

Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики
и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Языки Программирования (ЯП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе
на тему:

«ИГРА ФУТБОЛ»

БГУИР КП 1-40 01 01 03 014 ПЗ

Студент: гр. 551005 Коваленко И.А.

Руководитель: асс. Марина И.М.

Минск 2016

Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ПОИТ

(подпись)

Лапицкая Н.В. 2016 г.

ЗАДАНИЕ

по курсовому проектированию

Студенту Коваленко Илье Андреевичу

1. Тема работы Игра Футбол
2. Срок сдачи студентом законченной работы 20.12.2016
3. Исходные данные к работе Среда программирования RAD Studio 10.1 Berlin, модули стандартной библиотеки C++, модули библиотеки RAD Studio
4. Содержание расчётно-пояснительной записки (перечень вопросов, которые подлежат разработке)

Введение.

1. Аналитический обзор литературы и существующих аналогов;

2. Разработка алгоритма;

3. Разработка программного средства;

4. Обоснование технических приемов программирования;

5. Руководство пользователя программы;

Заключение, список литературы, ведомость, приложения.

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

1. Схема программы

6. Консультант по курсовому проекту Марина И.М.

7. Дата выдачи задания 10.09.2016 г.

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и процентом от общего объема работы):

раздел 1, введение к 20.09.2016 – 10 % готовности работы;

разделы 2 к 15.10.2016 – 30 % готовности работы;

разделы 3,4 к 15.11.2016 – 60 % готовности работы;

раздел 5 к 05.12.2016 – 90 % готовности работы;

оформление пояснительной записки и графического материала к 10.05.2016 – 100 % готовности работы.

Защита курсового проекта с 10.12 по 20.12 2016 г.

РУКОВОДИТЕЛЬ _____ И.М. Марина

(подпись)

Задание принял к исполнению _____ 11.09.2016 г.

(дата и подпись студента)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	5
1 АНАЛИЗ ИГРЫ ФУТБОЛ И АНАЛОГОВ ПО	6
1.1 Общие сведения о футболе и правила игры	6
1.2 Краткий анализ уже созданного ПО.....	10
1.3 Постановка требований к разрабатываемому ПО.....	13
2 ВЫБОР ПРОГРАММНЫХ СРЕДСТВ	14
3 РАЗРАБОТКА АЛГОРИТМА.....	15
4 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА	16
3.1 Выбор структур данных	16
3.2 Работа с файлами.....	20
5 ОБОСНОВАНИЕ ТЕХНИЧЕСКИХ ПРИЕМОВ ПРОГРАММИРОВАНИЯ	21
6 ПРОВЕРКА РАБОТЫ ПРОГРАММЫ.....	24
7 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ.....	25
7.1 Авторизация.....	25
7.2 Просмотр статистики	25
7.3 Просмотр информации об игре.....	25
7.4 Игра.....	26
7.5 Покупка улучшений	26
ЗАКЛЮЧЕНИЕ	27
СПИСОК ЛИТЕРАТУРЫ.....	28
ПРИЛОЖЕНИЕ А. ТЕКСТ МОДУЛЯ UnitAuth.....	29
ПРИЛОЖЕНИЕ Б. ТЕКСТ МОДУЛЯ UnitMain.....	32
ПРИЛОЖЕНИЕ В. ТЕКСТ МОДУЛЯ UnitMenu.....	43
ПРИЛОЖЕНИЕ Г. ТЕКСТ МОДУЛЯ UnitStat	50
ПРИЛОЖЕНИЕ Д. ТЕКСТ МОДУЛЯ UnitInfo	52
ПРИЛОЖЕНИЕ Е. ТЕКСТ МОДУЛЯ Классов	53

ВВЕДЕНИЕ

В игры с мячом играли во многих странах. В Китае такая разновидность называлась Чжу-Кэ. В древней Спарте игра называлась «Эпискирос», а в Древнем Риме «Харпастум». Где-то в Новое время в Брянских землях проводились игры, инвентарём которых был кожаный мяч размером с человеческую голову, набитый перьями. Эти состязания именовались как «шалыга» и «кила». Примерно в XIV веке итальянцы изобрели игру «Кальчо». Именно они завезли эту игру на Британские острова.

В XIX веке футбол в Англии приобрёл популярность, сравнимую с крикетом. В него играли в основном в колледжах. Но в некоторых колледжах правила разрешали ведение и передачу мяча руками, а в других напротив, запрещалось. Первая попытка создать единые правила была предпринята в 1846 году, когда встретились представители нескольких колледжей. Они установили первый свод правил. В 1855 году был основан первый специализированный футбольный клуб — «Шеффилд». В 1863 году после долгих переговоров был принят свод правил Футбольной Ассоциации Англии. Также были приняты размеры поля и ворот. А в 1871 году был основан Кубок Англии — старейший футбольный турнир в мире. В 1891 году было принято правило о пенальти. Но сначала пенальти билось не с точки, а с линии, которая также как и сейчас находилась на расстоянии 11 метров от ворот.

Планируется создать игру футбол, в которой пользователь будет играть против компьютера командами 3х3. При этом, игра будет с элементами развития своей команды.

В ходе курсового проектирования были выделены следующие разделы:

- 1) Анализ игры футбол и аналогов по. В данном разделе имеется анализ самой игры и аналогов разрабатываемого ПО.
- 2) Выбор программных средств. В данном разделе приводится аргументация выбора программных средств.
- 3) Разработка алгоритма. В данном разделе проводится разработка общей схемы работы программы.
- 4) Разработка программного средства. В разделе приводятся структуры данных и их использование.
- 5) Обоснование технических приемов программирования. В разделе поясняется выбор тех или иных методик программирования.
- 6) Проверка работы программы. Проводится проверка корректности работы программы.
- 7) Руководство пользователя. Пошаговые действия для пользователей игры.

1 АНАЛИЗ ИГРЫ ФУТБОЛ И АНАЛОГОВ ПО

1.1 Основные сведения о футболе и правила игры

Футбол (от англ. *foot* — ступня, *ball* — мяч) — командный вид спорта, в котором целью является забить мяч в ворота соперника ногами или другими частями тела (кроме рук) большее количество раз, чем команда соперника. В настоящее время самый популярