CreateInstance(context) // IGetQueries
        .TryGetGames(organiserId: tournament.OrganizerId,
                     tournament.TournamentId,
                     out IEnumerable<Game>? games);
    if (games is not null)
    {
        GamesHistory = games.Select(g:Game => (g.WhiteId, g.BlackId)).ToHashSet();
    }

    IGetQueries.CreateInstance(context) // IGetQueries
        .TryGetPlayers(organiserId: tournament.OrganizerId, tournament.TournamentId,
                        out IEnumerable<Player>? players);

    PlayersIds = players?.Where(p:Player => p.IsActive ?? false) // IEnumerable<Player>
        .Select(p:Player => p.PlayerId).ToList() ?? new List<int>();

    ConfigureTours();
}

```

Рисунок 4.17 – Исправление игнорирования активности игроков алгоритмом

Запустим тест заново. Теперь все работает корректно – неактивные игроки игнорируются алгоритмом.

Протестируем случай 1.1.4 (только неактивные игроки (нечетное количество)). Для этого убедимся, что в турнире пять игроков и они неактивные, и попробуем провести жеребьевку (см. рисунок 4.14).

```
// 1.1.4. Only inactive players (odd number).
[Test]
◇ new *
public void DrawFirstRound_WhenOnlyOddNumberOfInactivePlayers_ReturnsEmptyList()
{
    IGetQueries.CreateInstance(_context) // IGetQueries
        .TryGetTournamentsWithTeamsAndPlayers(OrgId,
            out IEnumerable<Tournament>? tournaments);
    _tournament = tournaments.Single(t:Tournament => t.TournamentId == TourId);
    // Arrange.
    ClearPlayers();

    (string, string)[] players = { ("Петров", "Петр"), ("Иванов", "Иван"),
        ("Сидоров", "Сидор"), ("Смирнов", "Сергей"),
        ("Кузнецов", "Андрей") };
    AddPlayers(players, isActive: false);

    _roundAlgorithm = IRoundRobin.Initialize(_context, _tournament);

    // Act.
    IEnumerable<(int, int)> result = _roundAlgorithm.StartNewTour(0);

    // Assert.
    Assert.AreEqual(expected: 5, actual: _tournament.Players.Count);
    Assert.AreEqual(expected: 0, actual: _tournament.Players.Count(p:Player => p.IsActive ?? false));
    Assert.IsEmpty(result);

    ClearPlayers();
}
```

Рисунок 4.18 – Тест случая 1.1.4

В данном тесте мы пытаемся провести жеребьевку первого тура без игроков, ожидая, что нам вернется пустой список пар.

Тест не выявил ошибок.

Рассмотрим следующие случаи:

1.2. Активный игрок один.

1.2.1. Один активный.

1.2.2. Один активный и один неактивный.

1.2.3. Один активный и четное число неактивных.

1.2.4. Один активный и нечетное число неактивных.

Протестируем случай 1.2.1 (один активный). Для этого убедимся, что в турнире ровно один игрок и он активный, и попробуем провести жеребьевку (см. рисунок 4.19).

Тест не выявил ошибок, был возвращен пустой список пар.

```
// 1.2.1. One active player.
[Test]
◇ aleks*
public void DrawFirstRound_WhenOnePlayer_ReturnsEmptyList()
{
    IGetQueries.CreateInstance(_context) // IGetQueries
        .TryGetTournamentsWithTeamsAndPlayers(OrgId,
            out IEnumerable<Tournament>? tournaments);
    _tournament = tournaments.Single(t:Tournament => t.TournamentId == TourId);
    // Arrange.
    ClearPlayers();

    AddPlayers(playerNames: new[] { ("Петров", "Петр") }, isActive: true);
    _roundAlgorithm = IRoundRobin.Initialize(_context, _tournament);

    // Act.
    IEnumerable<(int, int)> result = _roundAlgorithm.StartNewTour(0);

    // Assert.
    Assert.IsEmpty(result);
    Assert.AreEqual(expected: 1, actual: _tournament.Players.Count);

    ClearPlayers();
}
```

Рисунок 4.19 – Тест случая 1.2.1

Протестируем случай 1.2.2 (один активный и один неактивный). Для этого убедимся, что в турнире два игрока: один активный, другой – нет, и попробуем провести жеребьевку (см. рисунок 4.19).

Тест не выявил ошибок.

```
// 1.2.2. One active and one inactive player.  
[Test]  
◇ new *  
public void DrawFirstRound_WhenOneActiveAndOneInactivePlayer_ReturnsEmptyList()  
{  
    IGetQueries.CreateInstance(_context) //IGetQueries  
        .TryGetTournamentsWithTeamsAndPlayers(OrgId,  
                                                out IEnumerable<Tournament>? tournaments);  
    _tournament = tournaments.Single(t:Tournament => t.TournamentId == TourId);  
    // Arrange.  
    ClearPlayers();  
  
    AddPlayers(playerNames: new[] { ("Петров", "Петр") }, isActive: true);  
    AddPlayers(playerNames: new[] { ("Сидоров", "Сидор") }, isActive: false);  
    _roundAlgorithm = IRoundRobin.Initialize(_context, _tournament);  
  
    // Act.  
    IEnumerable<(int, int)> result = _roundAlgorithm.StartNewTour(0);  
  
    // Assert.  
    Assert.IsEmpty(result);  
    Assert.AreEqual(expected: 2, actual: _tournament.Players.Count);  
    Assert.AreEqual(expected: 1, actual: _tournament.Players.Count(p:Player => p.IsActive ?? false));  
  
    ClearPlayers();  
}
```

Рисунок 4.20 – Тест случая 1.2.2

Протестируем случай 1.2.3 (один активный и четное число неактивных). Для этого убедимся, что в турнире 5 игроков: один активный, остальные – нет, и попробуем провести жеребьевку (см. рисунок 4.19).

Тест не выявил ошибок.

```
// 1.2.3. One active and even number of inactive players (4).
[Test]
◇ new *
public void DrawFirstRound_WhenOneActiveAndEvenNumberOfInactivePlayers_ReturnsEmptyList()
{
    IGetQueries.CreateInstance(_context) // IGetQueries
        .TryGetTournamentsWithTeamsAndPlayers(OrgId,
            out IEnumerable<Tournament>? tournaments);
    _tournament = tournaments.Single(t:Tournament => t.TournamentId = TourId);
    // Arrange.
    ClearPlayers();

    AddPlayers(playerNames: new[] { ("Петров", "Петр") }, isActive: true);
    (string, string)[] players = { ("Иванов", "Иван"), ("Сидоров", "Сидор"),
        ("Смирнов", "Сергей"), ("Кузнецов", "Андрей") };
    AddPlayers(players, isActive: false);
    _roundAlgorithm = IRoundRobin.Initialize(_context, _tournament);

    // Act.
    IEnumerable<(int, int)> result = _roundAlgorithm.StartNewTour(0);

    // Assert.
    Assert.IsEmpty(result);
    Assert.AreEqual(expected: 5, actual: _tournament.Players.Count);
    Assert.AreEqual(expected: 1, actual: _tournament.Players.Count(p:Player => p.IsActive ?? false));

    ClearPlayers();
}
```

Рисунок 4.21 – Тест случая 1.2.3

Протестируем случай 1.2.4 (один активный и нечетное число неактивных).

Для этого убедимся, что в турнире 6 игроков: один активный, остальные – нет, и попробуем провести жеребьевку (см. рисунок 4.19).

Тест не выявил ошибок – неактивные игроки продолжают игнорироваться, активных игроков всего один – жеребьевка невозможна.

```
// 1.2.4. One active and odd number of inactive players (5).
[Test]
◇ new *
public void DrawFirstRound_WhenOneActiveAndOddNumberOfInactivePlayers_ReturnsEmptyList()
{
    IGetQueries.CreateInstance(_context) // IGetQueries
        .TryGetTournamentsWithTeamsAndPlayers(_orgId,
                                                out IEnumerable<Tournament>? tournaments);
    _tournament = tournaments.Single(t:Tournament => t.TournamentId == _tourId);
    // Arrange.
    ClearPlayers();

    AddPlayers(playerNames: new[] { ("Петров", "Петр") }, isActive: true);
    (string, string)[] players = { ("Иванов", "Иван"), ("Сидоров", "Сидор"),
                                  ("Смирнов", "Сергей"), ("Кузнецов", "Андрей"),
                                  ("Петров", "Андрей") };
    AddPlayers(players, isActive: false);
    _roundAlgorithm = IRoundRobin.Initialize(_context, _tournament);

    // Act.
    IEnumerable<(int, int)> result = _roundAlgorithm.StartNewTour(0);

    // Assert.
    Assert.IsEmpty(result);
    Assert.AreEqual(expected: 6, actual: _tournament.Players.Count);
    Assert.AreEqual(expected: 1, actual: _tournament.Players.Count(p:Player => p.IsActive ?? false));

    ClearPlayers();
}
```

Рисунок 4.22 – Тест случая 1.2.4

Рассмотрим следующие случаи:

1.3. Активных игроков четное количество.

1.3.1. Отсутствуют неактивные.

1.3.2. Присутствует один неактивный.

1.3.3. Присутствует четное число неактивных.

1.3.4. Присутствует нечетное число неактивных.

Протестируем случай 1.3.1 (активных игроков четное количество и отсутствуют неактивные). Для этого убедимся, что в турнире четное число игроков (например – 4), а неактивных нет. Попробуем провести жеребьевку первого тура (см. рисунок 4.23).

В результате алгоритм вернул 2 пары игроков, как и предполагалось.

```
// 1.3.1. Even number of active players (4).
[Test]
◇ aleks *
public void DrawFirstRound_WhenEvenNumberOfPlayers_ReturnsCorrectPairs()
{
    IGetQueries.CreateInstance(_context) // IGetQueries
        .TryGetTournamentsWithTeamsAndPlayers(_orgId,
            out IEnumerable<Tournament>? tournaments);
    _tournament = tournaments.Single(t:Tournament => t.TournamentId == _tourId);
    // Arrange.
    ClearPlayers();

    (string, string)[] playerNames = { ("Петров", "Петр"), ("Иванов", "Иван"),
        ("Сидоров", "Сидор"), ("Сергеев", "Сергей") };
    AddPlayers(playerNames, isActive: true);

    _roundAlgorithm = IRoundRobin.Initialize(_context, _tournament);

    // Act.
    IEnumerable<(int, int)> result = _roundAlgorithm.StartNewTour(0);

    // Assert.
    Assert.AreEqual(expected: 4, actual: _tournament.Players.Count);
    Assert.AreEqual(expected: 2, actual: result.Count());
}

ClearPlayers();
}
```

Рисунок 4.23 – Тест случая 1.3

Протестируем случай 1.3.2 (активных игроков четное количество и присутствует один неактивный). Для этого убедимся, что в турнире четное число игроков (например – 4) и один неактивный. Попробуем провести жеребьевку первого тура (см. рисунок 4.23).

В результате алгоритм вернул 2 пары игроков, как и предполагалось.

```
// 1.3.2. Active players are even number (4) and There is one inactive one.  
[Test]  
◇ new *  
public void DrawFirstRound_WhenEvenNumberOfPlayersAndOneInactivePlayer_ReturnsCorrectPairs()  
{  
    IGetQueries.CreateInstance(_context) //IGetQueries  
        .TryGetTournamentsWithTeamsAndPlayers(OrgId,  
                                                out IEnumerable<Tournament>? tournaments);  
    _tournament = tournaments.Single(t:Tournament => t.TournamentId == TourId);  
    // Arrange.  
    ClearPlayers();  
  
    (string, string)[] playerNames = { ("Петров", "Петр"), ("Иванов", "Иван"),  
                                      ("Сидоров", "Сидор"), ("Сергеев", "Сергей") };  
    AddPlayers(playerNames, isActive: true);  
    AddPlayers(playerNames: new[] { ("Алексеев", "Алексей") }, isActive: false);  
  
    _roundAlgorithm = IRoundRobin.Initialize(_context, _tournament);  
  
    // Act.  
    IEnumerable<(int, int)> result = _roundAlgorithm.StartNewTour(0);  
  
    // Assert.  
    Assert.AreEqual(expected: 5, actual: _tournament.Players.Count);  
    Assert.AreEqual(expected: 2, actual: result.Count());  
  
    ClearPlayers();  
}
```

Рисунок 4.24 – Тест случая 1.3.2

Протестируем случай 1.3.3 (активных игроков четное количество и присутствует четное число неактивных). Для этого убедимся, что в турнире четное число игроков (например – 4) и 4 неактивных. Попробуем провести жеребьевку первого тура (см. рисунок 4.23).

В результате алгоритм вернул 2 пары игроков, как и предполагалось.

```
// 1.3.3. Active players are even number (4) and inactive players are even number (4).
[Test]
◇ new *
public void DrawFirstRound_WhenEvenNumberOfPlayersAndEvenNumberOfInactivePlayers_ReturnsCorrectPairs()
{
    IGetQueries.CreateInstance(_context) // IGetQueries
        .TryGetTournamentsWithTeamsAndPlayers(orgId,
                                                out IEnumerable<Tournament>? tournaments);
    _tournament = tournaments.Single(t:Tournament => t.TournamentId == TourId);
    // Arrange.
    ClearPlayers();

    (string, string)[] playerNames = { ("Петров", "Петр"), ("Иванов", "Иван"),
                                       ("Сидоров", "Сидор"), ("Сергеев", "Сергей") };
    AddPlayers(playerNames, isActive: true);
    (string, string)[] inactivePlayers = { ("Алексеев", "Алексей"), ("Андреев", "Андрей"),
                                           ("Антонов", "Антон"), ("Артемов", "Артем") };
    AddPlayers(inactivePlayers, isActive: false);

    _roundAlgorithm = IRoundRobin.Initialize(_context, _tournament);

    // Act.
    IEnumerable<(int, int)> result = _roundAlgorithm.StartNewTour(0);

    // Assert.
    Assert.AreEqual(expected: 8, actual: _tournament.Players.Count);
    Assert.AreEqual(expected: 2, actual: result.Count());
}

ClearPlayers();
}
```

Рисунок 4.25 – Тестирование случая 1.3.3

Протестируем случай 1.3.4 (активных игроков четное количество и присутствует нечетное число неактивных). Для этого убедимся, что в турнире четное число игроков (например – 4) и 5 неактивных. Попробуем провести жеребьевку первого тура (см. рисунок 4.23).

В результате алгоритм вернул 2 пары игроков, как и предполагалось.

```
// 1.3.4. Active players are even number (4) and inactive players are odd number (5).
[Test]
◇ new *
public void DrawFirstRound_WhenEvenNumberOfPlayersAndOddNumberOfInactivePlayers_ReturnsCorrectPairs()
{
    IGetQueries.CreateInstance(_context) //IGetQueries
        .TryGetTournamentsWithTeamsAndPlayers(_orgId,
                                                out IEnumerable<Tournament>? tournaments);
    _tournament = tournaments.Single(t:Tournament => t.TournamentId == _tourId);
    // Arrange.
    ClearPlayers();

    (string, string)[] activePlayers = { ("Петров", "Петр"), ("Иванов", "Иван"),
                                         ("Сидоров", "Сидор"), ("Сергеев", "Сергей") };
    AddPlayers(activePlayers, isActive: true);
    (string, string)[] inactivePlayers = { ("Алексеев", "Алексей"), ("Андреев", "Андрей"),
                                           ("Антонов", "Антон"), ("Артемов", "Артем"),
                                           ("Борисов", "Борис") };
    AddPlayers(inactivePlayers, isActive: false);

    _roundAlgorithm = IRoundRobin.Initialize(_context, _tournament);

    // Act.
    IEnumerable<(int, int)> result = _roundAlgorithm.StartNewTour(0);

    // Assert.
    Assert.AreEqual(expected: 9, actual: _tournament.Players.Count);
    Assert.AreEqual(expected: 2, actual: result.Count());
}

ClearPlayers();
}
```

Рисунок 4.26 – Тестирование случая 1.3.4

Рассмотрим следующие случаи:

1.4. Активных игроков нечетное количество.

1.4.1. Отсутствуют неактивные.

1.4.2. Присутствует один неактивный.

1.4.3. Присутствует четное число неактивных.

1.4.4. Присутствует нечетное число неактивных.

Протестируем случай 1.4.1 (активных игроков нечетное количество и отсутствуют неактивные). Для этого убедимся, что активных игроков нечетное количество (например – 5), а неактивных нет. Попробуем провести жеребьевку первого тура.

На вход подается коллекция из 5 игроков, на выходе ожидается 3 пары – 2 пары игроков и в последней паре игрок без соперника – вместо соперника записывается код -1 (см. рисунок 4.24).

В результате теста обнаружилась ошибка – алгоритм игнорирует нечетность игроков, для этого необходимо добавить проверку на нечетность, и добавить -1 в список ID игроков («игрок-заглушка»), если количество игроков нечетно, таким образом компенсируется отсутствие игрока (см. рисунок 4.25).

```
// 1.4.1 Odd number of active players (5).
[Test]
◇ aleks *
public void DrawFirstRound_WhenOddNumberOfPlayers_ReturnsCorrectPairs()
{
    IGetQueries.CreateInstance(_context) //IGetQueries
        .TryGetTournamentsWithTeamsAndPlayers(_orgId,
            out IEnumerable<Tournament>? tournaments);
    _tournament = tournaments.Single(t:Tournament => t.TournamentId == _tourId);
    // Arrange.
    ClearPlayers();
    (string, string)[] playerNames = { ("Петров", "Петр"), ("Иванов", "Иван"),
        ("Сидоров", "Сидор"), ("Сергеев", "Сергей"),
        ("Алексеев", "Алексей") };
    AddPlayers(playerNames, isActive: true);

    _roundAlgorithm = IRoundRobin.Initialize(_context, _tournament);

    // Act.
    IEnumerable<(int, int)> result = _roundAlgorithm.StartNewTour(0);

    // Assert.
    Assert.AreEqual(expected: 5, actual: _tournament.Players.Count);
    Assert.AreEqual(expected: 3, actual: result.Count());
    Assert.IsTrue(result.Any(pair:(int,int) => pair.Item1 == -1 || pair.Item2 == -1));

    ClearPlayers();
}
```

Рисунок 4.27 – Тестирование случая 1.5.1

Теперь тест пройден. После изменения кода алгоритма необходимо перезапустить все его тесты, чтобы убедиться, что они до сих пор выполняются корректно.

В результате исправления предыдущей ошибки (при нечетном количестве игроков игрок без пары не появлялся в списке пар), появилась новая – теперь если в турнире участвует всего один игрок, то вместо пустого списка пар, возвращается список из одной пары, состоящей из этого игрока и «игрока-заглушки» (для поддержания четности количества игроков).

Чтобы исправить это, необходимо при составлении пар добавить явную проверку на то, чтобы игроков было более 2.

```
public IList<(int, int)> StartNewTour(int currentTour)
{
    NewTourNumber = currentTour + 1;
    int tour = NewTourNumber;
    if (tour >= _pairsForTour.Count || PlayersIds.Count < 3)
    {
        MessageBox.Show(messageBoxText: "Недостаточно игроков для "
            + "проведения следующего тура.",
            caption: "Ошибка", MessageBoxButtons.OK,
            MessageBoxIcon.Error);
        return new List<(int, int)>();
    }

    List<(int, int)> result = new();
    foreach ((int, int) pair in _pairsForTour[tour])
    {
        if (GamesHistory.Contains(pair))
        {
            tour++;
            continue;
        }

        result.Add(pair);
    }

    return result;
}
```

Рисунок 4.28 – Ограничение на количество игроков в методе старта нового тура

Протестируем случай 1.5 (присутствие неактивных участников). Для этого сделаем некоторых игроков активными.

Протестируем случай 1.4.2 (активных игроков нечетное количество и присутствует один неактивный). Для этого убедимся, что активных игроков нечетное количество (например – 5) и один неактивный. Попробуем провести жеребьевку первого тура.

Тест не выявил ошибок.

```
// 1.4.2. Odd number of active players (5) and one inactive player.  
[Test]  
◇ new *  
public void DrawFirstRound_WhenOddNumberOfPlayersAndOneInactivePlayer_ReturnsCorrectPairs()  
{  
    IGetQueries.CreateInstance(_context) // IGetQueries  
        .TryGetTournamentsWithTeamsAndPlayers(OrgId,  
                                                out IEnumerable<Tournament>? tournaments);  
    _tournament = tournaments.Single(t:Tournament => t.TournamentId = TourId);  
    // Arrange.  
    ClearPlayers();  
    (string, string)[] playerNames = { ("Петров", "Петр"), ("Иванов", "Иван"),  
                                      ("Сидоров", "Сидор"), ("Сергеев", "Сергей"),  
                                      ("Алексеев", "Алексей") };  
    AddPlayers(playerNames, isActive: true);  
    AddPlayers(playerNames: new[] { ("Андреев", "Андрей") }, isActive: false);  
  
    _roundAlgorithm = IRoundRobin.Initialize(_context, _tournament);  
  
    // Act.  
    IEnumerable<(int, int)> result = _roundAlgorithm.StartNewTour(0);  
  
    // Assert.  
    Assert.AreEqual(expected: 6, actual: _tournament.Players.Count);  
    Assert.AreEqual(expected: 3, actual: result.Count());  
    Assert.IsTrue(result.Any(pair:(int,int) => pair.Item1 == -1 || pair.Item2 == -1));  
  
    ClearPlayers();  
}
```

Рисунок 4.29 – Тест случая 1.4.2

Протестируем случай 1.4.3 (активных игроков нечетное количество и присутствует четное число неактивных). Для этого убедимся, что активных игроков нечетное количество (например – 5) и 2 неактивных. Попробуем провести жеребьевку первого тура.

Тест не выявил ошибок.

```
// 1.4.3. Odd number of active players (5) and two inactive players.  
[Test]  
◇ new *  
public void DrawFirstRound_WhenOddNumberOfPlayersAndTwoInactivePlayers_ReturnsCorrectPairs()  
{  
    IGetQueries.CreateInstance(_context) // IGetQueries  
        .TryGetTournamentsWithTeamsAndPlayers(_orgId,  
            out IEnumerable<Tournament>? tournaments);  
    _tournament = tournaments.Single(t:Tournament => t.TournamentId == _tourId);  
    // Arrange.  
    ClearPlayers();  
    (string, string)[] playerNames = { ("Петров", "Петр"), ("Иванов", "Иван"),  
        ("Сидоров", "Сидор"), ("Сергеев", "Сергей"),  
        ("Алексеев", "Алексей") };  
    AddPlayers(playerNames, isActive: true);  
    AddPlayers(playerNames: new[] { ("Андреев", "Андрей"), ("Антонов", "Антон") }, isActive: false);  
  
    _roundAlgorithm = IRoundRobin.Initialize(_context, _tournament);  
  
    // Act.  
    IEnumerable<(int, int)> result = _roundAlgorithm.StartNewTour(0);  
  
    // Assert.  
    Assert.AreEqual(expected: 7, actual: _tournament.Players.Count);  
    Assert.AreEqual(expected: 3, actual: result.Count());  
    Assert.IsTrue(result.Any(pair:(int,int) => pair.Item1 == -1 || pair.Item2 == -1));  
  
    ClearPlayers();  
}
```

Рисунок 4.30 – Тест случая 1.4.3

Протестируем случай 1.4.4 (активных игроков нечетное количество и присутствует нечетное число неактивных). Для этого убедимся, что активных игроков нечетное количество (например – 5) и 3 неактивных. Попробуем провести жеребьевку первого тура.

Тест не выявил ошибок.

```
// 1.4.4. Odd number of active players (5) and three inactive players.
[Test]
◇ new *
public void DrawFirstRound_WhenOddNumberOfPlayersAndThreeInactivePlayers_ReturnsCorrectPairs()
{
    IGetQueries.CreateInstance(_context) // IGetQueries
        .TryGetTournamentsWithTeamsAndPlayers(OrgId,
                                                out IEnumerable<Tournament>? tournaments);
    _tournament = tournaments.Single(t:Tournament => t.TournamentId == TourId);
    // Arrange.
    ClearPlayers();
    (string, string)[] playerNames = { ("Петров", "Петр"), ("Иванов", "Иван"),
                                       ("Сидоров", "Сидор"), ("Сергеев", "Сергей"),
                                       ("Алексеев", "Алексей") };
    AddPlayers(playerNames, isActive: true);
    AddPlayers(playerNames: new[] { ("Андреев", "Андрей"), ("Антонов", "Антон"),
                                   ("Артемов", "Артем") },
               isActive: false);

    _roundAlgorithm = IRoundRobin.Initialize(_context, _tournament);

    // Act.
    IEnumerable<(int, int)> result = _roundAlgorithm.StartNewTour(0);

    // Assert.
    Assert.AreEqual(expected: 8, actual: _tournament.Players.Count);
    Assert.AreEqual(expected: 3, actual: result.Count());
    Assert.IsTrue(result.Any(pair:(int,int) => pair.Item1 == -1 || pair.Item2 == -1));

    ClearPlayers();
}
```

Рисунок 4.31 – Тест случая 1.4.4

4.1.2.2 Тестирование жеребьевки не первого тура

Рассмотрим следующие случаи:

2. Жеребьевка не первого тура.
 - 2.1. Отсутствуют активные игроки.
 - 2.2. Активный игрок один.
 - 2.3. Активных игроков четное количество.
 - 2.4. Активных игроков нечетное количество.

Тестируем будем на втором туре. Для этого будем проводить первый так, чтобы он выполнялись корректно (активных игроков больше 2).

Протестируем случай 2.1 (список игроков пуст). Для этого проведем первый тур из 4 активных игроков, а потом сделаем их неактивными и попробуем провести жеребьевку (см. рисунок 4.11).

```
// 2.1.1. No players in second round.  
[Test]  
◇ new *  
public void DrawNotFirstRound_WhenNoPlayers_ReturnsEmptyList()  
{  
    IGetQueries.CreateInstance(_context) //IGetQueries  
        .TryGetTournamentsWithTeamsAndPlayers(_orgId,  
            out IEnumerable<Tournament>? tournaments);  
    _tournament = tournaments.Single(t:Tournament => t.TournamentId == _tourId);  
    // Arrange.  
    ClearPlayers();  
    (string, string)[] playerNames = { ("Петров", "Петр"), ("Иванов", "Иван"),  
        ("Сидоров", "Сидор"), ("Сергеев", "Сергей") };  
    AddPlayers(playerNames, isActive: true);  
    _roundAlgorithm = IRoundRobin.Initialize(_context, _tournament);  
  
    // Act.  
    // Start first round.  
    IEnumerable<int, int> result = _roundAlgorithm.StartNewTour(0);  
    Assert.AreEqual(expected: 2, actual: result.Count());  
  
    // Make all players inactive.  
    SetAllActivity(isActive: false);  
  
    // Start second round.  
    result = _roundAlgorithm.StartNewTour(1);  
  
    // Assert.  
    Assert.IsEmpty(result);  
  
    ClearPlayers();  
}
```

Рисунок 4.32 – Тест случая 2.1

Данный тест выявил ошибку: при изменении состояния игроков (смена активности, добавление) эти изменения не фиксируются алгоритмом, он продолжает выполнять операции со старым списком.

Решено было добавить приватное свойство турнира для получения актуального списка игроков и проверку в метод начала нового тура на то, не изменился ли список ID игроков. Если списки окажутся не равны, то необходимо перенастроить пары.

```

public IList<int, int> StartNewTour(int currentTour)
{
    ReconfigureIdsIfPlayersChanged();

    NewTourNumber = currentTour + 1;
    int tour = NewTourNumber;
    if (tour >= _pairsForTour.Count || _playersIds.Count < 3)
    {
        MessageBox.Show(messageBoxText: "Недостаточно игроков для "
            + "проведения следующего тура.",
            caption: "Ошибка", MessageBoxButtons.OK,
            MessageBoxIcon.Error);
        return new List<int, int>();
    }

    List<int, int> result = new();
    foreach ((int, int) pair in _pairsForTour[tour])
    {
        if (GamesHistory.Contains(pair))
        {
            tour++;
            continue;
        }

        result.Add(pair);
    }

    return result;
}

```

Рисунок 4.33 – Новый метод старта жеребьевки

```

private void ReconfigureIdsIfPlayersChanged()
{
    // If players changed, reconfigure the tours.
    List<int> ids = GetPlayersIds(_tournament.Players);
    if (ids.Count != _playersIds.Count)
    {
        _playersIds = ids;
        ConfigureTours();
        return;
    }

    for (var i = 0; i < _playersIds.Count; i++)
    {
        if (_playersIds[i] != ids[i])
        {
            _playersIds = ids;
            ConfigureTours();
            break;
        }
    }
}

```

Рисунок 4.34 – Метод обновления списка ID игроков, если он поменялся

Перезапустим тест. Теперь все сработало корректно. Также необходимо запустить все написанные ранее тесты, чтобы убедиться, что новых ошибок не появилось.

Протестируем случай 2.2 (один активный игрок). Для этого проведем первый тур из 4 активных игроков, а потом сделаем трех из них неактивными и попробуем провести жеребьевку (см. рисунок 4.11).

Ожидаем корректную жеребьевку на первом туре и отсутствие пар на втором.

Тест не нашел ошибок.

```
// 2.2. One player in second round.

[Test]
◇ new *
public void DrawNotFirstRound_WhenOnePlayer_ReturnsCorrectPair()
{
    IGetQueries.CreateInstance(_context) // IGetQueries
        .TryGetTournamentsWithTeamsAndPlayers(_orgId,
                                                out IEnumerable<Tournament>? tournaments);
    _tournament = tournaments.Single(t:Tournament => t.TournamentId == _tourId);
    // Arrange.
    ClearPlayers();
    (string, string)[] playerNames = { ("Петров", "Петр"), ("Иванов", "Иван"),
                                       ("Сидоров", "Сидор"), ("Сергеев", "Сергей") };
    _roundAlgorithm = IRoundRobin.Initialize(_context, _tournament);
    AddPlayers(playerNames, isActive: true);

    // Act.
    // Start first round.
    IEnumerable<int, int> result = _roundAlgorithm.StartNewTour(0);
    Assert.AreEqual(expected: 2, actual: result.Count());

    // Make all players inactive except one.
    SetAllActivity(isActive: false);
    _tournament.Players.First().IsActive = true;

    // Start second round.
    result = _roundAlgorithm.StartNewTour(1);

    // Assert.
    Assert.AreEqual(expected: 0, actual: result.Count());

    ClearPlayers();
}
```

Рисунок 4.35 – Тест случая 2.2

Протестируем случай 2.3 (активных игроков четное количество). Для этого проведем первый тур из 7 активных игроков, а потом сделаем трех из них неактивными и попробуем провести жеребьевку (см. рисунок 4.11).

Ожидаем корректную жеребьевку на 1 туре с появлением дополнительного игрока, так как нечетное число игроков, затем на 2 туре ожидаем, что этот игрок пропадет, так как 4 игрока смогут сыграть друг с другом.

```
// Arrange.
ClearPlayers();
(string, string)[] playerNames = { ("Петров", "Петр"), ("Иванов", "Иван"),
                                     ("Сидоров", "Сидор"), ("Сергеев", "Сергей"),
                                     ("Алексеев", "Алексей"), ("Андреев", "Андрей"),
                                     ("Антонов", "Антон") };
_roundAlgorithm = IRoundRobin.Initialize(_context, _tournament);
AddPlayers(playerNames, isActive: true);

// Act.
// Start first round.
IEnumerable<(int, int)> result = _roundAlgorithm.StartNewTour(0);
Assert.AreEqual(expected: 4, actual: result.Count());

// Check if there is one player with -1 in pair.
Assert.IsTrue(result.Any(pair:(int,int) => pair.Item1 == -1 || pair.Item2 == -1));

// Make 3 players inactive.
_tournament.Players.Take(3).ToList().ForEach(p:Player => p.IsActive = false);

// Start second round.
result = _roundAlgorithm.StartNewTour(1);

// Assert.
Assert.AreEqual(expected: 2, actual: result.Count());
// Check if no player with -1 in pair.
Assert.IsFalse(result.Any(pair:(int,int) => pair.Item1 == -1 || pair.Item2 == -1));
```

Рисунок 4.36 – Тест случая 2.3

Данный тест выявил ошибку: при уменьшении количества игроков, не перезаписываются заранее определенные пары жеребьевок в словаре пар, для решения чего пришлось перепроверять, если номер тура уже есть в словаре, то его нужно перезаписать.

```
private void ConfigureTours()
{
    _playersIds.Sort();

    // Add a dummy player to make the number of players even.
    if (_playersIds.Count % 2 != 0)
    {
        _playersIds.Add(item: -1);
    }

    for (int tourNumber = NewTourNumber; tourNumber <= _playersIds.Count; tourNumber++)
    {
        if (_pairsForTour.ContainsKey(tourNumber))
        {
            _pairsForTour[tourNumber] = ConfigurePairs();
        }
        else
        {
            _pairsForTour.Add(tourNumber, ConfigurePairs());
        }

        // Rotate players, so that the first player is always the same.
        _playersIds.Add(_playersIds[0]);
        _playersIds.RemoveAt(index: 0);
    }
}
```

Рисунок 4.37 – Исправление метода настройки тура

После исправления этого теста необходимо перезапустить прошлые. Новых ошибок на предыдущих тестах не обнаружилось.

Протестируем случай 2.4 (активных игроков нечетное количество). Для этого проведем первый тур из 8 активных игроков, а потом сделаем трех из них неактивными и попробуем провести жеребьевку (см. рисунок 4.11).

Ожидаем, что на первом туре 8 игроков образуют 4 пары и сыграют, затем перед вторым туром 3 игрока уйдут, останется 5 – один дополнительный для поддержания четного количества игроков – 3 пары во втором туре.

```
.....  
// Arrange.  
ClearPlayers();  
string[] playerNames = { ("Петров", "Петр"), ("Иванов", "Иван"),  
    ("Сидоров", "Сидор"), ("Сергеев", "Сергей"),  
    ("Алексеев", "Алексей"), ("Андреев", "Андрей"),  
    ("Аntonov", "Аnton"), ("Артемов", "Артем") };  
_roundAlgorithm = IRoundRobin.Initialize(_context, _tournament);  
AddPlayers(playerNames, isActive: true);  
  
// Act.  
// Start first round.  
IEnumerable<int, int> result = _roundAlgorithm.StartNewTour(0);  
Assert.AreEqual(expected: 4, actual: result.Count());  
  
// Check if there is no player with -1 in pair.  
Assert.IsFalse(result.Any(pair:(int,int) => pair.Item1 == -1 || pair.Item2 == -1));  
  
// Make 3 players inactive.  
_tournament.Players.Take(3).ToList().ForEach(p:Player => p.IsActive = false);  
  
// Start second round.  
result = _roundAlgorithm.StartNewTour(1);  
  
// Assert.  
Assert.AreEqual(expected: 3, actual: result.Count());  
// Check if there is player with -1 in pair.  
Assert.IsTrue(result.Any(pair:(int,int) => pair.Item1 == -1 || pair.Item2 == -1));
```

Рисунок 4.38 – Тест случая 2.4

Тест не обнаружил ошибок.

4.2 Интеграционное тестирование

В интеграционном тестировании тестируется взаимосвязь модулей алгоритмов и доступа к данным: необходимо запустить алгоритм, получив разбиение преобразовать его в игры и отправить в базу данных.

4.2.1 Первый сценарий

План интеграционного тестирования: открыть существующий турнир с сыгранными партиями, попытаться запустить алгоритм жеребьевки, получить результат в таблице списка пар.

```

[Test]
◊
public void Test()
{
    // Get tournament.
    IGetQueries.CreateInstance(_context) // IGetQueries
        .TryGetTournaments(_orgId, out IEnumerable<Tournament>? tournaments);
    Tournament tournament = tournaments?.Single(t:Tournament => t.TournamentId == _tourId)
        ?? throw new InvalidOperationException();

    // Create mock view model.
    var viewModel = new PairsGridViewModel();

    // Create mock TournamentsList view model.
    var tournamentsListViewModel = new TournamentsListViewModel();

    // Invoke open tournament command.
    var openTournamentCommand = new OpenTournamentCommand(tournamentsListViewModel);
    openTournamentCommand.Execute(tournament);

    int oldTourNumber = viewModel.CurrentTour;
    |
    // Execute start tour command.
    viewModel.StartNewTour.Execute(parameter: null);

    // Check result.
    Assert.AreEqual(expected: oldTourNumber + 1, actual: viewModel.CurrentTour);
}

```

Рисунок 4.39 – Интеграционный тест запуска нового тура

В процессе тестирования был открыт турнир, в котором 8 игроков и 3 тура сыграны. При вызове жеребьевки 4 тура возникла ошибка: невозможно сформировать пары.

В процессе отладки была выявлена причина ошибки: при открытии турнира и вызове алгоритма сначала происходит настройка пар под первый тур, из-за чего программа вызывала исключение, что невозможно сформировать список пар.

В результате было найдено решение: при инициализации алгоритма добавить вычисление номера следующего тура.

Перезапуск unit-тестов после исправления не обнаружил ошибок.

```

public RoundRobin(ChessTourContext context, Tournament tournament)
{
    _tournament = tournament;
    IGetQueries.CreateInstance(context) // IGetQueries
        .TryGetGames(organiserId: tournament.OrganizerId,
                     tournament.TournamentId,
                     out IEnumerable<Game>? games);
    if (games is not null)
    {
        IEnumerable<Game> gamesEnum = games.ToList();
        GamesHistory = gamesEnum.Select(g:Game => (g.WhiteId, g.BlackId)).ToHashSet();

        if (gamesEnum.Any())
        {
            NewTourNumber = gamesEnum.Max(g:Game => g.TourNumber) + 1;
        }
        else
        {
            NewTourNumber = 1;
        }
    }

    IGetQueries.CreateInstance(context) // IGetQueries
        .TryGetPlayers(organiserId: tournament.OrganizerId, tournament.TournamentId,
                       out IEnumerable<Player>? players);

    _playersIds = GetPlayersIds(players);
    ConfigureTours();
}

```

Рисунок 4.40 – Исправление жеребьевки не первого тура

Повторный запуск сценария тестирования показал, что во время добавления новых игроков, это добавление не отслеживается алгоритмом, как предполагалось изначально. При unit-тестировании эта ошибка была обнаружена, но вероятно исправлена не полностью.

Для исправления этой ошибки в методе перенастройки ID игроков для алгоритма игроки напрямую перезагружаются с базы данных.

```

⌚ 1 usage  ✅ 20 tests OK *  ✎ new *
private void ReconfigureIdsIfPlayersChanged()
{
    // If players changed, reconfigure the tours.
    List<int> ids = GetPlayersIds(GetPlayers(_context, _tournament));

```

Рисунок 4.41 – Исправление обновления ID игроков для алгоритма

4.2.2 Второй сценарий

Далее создадим новый тест. Теперь добавим еще одного игрока так, чтобы их число стало нечетным и попробуем запустить алгоритм жеребьевки с добавлением списка пар игроков.

```
// Create mock pairs view model.  
var viewModel = new PairsGridViewModel();  
// Create mock TournamentsList view model.  
var tournamentsListViewModel = new TournamentsListViewModel();  
  
// Invoke login command.  
loginCommand.Execute(parameter: null);  
  
// Invoke open tournament command.  
var openTournamentCommand = new OpenTournamentCommand(tournamentsListViewModel);  
openTournamentCommand.Execute(tournament);  
  
// Add new player.  
IInsertQueries.CreateInstance(_context) // IInsertQueries  
    .TryAddPlayer(out Player? _, tournamentId: _tourId, organizerId: _orgId,  
        lastName: "Test" + DateTime.Now.Ticks, firstName: "Player" + DateTime.Now.Ticks);  
  
int oldTourNumber = viewModel.CurrentTour;  
  
// Execute start tour command.  
viewModel.StartNewTour.Execute(parameter: null);  
  
// Check result.  
Assert.AreEqual(expected: oldTourNumber + 1, actual: viewModel.CurrentTour);
```

Рисунок 4.42 – Тестирование второго сценария

Тест выявил ошибку: не предусмотрено добавление списка пар при нечетном количестве игроков.

Необходимо проверять ID игроков, и если там попадается ID равное -1, то необходимо добавить сущность игрока на время, пока количество игроков нечетно.

Исправление предоставлено на рисунке 4.43 – при появлении элемента с ID равным -1, создается игрок-заместитель недостающего.

```

// If id = -1 then it means that the player is absent.
// Then add a game with the result of 0:1 and mark as not played.
// Also add new dummy player to the database.
if (idPair.Item1 == -1)
{
    IInsertQueries.CreateInstance(PairsGridViewModel.PairsContext) // IInsertQueries
        .TryAddPlayer(out Player? dummyPlayer,
                      TournamentsListViewModel.SelectedTournament!.TournamentId,
                      organizerId: TournamentsListViewModel.SelectedTournament.OrganizerId,
                      lastName: "<Пусто>", firstName: "<Пусто>");
    IInsertQueries.CreateInstance(PairsGridViewModel.PairsContext) // IInsertQueries
        .TryAddGamePair(out Game game, whited: dummyPlayer!.PlayerId, blackId: idPair.Item2,
                        TournamentsListViewModel.SelectedTournament! // Tournament
                            .TournamentId,
                        TournamentsListViewModel.SelectedTournament.OrganizerId,
                        RoundRobin.NewTourNumber);

    game.WhitePoints = 0;
    game.BlackPoints = 1;
    game.IsPlayed = false;

    GameAddedEvent.OnGameAdded(sender: this, e: new GameAddedEventArgs(game!));
    continue;
}

if (idPair.Item2 == -1)
{
    IInsertQueries.CreateInstance(PairsGridViewModel.PairsContext) // IInsertQueries
        .TryAddPlayer(out Player? dummyPlayer,
                      TournamentsListViewModel.SelectedTournament!.TournamentId,
                      organizerId: TournamentsListViewModel.SelectedTournament.OrganizerId,
                      lastName: "<Пусто>", firstName: "<Пусто>");
    IInsertQueries.CreateInstance(PairsGridViewModel.PairsContext) // IInsertQueries

```

Рисунок 4.43 – Исправление добавление пары без одного игрока

Также теперь необходимо добавить проверку на то, чтобы при появлении нового игрока игрок-заместитель переставал быть активным.

```

public override void Execute(object? parameter)
{
    // Check if there are any dummy players and make them inactive if there are odd number of players.
    if (TournamentsListViewModel.SelectedTournament!.Players.Contains(_pairsGridViewModel.DummyPlayer)
        && TournamentsListViewModel.SelectedTournament!.Players.Count % 2 == 1)
    {
        _pairsGridViewModel.DummyPlayer!.IsActive = false;
        PairsGridViewModel.PairsContext.SaveChanges();
    }
}

```

Рисунок 4.44 – Отключение игрока-заместителя, если количество игроков стало нечетным

Для проверки отключения игрока-заместителя снова запустим второй тест – добавится новый игрок, и вместе их станет нечетное количество – в новом туре игрока-заместителя быть не должно.

4.3 Системное тестирование

В системном тестировании проверим, что все основные компоненты системы (регистрация, вход, создание турнира, заполнение участников, команд, групп, проведение жеребьевки) работают.

Сценарий тестирования:

1. Зарегистрировать пользователя.
2. Войти в аккаунт.
3. Создать турнир.
4. Открыть турнир.
5. Заполнить список участников.
6. Заполнить список команд.
7. Заполнить список групп.
8. Провести жеребьевку всего турнира.

В методе `SetUp` определим модели представления, с помощью которых будем осуществлять запуск команд (см. рисунок 4.45).

```
public class Tests
{
    private RegisterViewModel      _registerViewModel;
    private LoginViewModel        _loginViewModel;
    private TournamentsListViewModel _tournamentsListViewModel;
    private CreateTournamentViewModel _createTournamentViewModel;
    private PlayersViewModel       _playersViewModel;
    private AddPlayerViewModel    _addPlayerViewModel;
    private ManageGroupsViewModel _manageGroupsViewModel;
    private ManageTeamsViewModel  _manageTeamsViewModel;
    private PairsGridViewModel     _pairsGridViewModel;
    private ManageRatingsViewModel _manageRatingsViewModel;

    [SetUp]
    ◇
    public void Setup()
    {
        _pairsGridViewModel      = new PairsGridViewModel();
        _manageRatingsViewModel  = new ManageRatingsViewModel();
        _manageGroupsViewModel   = new ManageGroupsViewModel();
        _manageTeamsViewModel    = new ManageTeamsViewModel();
        _playersViewModel         = new PlayersViewModel();
        _addPlayerViewModel       = new AddPlayerViewModel();
        _tournamentsListViewModel= new TournamentsListViewModel();
        _createTournamentViewModel= new CreateTournamentViewModel();
        _registerViewModel        = new RegisterViewModel();
        _loginViewModel           = new LoginViewModel();
    }
}
```

Рисунок 4.45 – Инициализация моделей представлений

Далее определим сценарий тестирования. Зарегистрируем пользователя, войдем в систему, создадим турнир, откроем его, вставим 6 игроков, 2 команды, 3 группы, затем проведем жеребьевку 5 туров.

```
/// Testing scenario: ...
[Test]
◊
public void Test()
{
    string login = "petrov@mail.ru" + DateTime.Now.Ticks;
    RegisterNewUser(lastName: "Петров", firstName: "Петр", login, password: "123qwe");

    Login(login, password: "123qwe");

    CreateTournament(name: "Турнир по шахматам");

    OpenTournament(name: "Турнир по шахматам");

    InsertPlayers(amount: 6);

    InsertTeams(amount: 2);

    InsertGroups(amount: 3);

    ConductDraw(toursNumber: 5);
}
```

Рисунок 4.46 – Сценарий тестирования

Для регистрации пользователя в модели представления заполним данные о пользователе и вызовем команду регистрации. Если ошибок на этом этапе не возникло, пользователь будет зарегистрирован.

```
private void RegisterNewUser(string lastName, string firstName, string login, string password)
{
    _registerViewModel.LastName      = lastName;
    _registerViewModel.FirstName     = firstName;
    _registerViewModel.Email         = login;
    _registerViewModel.PasswordInit = password;
    _registerViewModel.PasswordConfirm = password;

    var registerCommand = new RegisterCommand(_registerViewModel);
    registerCommand.Execute(parameter: null);
}
```

Рисунок 4.47 – Регистрация пользователя

Таким же образом войдем в аккаунт этого пользователя.

```
private void Login(string login, string password)
{
    _loginViewModel.Login = login;
    _loginViewModel.Password = password;

    var loginCommand = new LoginCommand(_loginViewModel);
    loginCommand.Execute(parameter: null);
}
```

Рисунок 4.48 – Вход в аккаунт

Далее создадим турнир, для этого в модели представления создания турнира запишем название турнира и вызовем команду создания турнира.

```
private void CreateTournament(string name)
{
    _createTournamentViewModel.TournamentNameText = name;
    var createTournamentCommand = new CreateTournamentCommand(_createTournamentViewModel);
    createTournamentCommand.Execute(parameter: null);
}
```

Рисунок 4.49 – Создание турнира

Далее откроем этот турнир. Метод расширения «Single» позволяет убедиться, что такой турнир один, так как если их несколько, то он поднимает исключение. Таким образом, мы убедимся, что турнир создался в одном экземпляре.

В метод вызова команды открытия турнира передадим этот турнир.

```
private void OpenTournament(string name)
{
    var openTournamentCommand = new OpenTournamentCommand(_tournamentsListViewModel);
    Tournament tournament = _tournamentsListViewModel // TournamentsListViewModel
        .TournamentsCollection // ObservableCollection<Tournament>?
            .Single(t:Tournament => t.TournamentName == name);
    openTournamentCommand.Execute(tournament);
}
```

Рисунок 4.50 – Открытие турнира

Далее добавим игроков в турнир. В цикле определим имена и вызовем команду добавления игрока. Если при этом не возникло исключений, сценарий продолжается.

```
private void InsertPlayers(int amount)
{
    for (var i = 0; i < amount; i++)
    {
        _addPlayerViewModel.PlayerLastName = $"Иванов{i}";
        _addPlayerViewModel.PlayerFirstName = $"Иван{i}";

        var completeAddPlayerCommand = new CompleteAddPlayerCommand(_addPlayerViewModel);
        completeAddPlayerCommand.Execute(parameter: null);
    }
}
```

Рисунок 4.51 – Добавление игроков.

Далее добавим команды и группы. Для команд достаточно определить ее название, для группы также определим краткий идентификатор.

```
private void InsertGroups(int amount)
{
    for (var i = 0; i < amount; i++)
    {
        _manageGroupsViewModel.GroupName      = $"Группа{i}";
        _manageGroupsViewModel.GroupIdentifier = $"Г{i}";

        var completeAddGroupCommand = new CompleteAddGroupCommand(_manageGroupsViewModel);
        completeAddGroupCommand.Execute(parameter: null);
    }
}

◆ 1 usage
private void InsertTeams(int amount)
{
    for (var i = 0; i < amount; i++)
    {
        _manageTeamsViewModel.TeamName = $"Команда{i}";

        var completeAddTeamCommand = new CompleteAddTeamCommand(_manageTeamsViewModel);
        completeAddTeamCommand.Execute(parameter: null);
    }
}
```

Рисунок 4.52 – Добавление команд и групп

Далее проведем жеребьевку турнира. В команде начала нового тура вызовется алгоритм жеребьевки для выбранного тура, после чего полученные пары добавятся в базу данных. Далее проверим, что текущий тур сменился, а также, что пар в туре 3, так как в турнире определено 6 игроков.

```
private void ConductDraw(int toursNumber)
{
    for (var i = 0; i < toursNumber; i++)
    {
        _pairsGridViewModel.CurrentTour = i;
        _pairsGridViewModel.StartNewTour.Execute(parameter: null);
        Assert.AreEqual(expected: i + 1, actual: _pairsGridViewModel.CurrentTour);
        Assert.AreEqual(expected: 3, actual: _pairsGridViewModel.Pairs.Count);
    }
}
```

Рисунок 4.53 – Проведение жеребьевки

Сценарий не выявил ошибок, жеребьевка была проведена.

Заключение

В результате выполнения курсовой работы по созданию приложения для организации шахматных турниров, были выполнены следующие задачи:

1. Проанализирована предметная область, были выявлены проблемные места текущих бизнес-процессов.
2. Проанализированы существующие решения, выявлены их недостатки и положительные стороны.
3. Были разработаны функциональные и нефункциональные требования к системе.
4. Спроектирована и реализована информационная система с использованием следующих средств: IDE Microsoft Visual Studio, .NET 6, C# 10, СУБД PostgreSQL.
5. Проведено unit-тестирование системы, а также интеграционное и системное.
6. Подготовлена программная документация в соответствии с ГОСТ 19 (техническое задание, руководство пользователя).

В ходе разработки приложения активно использовался архитектурный паттерн MVVM [5], позволивший организовать взаимодействие между классами. Также программа основывается на принципах объектно-ориентированного программирования. Для организации слоя доступа к данным, реализованного с помощью ORM Entity Framework [6], использовался паттерн «Репозиторий» [7], благодаря которому организовывался доступ к сущностям базы данных.

Исходный код программы открыт и расположен на хостинге программных продуктов GitHub (см. ПРИЛОЖЕНИЕ 3).

Разработанное приложение отвечает большинству заявленных функциональных и нефункциональных требований и готова к эксплуатированию, однако требует доработки и доразвития функционала.

Библиографический список

- Википедия [Электронный ресурс] // Шахматы: [сайт]. [2022].
URL: <https://ru.wikipedia.org/wiki/Шахматы> (дата обращения: 22.11.2022).
- Microsoft [Электронный ресурс] // What is .NET?: [сайт]. [2023]. URL: <https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet> (дата обращения: 15.01.2023).
- Chess.sainfo.ru [Электронный ресурс] // Как провести круговой турнир не имея таблиц очередности игры: [сайт]. [2023] URL: <http://chess.sainfo.ru/tabl/tablei.html> (дата обращения: 20.02.2023).
- www.sportzone.ru [Электронный ресурс] // Шахматы - официальные правила: [сайт]. [2023]
URL: <https://www.sportzone.ru/sport/rules.html?sport=chess&chapter=04> (дата обращения: 20.01.2023).
- www.rrweb.org [Электронный ресурс] // C.04 FIDE Swiss rules: [документ]. [2023]
URL: <http://www.rrweb.org/javafo/C04.pdf> (дата обращения: 20.01.2023).
- ratings.ruchess.ru [Электронный ресурс] // Турниры: [сайт]. [2023]
URL: <https://ratings.ruchess.ru/tournaments> (дата обращения: 16.03.2023).
- learn.microsoft.com [Электронный ресурс] // Patterns - WPF Apps With The Model-View-ViewModel Design Pattern: [сайт]. [2022] URL: <https://learn.microsoft.com/en-us/archive/msdn-magazine/2009/february/patterns-wpf-apps-with-the-model-view-viewmodel-design-pattern> (дата обращения: 12.12.2022).
- metanit.com [Электронный ресурс] // Введение в Entity Framework: [сайт]. [2023]
URL: <https://metanit.com/sharp/entityframework/> (дата обращения: 12.01.2023).
- learn.microsoft.com [Электронный ресурс] // Design the infrastructure persistence layer: [сайт]. [2023] URL: <https://learn.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design>
(дата обращения: 12.01.2023).

ПРИЛОЖЕНИЕ А

Терминология

Шахматы — настольная логическая игра с шахматными фигурами на 64-клеточной доске, сочетающая в себе элементы искусства (в том числе в части шахматной композиции), науки и спорта. В шахматы обычно играют два игрока (именуемые шахматистами) друг против друга. [1]

Швейцарская система жеребьевки — система проведения спортивных турниров, распределяющая игроков по силе. [2]

Круговая система жеребьевки — система розыгрыша, при которой каждый участник турнира играет с каждым в ходе тура или раунда. [3]

Коэффициенты жеребьевки — показатель, помогающий определению мест в турнире в таких играх, как шахматы, шашки, сёги, го, рэндзю, новус и т.д. среди участников, набравших равное количество очков. [4]

Паттерн MVVM — архитектурный шаблон в компьютерном программном обеспечении, облегчающий отделение пользовательского интерфейса от бизнес-логики. [5]

Entity Framework — объектно-ориентированная технология доступа к данным, является object-relational mapping (ORM) решением для .NET от Microsoft. [6]

Паттерн «Репозиторий» — шаблон конструктора Domain-Driven, предназначенный для соблюдения сохраняемости данных за пределами модели предметной области системы. [7]

ПРИЛОЖЕНИЕ Б

Техническое задание

**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет социально-экономических и компьютерных наук
Образовательная программа бакалавриата «Программная инженерия»

СОГЛАСОВАНО
Руководитель проекта,
доцент кафедры информационных
технологий в бизнесе НИУ ВШЭ – Пермь
канд. тех. наук

УТВЕРЖДАЮ
Академический руководитель
образовательной программы
«Программная инженерия»

_____ О. Л. Викентьева
«___» _____ 2023 г.

_____ В. В. Ланин
«___» _____ 2023 г.

РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ЖЕРЕБЬЁВКИ ШАХМАТНЫХ ТУРНИРОВ

Техническое задание

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.05.03-01 ТЗ 01-1-ЛУ

Инв. № подп.	Подп. и дата	Взим. инв. №	Инв. №	Подп. и дата
RU.17701729.05.03-01 ТЗ 01-1				

Исполнитель:
студент группы ПИ-21-1
А. Ю. Некрасов
«___» _____ 2023 г.

Пермь, 2023

УТВЕРЖДЕН
RU.17701729.05.03-01 Т3 01-1-ЛУ

**РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ЖЕРЕБЬЁВКИ
ШАХМАТНЫХ ТУРНИРОВ**

Техническое задание

RU.17701729.05.03-01 Т3 01-1

Листов 17

Инв. № подп.	Подп. и дата	Взаем. инв. №	Инв. № дубл.	Подп. и дата
RU.17701729.05.03-01 T3 01-1				

Пермь, 2023

Оглавление

1 Введение.....	3
1.1 Наименование программы	3
1.2 Краткая характеристика области применения программы.....	3
2 Основание для разработки	4
2.1 Основание для проведения разработки	4
2.2 Наименование и условное обозначение темы разработки.....	4
3 Назначение разработки.....	5
3.1 Функциональное назначение программы.....	5
3.2 Эксплуатационное назначение программы.....	5
4 Требования к программе	6
4.1 Требования к функциональным характеристикам	6
4.2 Требования к надёжности	8
4.3 Условия эксплуатации.....	8
4.4 Требования к составу и параметрам технических средств	8
4.5 Требования к информационной и программной совместимости.....	8
4.6 Требования к маркировке и упаковке	9
5 Требования к программной документации	10
5.1 Состав программной документации	10
5.2 Специальные требования к программной документации.....	10
6 Технико-экономические показатели	12
7 Стадии и этапы разработки	13
8 Порядок контроля и приёмки.....	14
Приложения.....	15
Терминология.....	15
Список литературы.....	160
Заявка на проект	17

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.03-01 Т3 01-1				
Инв. № подп.	Подп. и дата	Взаим. инв. №	Инв. № дубл.	Подп. и дата

1 Введение

1.1 Наименование программы

Наименование – «Приложение для жеребьёвки шахматных турниров».

1.2 Краткая характеристика области применения программы

Программа предназначена для организации процесса проведения очных шахматных турниров от стадии регистрации участников до подведения итогов турнира. При этом данная программа может быть использована не только для жеребьевки турниров по шахматам, но и шашкам, и другим играм на двоих.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.03-01 Т3 01-1				
Инв. № подп.	Подп. и дата	Взаим. инв. №	Инв. № дубл.	Подп. и дата

2 Основание для разработки

2.1 Основание для проведения разработки

Заявка на выполнение курсовой работы (см. ПРИЛОЖЕНИЕ В).

2.2 Наименование и условное обозначение темы разработки

Наименование темы разработки – «Разработка приложения для жеребьёвки шахматных турниров».

Иzm.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.03-01 Т3 01-1				
Инв. № подп.	Подп. и дата	Взаим. инв. №	Инв. № дубл.	Подп. и дата

3 Назначение разработки

3.1 Функциональное назначение программы

Программа предоставляет возможность пользователю создания в системе турнира, ввода игроков и их команд, осуществления жеребьевки для каждого игрового тура, подсчет очков и других коэффициентов, подведение итогов в виде рейтинг-листов, вывод этих листов на печать или экспорт в виде документов.

3.2 Эксплуатационное назначение программы

Программа позволяет упростить и автоматизировать процесс проведения очных шахматных турниров.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.03-01 Т3 01-1				
Инв. № подп.	Подп. и дата	Взаим. инв. №	Инв. № дубл.	Подп. и дата

4 Требования к программе

4.1 Требования к функциональным характеристикам

Программа состоит из клиентской и серверной частей. Взаимодействие с пользователем происходит через приложение.

Серверная часть программы представляет собой сервер с базой данных, хранящей информацию о пользователях, турнирах, ими созданных.

Клиентская часть программы представляет собой настольное приложение, которое представляет собой интерфейс между базой данных и пользователем. Также в клиентской части программы происходит сам процесс жеребьевки участников турнира с помощью алгоритмов.

Программа должна отвечать данным требованиям:

- программа должна предоставлять пользователю возможность просмотра списков турниров, принадлежащих ему, а также возможность их просмотра, редактирования, создания, удаления;
- программа должна предоставлять пользователю возможность ввода участников турнира в турнирные списки, их редактирование, удаление;
- программа должна предоставлять пользователю выбор системы проведения турнира (круговая, швейцарская, или другие), вида турнира (командный, личный, лично-командный), коэффициентов ранжирования игроков;

Иzm.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.03-01 Т3 01-1				
Инв. № подп.	Подп. и дата	Взаим. инв. №	Инв. № дубл.	Подп. и дата

- программа должна предоставлять пользователю возможность осуществления групповых турниров, при этом давая пользователю выбор в позволнении игрокам из разных групп играть вместе или раздельно;
- программа должна предоставлять пользователю возможно редактирования результатов предыдущих туров в случае обнаружения опечатки или другой ошибки без необходимости переигрывания других туров;
- программа должна автоматически осуществлять подсчёт очков игроков, коэффициентов, а также следить за тем, чтобы игроки не играли друг с другом более одного раза;
- программа должна предоставить пользователю возможность ввода команд игроков, если турнир командный или лично-командный;
- программа должна поддерживать экспорт данных в виде различных типов документов (.xls, .doc);
- программа должна предоставлять пользователю возможность импорта данных (список участников) из офисных документов (.csv, .xls).

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.03-01 Т3 01-1				
Инв. № подп.	Подп. и дата	Взаим. инв. №	Инв. № дубл.	Подп. и дата

4.2 Требования к надёжности

Программа должна безотказно работать (не требовать перезагрузок или других технических работ). Программа должна предоставлять пользователю доступ к базе турниров, находящихся на сервере, как и тем турнирам, находящимся в локальной базе данных, тем самым, программа должна безотказно работать как при подключеннном интернет-соединении, так и без него.

4.3 Условия эксплуатации

4.3.1 Климатические условия эксплуатации

Требований к климатическим условиям эксплуатации не предъявляется.

4.3.2 Требования к видам обслуживания

Обслуживание не требуется.

4.3.3 Требования к численности и квалификации персонала

Техническая поддержка производится силами разработчика. Для пользования программой дополнительное обучение не требуется.

4.4 Требования к составу и параметрам технических средств

Устройство, поддерживающее запуск приложения на платформе .NET 6, операционная система Windows 10, рекомендуемая версия операционной системы: 22H2.

4.5 Требования к информационной и программной совместимости

4.5.1 Требования к исходным кодам и языкам программирования

Исходные коды программы должны быть написаны на языке C# и выложены на сервис для хостинга IT проектов GitHub.

4.5.2 Требования к программным средствам, используемым программой

Программа должна работать на персональном компьютере с установленной операционной системой Windows 10 22H2, платформой .NET 6.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.03-01 Т3 01-1				
Инв. № подп.	Подп. и дата	Взаим. инв. №	Инв. № дубл.	Подп. и дата

4.6 Требования к маркировке и упаковке

Программа распространяется посредством прямой ссылки на установочный пакет.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.03-01 Т3 01-1				
Инв. № подп.	Подп. и дата	Взаим. инв. №	Инв. № дубл.	Подп. и дата

5 Требования к программной документации

5.1 Состав программной документации

Состав программной документации:

- программа для жеребьёвки шахматных турниров. Техническое задание (ГОСТ 19.201-78);
- программа для жеребьёвки шахматных турниров. Руководство оператора (ГОСТ 19.505-79);
- программа для жеребьёвки шахматных турниров. Текст программы. (ГОСТ 19.401-78).

5.2 Специальные требования к программной документации

Документы к программе должны быть выполнены в соответствии с ГОСТ 19.106–78 и ГОСТАми к каждому виду документа (см. п. 5.1).

Пояснительная записка должна быть загружена в систему Антиплагиат через LMS «НИУ ВШЭ». Лист, подтверждающий загрузку пояснительной записи, сдаётся в учебный офис вместе со всеми материалами не позже, чем за день до защиты курсовой работы.

Техническое задание и пояснительная записка, титульные листы других документов должны быть напечатаны, подписаны академическим руководителем образовательной программы 09.03.04 «Программная инженерия», руководителем разработки и исполнителем перед сдачей курсовой работы в учебный офис не позже одного дня до защиты.

Документация и программа также сдаётся в электронном виде в формате .pdf или .docx в архиве формата .zip или .rar.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.03-01 Т3 01-1				
Инв. № подп.	Подп. и дата	Взаим. инв. №	Инв. № дубл.	Подп. и дата

За один день до защиты комиссии все материалы курсового проекта:

- техническая документация;
- программный проект;
- исполняемый файл;
- отзыв руководителя

должны быть загружены одним или несколькими архивами в проект дисциплины «Курсовая работа 2023–2023» в личном кабинете в информационной образовательной среде «LMS» («Learning Management System») НИУ ВШЭ.

Иzm.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.03-01 ТЗ 01-1				
Инв. № подп.	Подп. и дата	Взаим. инв. №	Инв. № дубл.	Подп. и дата

6 Технико-экономические показатели

В рамках данной работы расчёт экономической эффективности не предусмотрено.

Использование разрабатываемого инструмента предназначено для упрощения процесса организации очных турниров по шахматам.

Предполагаемая потребность обуславливается тем фактом, что на данный момент трудно найти бесплатный инструмент, обеспечивающий выполнение всех описанных функциональных требований.

Существующие аналоги не дают возможности работы с исходным кодом и тонкой настройки функциональных возможностей.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.03-01 Т3 01-1				
Инв. № подп.	Подп. и дата	Взаим. инв. №	Инв. № дубл.	Подп. и дата

7 Стадии и этапы разработки

Количество и длительность проекта определяется итерационным характером модели жизненного цикла продукта и линейным характером жизненного цикла проекта.

С точки зрения управления проектом выделяются стадии:

1. Инициация.
2. Планирование.
3. Исполнение.
4. Завершение.

С точки зрения управления жизненным циклом продукта проект состоит из трёх итераций, каждая из которых состоит из пяти этапов:

1. Инициализация.
2. Проектирование.
3. Реализация.
4. Тестирование.
5. Развёртывание.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.03-01 Т3 01-1				
Инв. № подп.	Подп. и дата	Взаим. инв. №	Инв. № дубл.	Подп. и дата

8 Порядок контроля и приёмки

Контроль и приёмка разработки осуществляются в соответствии с требованиями к программе, требованиями к программной документации и положениями, описанными в отчёте в главе 4 «Тестирование».

Иzm.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.03-01 Т3 01-1				
Инв. № подп.	Подп. и дата	Взаим. инв. №	Инв. № дубл.	Подп. и дата

Приложения

Терминология

Приложение А

Шахматы — настольная логическая игра с шахматными фигурами на 64-клеточной доске, сочетающая в себе элементы искусства (в том числе в части шахматной композиции), науки и спорта. В шахматы обычно играют два игрока (именуемые шахматистами) друг против друга. [1]

.NET 6 — бесплатная кроссплатформенная платформа с открытым исходным кодом для разработки приложений различных типов.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.03-01 Т3 01-1				
Инв. № подп.	Подп. и дата	Взаим. инв. №	Инв. № дубл.	Подп. и дата

Список литературы

Приложение Б

- Википедия [Электронный ресурс] // Шахматы: [сайт]. [2022]. URL:
<https://ru.wikipedia.org/wiki/Шахматы> (дата обращения: 22.11.2022).
- Microsoft [Электронный ресурс] // What is .NET?: [сайт]. [2023]. URL:
<https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet> (дата обращения: 15.01.2023).
- Википедия [Электронный ресурс] // Шахматы: [сайт]. [2022]. URL:
https://ru.wikipedia.org/wiki/Круговая_система (дата обращения: 22.12.2022).

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.03-01 Т3 01-1				
Инв. № подп.	Подп. и дата	Взаим. инв. №	Инв. № дубл.	Подп. и дата

Заявка на проект

Приложение В

СОГЛАСОВАНО

И. О. академического руководителя
образовательной программы
«Программная инженерия»

_____ О.Л. Викентьева
«_____» 20____ г.

ЗАЯВКА НА ПРОЕКТ

Название проекта: _____

Инициатор проекта: _____

Тип проекта: _____

Планируемые результаты проекта

Образовательные: _____

Проектные: _____

Сроки и условия реализации проекта

Плановые сроки начала: _____

Плановые сроки окончания: _____

Содержание проекта: _____

Форма представления итогового результата: _____

Руководитель проекта: _____

(должность, Фамилия, Имя, Отчество)

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.03-01 ТЗ 01-1				
Инв. № подп.	Подп. и дата	Взаим. инв. №	Инв. № дубл.	Подп. и дата

ПРИЛОЖЕНИЕ В **Руководство оператора**

**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет социально-экономических и компьютерных наук
Образовательная программа бакалавриата «Программная инженерия»

СОГЛАСОВАНО
Руководитель проекта,
доцент кафедры информационных
технологий в бизнесе НИУ ВШЭ – Пермь
канд. тех. наук

_____ О. Л. Викентьева
«__» 2023 г.

УТВЕРЖДАЮ
Академический руководитель
образовательной программы
«Программная инженерия»,
доцент, канд. тех. наук

_____ В. В. Ланин
«__» 2023 г.

Инв. № подп.	Подп. и дата	Взаем. инв. №	Инв. № дубл.	Подп. и дата
RU.17701729.05.03-0 PO 01-1				

РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ЖЕРЕБЬЁВКИ ШАХМАТНЫХ ТУРНИРОВ **Руководство оператора** **ЛИСТ УТВЕРЖДЕНИЯ** **RU.17701729.05.03-01 РО 01-1-ЛУ**

Исполнитель:
студент группы _____
А. Ю. Некрасов
«__» 2023 г.

Пермь, 2023

УТВЕРЖДЕН
RU.17701729.12151-01 РО 01-1-ЛУ

Инв. № подп.	Подп. и дата	Взаем. инв. №	Инв. № дубл.	Подп. и дата
RU.17701729.05.03-0 РО 01-1				

**РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ЖЕРЕБЬЁВКИ
ШАХМАТНЫХ ТУРНИРОВ**
Руководство оператора
RU.17701729.12151-01 РО 01-1
Листов 10

Пермь, 2023

Оглавление

1 Назначение программы	3
2 Условия выполнения программы	4
3 Выполнение программы	5
3.1 Установка программы	5
3.2 Запуск программы.....	5
3.3 Выполнение программы.....	5
3.4 Завершение программы.....	7
4 Сообщения оператору.....	8

Иzm.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.03-01 РО 01-1				
Инв. № подп.	Подп. и дата	Взаим. инв. №	Инв. № дубл.	Подп. и дата

1 Назначение программы

Программа предназначена для проведения жеребьевки шахматных турниров.

Программа обеспечивает хранение списка турниров, игроков, команд, групп, определенных для турнира. Доступ к данным осуществляется напрямую посредством взаимодействия с элементами управления программы. Дополнительных средств для заполнения данных не требуется.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.03-01 РО 01-1				
Инв. № подп.	Подп. и дата	Взаим. инв. №	Инв. № дубл.	Подп. и дата

2 Условия выполнения программы

Минимальный состав аппаратурных и программных средств, необходимый для запуска и использования программы:

1. Персональный компьютер.
2. Операционные системы Windows 7, Windows 8, Windows 10, Windows 11.
3. Разрядность процессора: x32, x86.
4. Разрешение экрана 1000×700 пикселей.
5. Интернет-доступ.
6. Объем дискового пространства: 50 МБ.
7. Оперативная память: 100 МБ.
8. Поддержка запуска приложений .NET 6.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.03-01 РО 01-1				
Инв. № подп.	Подп. и дата	Взаим. инв. №	Инв. № дубл.	Подп. и дата

3 Выполнение программы

3.1 Установка программы

Программа устанавливается на операционную систему пользователя при выполнении установки с помощью заранее подготовленного установочного файла. Пользователь может выбрать каталог установки, также запустить ли программу сразу после завершения установки.

3.2 Запуск программы

Программа запускается посредством нажатия на её ярлык. В результате чего перед пользователем возникает главное окно приложения. Далее пользователь самостоятельно решает, что делать.

3.3 Выполнение программы

Далее приведены возможные варианты выполнения программы.

3.3.1 Авторизация

При запуске программы открывается окно авторизации. Пользователь также может выбрать регистрацию. Нажав на кнопку «Зарегистрироваться», откроется окно регистрации, где пользователь может ввести свои данные и, нажав кнопку «Зарегистрироваться», откроется окно авторизации, где пользователь может ввести свои данные для входа.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.03-01 РО 01-1				
Инв. № подп.	Подп. и дата	Взаим. инв. №	Инв. № дубл.	Подп. и дата

3.3.2 Основное окно программы

После успешного входа откроется главное окно программы, в котором будут доступны следующие вкладки:

1. Профиль пользователя – страница с данными пользователя.
2. Список турниров – страница со списком турниров пользователя.
3. Дерево турниров – древовидное представление списка командных турниров, позволяющее редактировать данные сразу нескольких турниров.

Также в главном окне программы доступны следующие пункты меню:

1. Создать турнир – при запуске открывает окно с заполнением параметров нового турнира.
2. Экспорт – позволяет выполнить экспорт списка участников, списка пар и рейтинг-листа в файл.
3. Справка – содержит вложенный пункт меню «О программе», позволяющий открыть окно с информацией о программе

3.3.2.1. Вкладка турнира

При открытии турнира из списка, откроется вкладка турнира, содержащая другие вкладки:

1. Список участников – страница со списком участников и кнопкой добавления нового участника.
2. Список команд – страница с древовидным списком команд, где у каждой команды есть вложенный список игроков, состоящих в этой команде.
3. Список групп – страница с древовидным списком групп, где у каждой группы есть вложенный список игроков, состоящих в этой группе.
4. Список пар – страница со списком пар. Также содержит кнопки навигации по турам (просмотр предыдущего, следующего), кнопка начала нового тура (запуск алгоритма жеребьевки).

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.03-01 РО 01-1				
Инв. № подп.	Подп. и дата	Взаим. инв. №	Инв. № дубл.	Подп. и дата

5. Рейтинг-лист – страница с таблицей игроков, отсортированных по их очкам им коэффициентам.

4.3.1 Добавление данных

Добавление турнира осуществляется посредством нажатия соответствующей кнопки главного меню, после чего открывается окно с вводом параметров турнира. После проверки параметров, турнир добавится в список турниров пользователя.

Добавление данных в список участников осуществляется посредством нажатия соответствующей кнопки, находящейся на странице списка участников. При ее нажатии, в таблице участников появится пустое поле, которое пользователь может отредактировать.

Добавление групп и команд осуществляется посредством нажатия соответствующей кнопки на странице списка групп или команд, после чего открывается окно с заполнением необходимых параметров. При нажатии кнопки сохранения, команда или группа появятся в списке.

4.3.2 Редактирование данных

Редактирование данных турнира осуществляется посредством нажатия соответствующей кнопки в списке турниров. При ее нажатии открывается окно с параметрами турнира, которые можно отредактировать.

Редактирование игроков происходит посредством редактирования данных в таблице игроков.

Редактирование данных о группе и команде происходит посредством нажатия соответствующей кнопки в списке группы или команды.

Редактирование результатов игр происходит посредством выбора результата из выпадающего списка в строке пары игроков.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.05.03-01 РО 01-1				
Инв. № подп.	Подп. и дата	Взаим. инв. №	Инв. № дубл.	Подп. и дата

3.4 Завершение программы

Программа завершается пользователем посредством закрытия главного окна.
Дополнительных действий для сохранения данных не требуется.

4 Сообщения оператору.

В процессе работы с программой пользователь получает обратную реакцию, которую можно разделить на позитивную (команда выполнена успешно), негативную (возникла ошибка) и вопрос (подтверждение действия).

Далее приведен список возможных позитивных реакций:

1. «Вы успешно зарегистрировались!» – возникает при успешной регистрации пользователя.
2. «Турнир успешно создан!» – возникает при успешном завершении создания турнира.
3. «Игрок успешно удален!» – возникает при успешном завершении удаления игрока.
4. «Турнир успешно отредактирован!» – возникает при успешном завершении редактирования турнира.
5. «Команда успешно добавлена!» – возникает при успешном завершении добавлении команды.
6. «Изменения в команде успешно сохранены!» – возникает при успешном завершении редактирования команды.
7. «Группа успешно добавлена!» – возникает при успешном завершении добавлении группы.
8. «Изменения в группе успешно сохранены!» – возникает при успешном завершении редактирования группы.

Негативные реакции:

1. «Недостаточно игроков для проведения жеребьевки!» – сообщение при попытке запуска жеребьевки, когда недостаточное количество игроков.
2. «Возможно игрок с такими параметрами уже существует» – сообщение при попытке отредактировать параметры игрока так, что они совпадают с параметрами другого игрока.
3. «Пользователь не найден!» – сообщение при попытке получения списка турниров несуществующего пользователя.
4. «Не удалось сохранить изменения. Команда не найдена.» – сообщение при попытке изменить несуществующую команду.

5. «Нельзя удалить игрока, участвующего в игре!» – сообщение при попытке удаления игрока, который записан в списке пар.
6. «Возможно пользователь с таким email уже существует» – сообщение при попытке регистрации пользователя с повторяющимся email.
7. «Пароли не совпадают!» – сообщение при попытке зарегистрироваться, указав не совпадающие пароли.

Вопросительные реакции:

1. «Вы действительно хотите сохранить изменения в команде?» – подтверждение редактирования команды.
2. «Вы действительно хотите удалить группу?» – подтверждение удаления группы.

ПРИЛОЖЕНИЕ Г

Требования к данным

Таблица Г1 – Требования к данным

Имя атрибута	Тип данных	Значение по умолчанию	Формат ввода (маска)	Ограничение на значения атрибута	Комментарий
ID пользователя;	Целое	Нет	Нет	Не меньше 1, уникальный, обязательный, инкрементируемый	Не виден пользователю
Электронная почта;	Строка	Нет	Нет	Обязательный, уникальный, соответствует шаблону электронной почты	Проверяется регулярным выражением, необходима при входе в систему
Фамилия пользователя;	Строка	Нет	Нет	Обязательный, максимальная длина – 255 символов	
Имя пользователя;	Строка	Нет	Нет	Обязательный, максимальная длина – 255 символов	
Отчество пользователя;	Строка	–	Нет	Необязательный, максимальная длина – 255 символов	Если у пользователя нет отчества – вместо него ставится прочерк
Хеш пароля;	Строка	Нет	Нет	Обязательный, минимальная длина – 8 символов, максимальная – 50	Необходим для входа в систему
Дата регистрации	Дата и время	Текущая дата и время	Нет	Обязательный, не изменяемый	Системная информация о пользователе

Имя атрибута	Тип данных	Значение по умолчанию	Формат ввода (маска)	Ограничение на значения атрибута	Комментарий
Лимит (максимальное число) турниров	Число	50	Нет	Не меньше 5 и не больше 100, обязательный	Для пользователей с разным уровнем доступа — разное
ID турнира;	Целое	Нет	Нет	Не меньше 1, уникальный, обязательный, инкрементируемый	Не виден пользователю
Название организации	Строка	Нет	Нет	Необязательный, максимальная длина – 255 символов	Название организации, от имени которой проводится турнир
Название турнира;	Строка	Нет	Нет	Обязательный, уникальный, максимальная длина – 255 символов	Название должно быть уникальным, так как в данной предметной области не может существовать турниров с одинаковыми названиями
Место проведения турнира	Строка	Нет	Нет	Необязательный, максимальная длина – 255 символов	
Дата и время начала;	Дата и время	Текущая дата и время	—. —.—	Обязательный, не может быть ранее текущей системной даты и времени	Записывается с помощью специальной формы выбора даты и времени
Длительность	Целое	0	Нет	Не меньше 0 и не больше 10000	Длительность турнира в часах
ID системы турнира;	Целое	Нет	Нет	Не меньше 0 и не больше 1, обязательный	1 – Круговая система 2 – Швейцарская система
Название системы турнира	Строка	Нет	Нет	Обязательный	Задаётся системой и недоступно для изменения пользователями

Имя атрибута	Тип данных	Значение по умолчанию	Формат ввода (маска)	Ограничение на значения атрибута	Комментарий
ID вида турнира;	Целое	Нет	Нет	Не меньше 0 и не больше 2, обязательный	1 – Личный 2 – Командный 3 – Лично-командный
Название вида турнира	Строка	Нет	Нет	Обязательный	Задаётся системой и недоступно для изменения пользователями
Количество туров	Целое	7	Нет	Не меньше 4 и не больше 30, обязательный, разрешено редактировать до начала первого тура.	В круговой системе задаётся автоматически в зависимости от количества участников.
Дата и время создания;	Дата и время	Текущая дата и время	Нет	Обязательный, не редактируемый	Определяется автоматически относительно системного времени организатора
Дата и время последнего изменения;	Дата и время	Текущая дата и время	Нет	Обязательный, не редактируемый	Определяется автоматически относительно системного времени организатора
Максимальное число игроков в команде	Целое	5	Нет	Не меньше 2 и не больше 30	Определяет максимальное число игроков в команде, для которых будут учитываться очки, вкладываемые в команду, добавление игроков сверх этого числа разрешается.
Могут ли игроки из разных групп играть вместе	Логическое	Да	Нет	Обязательный, разрешено редактировать до начала первого тура.	Влияет на систему проведения турнира: если игрокам из разных групп разрешено играть вместе, то жеребьёвка для всех игроков будет одна, если игрокам из разных групп запрещено играть вместе, то жеребьёвка для каждой группы будет отдельная.

Имя атрибута	Тип данных	Значение по умолчанию	Формат ввода (маска)	Ограничение на значения атрибута	Комментарий
ID игрока;	Целое	Нет	Нет	Не меньше 1, обязательный, уникальный, инкрементируемый,	Не виден пользователю
Фамилия игрока;	Строка	Нет	Нет	Обязательный, максимальная длина – 255 символов	
Имя игрока;	Строка	Нет	Нет	Обязательный, максимальная длина – 255 символов	
Пол;	Строка	Нет	#	Необязательный, 1 символ	Мужской или женский
Атрибут игрока	Строка	Нет	Нет	Необязательный, максимальная длина – 3 символа.	Может использоваться для выделения особых игроков.
Год рождения игрока;	Целое	Нет	Нет	Необязательный, минимум – 1900, максимум – текущий системный год	Может помочь организатору распределить атрибуты игроков
Статус игрока (активен ли игрок)	Логический	Да	Нет	Обязательный.	Если игрок активен, то включается в жеребьевку, если неактивен – исключается из жеребьевки, но при этом остается в турнире и имеет право продолжить играть, в таком случае, его снова необходимо сделать активным.
Количество очков	Дробное	0	Нет	Обязательный, не редактируемый, обновляется по триггеру.	Высчитывается триггером после каждой записи результата для игрока с подсчетом суммы всех результатов встреч игрока.

Имя атрибута	Тип данных	Значение по умолчанию	Формат ввода (маска)	Ограничение на значения атрибута	Комментарий
Сумма коэффициентов	Дробное	0	Нет	Обязательный, не редактируемый, обновляется по триггеру.	Высчитывается триггером после каждой записи результата для игрока с подсчетом коэффициента Бухгольца или другого коэффициента.
Количество выигрышей	Целое	0	Нет	Обязательный, не редактируемый, обновляется по триггеру.	Высчитывается триггером после каждой записи результата для игрока с подсчетом суммы всех выигрышней игрока.
Количество проигрышней	Целое	0	Нет	Обязательный, не редактируемый, обновляется по триггеру.	Высчитывается триггером после каждой записи результата для игрока с подсчетом суммы всех проигрышней игрока.
Количество ничей	Целое	0	Нет	Обязательный, не редактируемый, обновляется по триггеру.	Высчитывается триггером после каждой записи результата для игрока с подсчетом суммы всех ничей игрока.
Номер доски	Целое	1	Нет	Не меньше 1 и меньше ли равно количеству игроков в команде, обязательен при командном турнире.	Позволяет ранжировать игроков внутри команды по доскам. Необходимо при строгом командном турнире, когда команды встречаются с командой и игроки играют по доскам.

Имя атрибута	Тип данных	Значение по умолчанию	Формат ввода (маска)	Ограничение на значения атрибута	Комментарий
ID группы	Целое	Нет	Нет	Обязательный, инкрементируемый, не редактируемый.	ID группы, к которой относится игрок, при командном турнире.
Идентификатор группы	Строка	1	Нет	Обязателен, максимальная длина – 4 символа.	Используется для краткого наименования группы.
Полное название группы	Строка	Без группы	Нет	Обязателен, максимальная длина – 255 символов.	Используется для полного наименования группы.
Состоялась ли игра	Логическое	Нет	Нет	Обязательный, определяется относительно результата встречи, предоставленного пользователем	Не виден пользователю
Номер тура;	Целое	0	Нет	Обязательный, инкрементируемый, максимальное значение соответствует максимальному заданному	Инкрементируется при каждом составлении новых пар и переходе к следующему турниру, до начала турнира равен 0
ID игрока белых;	Целое	Нет	Нет	Обязательный	Вместе образуют уникальную пару, так как в данной предметной области игроки играют с одним игроком максимум 1 раз
ID игрока чёрных;	Целое	Нет	Нет	Обязательный	
Очки белым	Дробное	Нет	Нет	Обязательный, либо 0, либо 1	Возможные значения: 0-0 (поражение обоим) 1-0 (игрок за белых победил) 0-1 (игрок за чёрных победил) 0.5-0.5 (ничья)

Имя атрибута	Тип данных	Значение по умолчанию	Формат ввода (маска)	Ограничение на значения атрибута	Комментарий
Очки черным	Дробное	Нет	Нет	Обязательный, либо 0, либо 1	+-- (победа игроку за белых, так как игрок за чёрных не явился) --+ (победа игроку за чёрных, так как игрок за белых не явился)
ID команды;	Целое	Нет	Нет	Обязательный, минимум – 1, уникальный, инкрементируемый	Не виден пользователю
Название команды;	Строка	Нет	Нет	Обязательный, уникальный, максимальная длина символов – 255	Исходя из предметной области, каждая команда должна иметь уникальное название, чтобы их можно было различать пользователю, который не видит ID команд
Атрибут команды;	Строка	Нет	Нет	Необязательный, максимальная длина – 10 символов	Может использоваться при разделении на группы
Статус команды;	Целое	1	Нет	Обязательный, 0 или 1	0 – команда неактивна, 1 – команда активна. При объявлении команды неактивной, организатор имеет право либо исключить всех игроков из команды и перенести в другие команды, либо сделать всех игроков команды неактивными
ID коэффициента;	Целое	Нет	Нет	Обязательный, инкрементируемый, не редактируемый	Идентификатор коэффициента
Название коэффициента.	Строка	Нет	Нет	Обязательный, не редактируемый	Название коэффициента для отображения пользователю

ПРИЛОЖЕНИЕ Д

Листинги встроенных процедур

```

CREATE OR REPLACE PROCEDURE __add_player(
    IN p_tournament_id INTEGER,
    IN p_organizer_id INTEGER,
    IN p_player_last_name VARCHAR(255),
    IN p_player_first_name VARCHAR(255),
    IN p_gender CHARACTER(1),
    IN p_player_attribute CHARACTER(3),
    IN p_team_name VARCHAR(255),
    IN p_birth_year INTEGER DEFAULT 1900,
    IN p_board_number INTEGER DEFAULT 1)
LANGUAGE plpgsql
AS
$$
BEGIN
    -- Check if organizer exists.
    IF NOT EXISTS(SELECT 1
                  FROM users
                  WHERE user_id = p_organizer_id) THEN
        RAISE EXCEPTION 'Organizer with id % does not exist', p_organizer_id;
    END IF;
    -- Check if tournament exists.
    IF NOT EXISTS(SELECT 1
                  FROM tournaments
                  WHERE tournament_id = p_tournament_id
                        AND organizer_id = p_organizer_id) THEN
        RAISE EXCEPTION 'Tournament with id % and organizer id % does not exist', p_tournament_id, p_organizer_id;
    END IF;
    -- Check if team exists.
    IF NOT EXISTS(SELECT 1
                  FROM teams
                  WHERE p_team_name IN (SELECT team_name
                                         FROM teams
                                         WHERE tournament_id = p_tournament_id
                                               AND organizer_id = p_organizer_id)) THEN
        RAISE EXCEPTION 'Team with name % does not exist', p_team_name;
    END IF;
    -- Check if player with the same name and attribute already exists.
    IF EXISTS(SELECT 1
              FROM players
              WHERE tournament_id = p_tournament_id
                    AND organizer_id = p_organizer_id
                    AND player_last_name = p_player_last_name
                    AND player_first_name = p_player_first_name
                    AND player_attribute = p_player_attribute) THEN
        RAISE EXCEPTION 'Player with the same name and attribute already exists';
    END IF;
    -- Check if board number is greater than count of players in team.
    IF p_board_number - 1 > (SELECT COUNT(*)
                              FROM players
                              WHERE tournament_id = p_tournament_id
                                    AND organizer_id = p_organizer_id
                                    AND team_id = (SELECT team_id
                                                   FROM teams
                                                   WHERE teams.team_id = players.team_id
                                                         AND teams.tournament_id = players.tournament_id
                                                         AND teams.organizer_id = players.organizer_id)) THEN
        RAISE EXCEPTION 'Board number is greater than count of players in team';
    END IF;
    INSERT INTO players (tournament_id, organizer_id, player_last_name, player_first_name, gender, player_attribute,
                         team_id, player_birth_year, board_number)
    VALUES (p_tournament_id, p_organizer_id, p_player_last_name, p_player_first_name, p_gender, p_player_attribute,
            (SELECT team_id
             FROM teams
             WHERE tournament_id = p_tournament_id
                   AND organizer_id = p_organizer_id
                   AND team_name = p_team_name), p_birth_year, p_board_number);
END;
$$;

```

Рисунок Д1 – Процедура добавления игрока

```

CREATE OR REPLACE PROCEDURE __add_team(
    IN p_tournament_id INT,
    IN p_organizer_id INT,
    IN p_team_name VARCHAR(255),
    IN p_team_attribute CHARACTER(3) DEFAULT 'NoA'
    LANGUAGE plpgsql
AS
$$
BEGIN
    -- Check if organizer exists
    IF NOT EXISTS(SELECT user_id
                  FROM users
                  WHERE user_id = p_organizer_id) THEN
        RAISE EXCEPTION 'Organizer with id % does not exist', p_organizer_id;
    END IF;
    -- Check if tournament exists
    IF NOT EXISTS(SELECT 1
                  FROM tournaments
                  WHERE tournament_id = p_tournament_id
                        AND organizer_id = p_organizer_id) THEN
        RAISE EXCEPTION 'Tournament with id % and organizer id % does not exist', p_tournament_id, p_organizer_id;
    END IF;
    -- Check if team with the same name and attribute already exists
    IF EXISTS(SELECT 1
                  FROM teams
                  WHERE tournament_id = p_tournament_id
                        AND organizer_id = p_organizer_id
                        AND team_name = p_team_name
                        AND team_attribute = p_team_attribute) THEN
        RAISE EXCEPTION 'Team with the same name and attribute already exists';
    END IF;
    INSERT INTO teams (tournament_id, organizer_id, team_name, team_attribute)
    VALUES (p_tournament_id, p_organizer_id, p_team_name, p_team_attribute);
END;
$$;

```

Рисунок Д2 – Процедура добавления команды

```

CREATE OR REPLACE PROCEDURE __add_tournament(
    IN p_organizer_id INT,
    IN p_tournament_name VARCHAR(255),
    IN p_system_id INTEGER,
    IN p_kind_id INTEGER,
    IN p_tours_count INTEGER DEFAULT 7,
    IN p_place VARCHAR(255) DEFAULT '-',
    IN p_tournament_date_start DATE DEFAULT NOW()::DATE,
    IN p_tournament_time_start TIME WITHOUT TIME ZONE DEFAULT NOW()::TIME(6),
    IN p_duration INTEGER DEFAULT 0,
    IN p_max_team_players INTEGER DEFAULT 5,
    IN p_organization_name VARCHAR(255) DEFAULT '-',
    IN p_is_mixed_groups BOOLEAN DEFAULT TRUE
    LANGUAGE plpgsql
AS
$$
BEGIN
    -- Check if organizer exists
    IF NOT EXISTS(SELECT 1
                  FROM users
                  WHERE user_id = p_organizer_id) THEN
        RAISE EXCEPTION 'Organizer with id % does not exist', p_organizer_id;
    END IF;
    -- Check if tournament with the same name already exists
    IF EXISTS(SELECT 1
                  FROM tournaments
                  WHERE organizer_id = p_organizer_id
                        AND tournament_name = p_tournament_name) THEN
        RAISE EXCEPTION 'Tournament with the same name already exists';
    END IF;
    INSERT INTO tournaments (organizer_id, tournament_name, tours_count, place, date_start, time_start, duration,
                           max_team_players, organization_name, system_id, kind_id, is_mixed_groups)
    VALUES (p_organizer_id, p_tournament_name, p_tours_count, p_place, p_tournament_date_start, p_tournament_time_start,
            p_duration, p_max_team_players, p_organization_name, p_system_id, p_kind_id, p_is_mixed_groups);
END;
$$;

```

Рисунок Д3 – Процедура добавления турнира

```

CREATE OR REPLACE PROCEDURE __add_user(
    IN p_last_name VARCHAR(255),
    IN p_first_name VARCHAR(255),
    IN p_patronymic VARCHAR(255),
    IN p_email VARCHAR(255),
    IN p_password_hash VARCHAR(255),
    IN p_tour_limit INTEGER DEFAULT 50)
LANGUAGE plpgsql
AS
$$
BEGIN
    -- Check if user with the same email already exists
    IF EXISTS(SELECT 1
        |     FROM users
        |     WHERE email = p_email) THEN
        RAISE EXCEPTION 'User with the same email already exists';
    END IF;
    INSERT INTO users (user_lastname, user_firstname, user_patronymic, email, pass_hash, tournaments_lim)
    VALUES (p_last_name, p_first_name, p_patronymic, p_email, p_password_hash, p_tour_limit);
END;
$$;

```

Рисунок Д4 – Процедура добавления пользователя

```

CREATE OR REPLACE PROCEDURE __delete_player(
    IN p_player_id INTEGER,
    IN p_tournament_id INTEGER,
    IN p_organizer_id INTEGER)
LANGUAGE plpgsql
AS
$$
BEGIN
    -- Check if player exists
    IF NOT EXISTS(SELECT 1
        |     FROM players
        |     WHERE player_id = p_player_id
        |     AND tournament_id = p_tournament_id
        |     AND organizer_id = p_organizer_id) THEN
        RAISE EXCEPTION 'Player with id % does not exist', p_player_id;
    END IF;
    -- Check if player has any games
    IF EXISTS(SELECT 1
        |     FROM games
        |     WHERE (white_id = p_player_id
        |             | OR black_id = p_player_id)
        |             AND tournament_id = p_tournament_id
        |             AND organizer_id = p_organizer_id) THEN
        RAISE EXCEPTION 'Player has games';
    END IF;
    DELETE
    FROM players
    WHERE player_id = p_player_id
    |     AND tournament_id = p_tournament_id
    |     AND organizer_id = p_organizer_id;
END;
$$;

```

Рисунок Д5 – Процедура удаления игрока

```

CREATE OR REPLACE PROCEDURE __delete_team(
    IN p_team_id INTEGER,
    IN p_tournament_id INTEGER,
    IN p_organizer_id INTEGER)
LANGUAGE plpgsql
AS
$$
BEGIN
    -- Check if team exists
    IF NOT EXISTS(SELECT 1
        FROM teams
        WHERE team_id = p_team_id
            AND tournament_id = p_tournament_id
            AND organizer_id = p_organizer_id) THEN
        RAISE EXCEPTION 'Team with id % does not exist', p_team_id;
    END IF;
    -- Check if team has any games
    IF EXISTS(SELECT 1
        FROM games
        WHERE (white_id IN (SELECT player_id
            FROM players
            WHERE team_id = p_team_id
                AND tournament_id = p_tournament_id
                AND organizer_id = p_organizer_id)
            OR black_id IN (SELECT player_id
            FROM players
            WHERE team_id = p_team_id
                AND tournament_id = p_tournament_id
                AND organizer_id = p_organizer_id))
            AND tournament_id = p_tournament_id
            AND organizer_id = p_organizer_id) THEN
        RAISE EXCEPTION 'Team has games';
    END IF;
    DELETE
    FROM teams
    WHERE team_id = p_team_id
        AND tournament_id = p_tournament_id
        AND organizer_id = p_organizer_id;
END;
$$;

```

Рисунок Д6 – Процедура удаления команды

```

CREATE OR REPLACE PROCEDURE __delete_tournament(
    IN p_tournament_id INTEGER,
    IN p_organizer_id INTEGER)
LANGUAGE plpgsql
AS
$$
BEGIN
    -- Check if tournament exists
    IF NOT EXISTS(SELECT 1
        FROM tournaments
        WHERE tournament_id = p_tournament_id
            AND organizer_id = p_organizer_id) THEN
        RAISE EXCEPTION 'Tournament with id % and organizer id % does not exist', p_tournament_id, p_organizer_id;
    END IF;
    -- Check if tournament has any games
    IF EXISTS(SELECT 1
        FROM games
        WHERE tournament_id = p_tournament_id
            AND organizer_id = p_organizer_id) THEN
        RAISE NOTICE 'Tournament has games, they will be deleted';
    END IF;
    DELETE
    FROM tournaments
    WHERE tournament_id = p_tournament_id
        AND organizer_id = p_organizer_id;
END;
$$;

```

Рисунок Д7 – Процедура удаления турнира

```
CREATE OR REPLACE PROCEDURE __delete_user(
    IN p_user_id INTEGER)
    LANGUAGE plpgsql
AS
$$
BEGIN
    -- Check if user exists
    IF NOT EXISTS(SELECT 1
        FROM users
        WHERE user_id = p_user_id) THEN
        RAISE EXCEPTION 'User with id % does not exist', p_user_id;
    END IF;

    DELETE
    FROM users
    WHERE user_id = p_user_id;
END;
$$;
```

Рисунок Д8 – Процедура удаления пользователя

ПРИЛОЖЕНИЕ Е

Листинги триггеров

```
-- Trigger on updating player points when they play a game
CREATE OR REPLACE FUNCTION __update_points()
RETURNS TRIGGER
LANGUAGE plpgsql
AS
$update_points$
BEGIN
    UPDATE players
    SET points_count = points_count + NEW.white_points
    WHERE player_id = NEW.white_id
        AND tournament_id = NEW.tournament_id
        AND organizer_id = NEW.organizer_id;
    UPDATE players
    SET points_count = points_count + NEW.black_points
    WHERE player_id = NEW.black_id
        AND tournament_id = NEW.tournament_id
        AND organizer_id = NEW.organizer_id;
    RETURN NEW;
END;
$update_points$;

-- DROP TRIGGER IF EXISTS __update_points ON games;

CREATE TRIGGER __update_points
AFTER INSERT OR UPDATE
ON games
FOR EACH ROW
EXECUTE PROCEDURE __update_points();
```

Рисунок Е1 – Триггер обновления очков

```

-- Trigger on updating player win, loose and draw points when they play a game
CREATE OR REPLACE FUNCTION __update_win_loose_draw()
RETURNS TRIGGER
LANGUAGE plpgsql
AS
$update_win_loose_draw$
BEGIN
    IF NEW.white_points = 1 THEN
        UPDATE players
        SET wins_count = wins_count + 1
        WHERE player_id = NEW.white_id
        AND tournament_id = NEW.tournament_id
        AND organizer_id = NEW.organizer_id;
    UPDATE players
    SET losses_count = losses_count + 1
    WHERE player_id = NEW.black_id
    AND tournament_id = NEW.tournament_id
    AND organizer_id = NEW.organizer_id;
ELSIF NEW.white_points = 0.5 THEN
    UPDATE players
    SET draws_count = draws_count + 1
    WHERE player_id = NEW.white_id
    AND tournament_id = NEW.tournament_id
    AND organizer_id = NEW.organizer_id;
    UPDATE players
    SET draws_count = draws_count + 1
    WHERE player_id = NEW.black_id
    AND tournament_id = NEW.tournament_id
    AND organizer_id = NEW.organizer_id;
ELSIF NEW.white_points = 0 THEN
    UPDATE players
    SET losses_count = losses_count + 1
    WHERE player_id = NEW.white_id
    AND tournament_id = NEW.tournament_id
    AND organizer_id = NEW.organizer_id;
    UPDATE players
    SET wins_count = wins_count + 1
    WHERE player_id = NEW.black_id
    AND tournament_id = NEW.tournament_id
    AND organizer_id = NEW.organizer_id;
END IF;
RETURN NEW;
END;
$update_win_loose_draw$;

-- DROP TRIGGER IF EXISTS __update_win_loose_draw ON games;

CREATE TRIGGER __update_win_loose_draw
AFTER INSERT OR UPDATE
ON games
FOR EACH ROW
EXECUTE PROCEDURE __update_win_loose_draw();

```

Рисунок Е2 – Триггер обновления результатов игроков

ПРИЛОЖЕНИЕ Ж

Листинги представлений

```
CREATE OR REPLACE VIEW
    players_list_view
AS
SELECT p.player_id,
       p.tournament_id,
       p.organizer_id,
       p.player_last_name,
       p.player_first_name,
       p.gender,
       p.player_attribute,
       (SELECT g.identity
        FROM groups g
        WHERE p.group_id = g.group_id) AS group_ident,
       (SELECT team_name
        FROM teams
        WHERE team_id = p.team_id
          AND organizer_id = p.organizer_id
          AND tournament_id = p.tournament_id) AS team_name,
       p.player_birth_year,
       p.is_active
FROM players p;
```

Рисунок Ж1 – Представление списка игроков

```
CREATE OR REPLACE VIEW teams_list_view AS
SELECT t.team_id,
       t.tournament_id,
       t.organizer_id,
       t.team_name,
       t.team_attribute,
       (SELECT COUNT(*)
        FROM players
        WHERE team_id = t.team_id
          AND organizer_id = t.organizer_id
          AND tournament_id = t.tournament_id) AS players_count,
       t.is_active
FROM teams t;
```

Рисунок Ж2 – Представление списка команд

```
CREATE OR REPLACE VIEW team_view AS
SELECT t.team_id,
       t.tournament_id,
       t.organizer_id,
       p.player_id,
       p.board_number,
       p.player_last_name,
       p.player_first_name,
       (SELECT g.identity
        FROM groups g
        WHERE p.group_id = g.group_id) AS group_ident,
       p.is_active
FROM teams t
     JOIN players p
       ON (t.organizer_id = p.organizer_id
           AND t.tournament_id = p.tournament_id
           AND t.team_id = p.team_id)
ORDER BY p.board_number;
```

Рисунок Ж3 – Представление команды

```
CREATE OR REPLACE VIEW single_rating_list_view AS
SELECT p.player_id,
       p.tournament_id,
       p.organizer_id,
       DENSE_RANK() OVER (PARTITION BY p.tournament_id, p.organizer_id
                           ORDER BY p.points_count DESC, p.ratio_sum1 DESC, p.ratio_sum2 DESC ) AS player_rank,
       p.player_last_name,
       p.player_first_name,
       p.player_attribute,
       (SELECT g.identity
        FROM groups g
        WHERE p.group_id = g.group_id) AS group_ident,
       p.wins_count,
       p.draws_count,
       p.losses_count,
       p.points_count,
       p.ratio_sum1,
       p.ratio_sum2
FROM players p
ORDER BY p.points_count DESC, p.ratio_sum1 DESC, p.ratio_sum2 DESC;
```

Рисунок Ж4 – Представление одиночного рейтинг-листа

```

CREATE OR REPLACE VIEW team_rating_list_view AS
WITH team_points AS
    (SELECT t.team_id,
           t.tournament_id,
           t.organizer_id,
           t.team_name,
           t.team_attribute,
           (SELECT SUM(wins_count)
            FROM players
            WHERE team_id = t.team_id
              AND organizer_id = t.organizer_id
              AND tournament_id = t.tournament_id) AS wins_count,
           (SELECT SUM(draws_count)
            FROM players
            WHERE team_id = t.team_id
              AND organizer_id = t.organizer_id
              AND tournament_id = t.tournament_id) AS draws_count,
           (SELECT SUM(losses_count)
            FROM players
            WHERE team_id = t.team_id
              AND organizer_id = t.organizer_id
              AND tournament_id = t.tournament_id) AS losses_count,
           (SELECT SUM(points_count)
            FROM players
            WHERE team_id = t.team_id
              AND organizer_id = t.organizer_id
              AND tournament_id = t.tournament_id) AS points_count,
           (SELECT SUM(ratio_sum1)
            FROM players
            WHERE team_id = t.team_id
              AND organizer_id = t.organizer_id
              AND tournament_id = t.tournament_id) AS ratio_sum1,
           (SELECT SUM(ratio_sum2)
            FROM players
            WHERE team_id = t.team_id
              AND organizer_id = t.organizer_id
              AND tournament_id = t.tournament_id) AS ratio_sum2
        FROM teams t)
SELECT tp.team_id,
       tp.tournament_id,
       tp.organizer_id,
       DENSE_RANK() OVER (PARTITION BY tp.tournament_id, tp.organizer_id
          | ORDER BY tp.points_count DESC, tp.ratio_sum1 DESC, tp.ratio_sum2 DESC ) AS team_rank,
       tp.team_name,
       tp.team_attribute,
       tp.wins_count,
       tp.draws_count,
       tp.losses_count,
       tp.points_count,
       tp.ratio_sum1,
       tp.ratio_sum2
  FROM team_points tp
 ORDER BY tp.points_count DESC, tp.ratio_sum1 DESC, tp.ratio_sum2 DESC;

```

Рисунок Ж5 – Представление командного рейтинг-листа

ПРИЛОЖЕНИЕ З

Текст программы

Так как программа содержит множество модулей, классов, то объём всего кода занимает значительную часть, из-за чего было принято решение осуществить демонстрацию кода на веб хостинге программных проектов GitHub. Ссылка на репозиторий: <https://github.com/AleksanderNekr/ChessTourManager>.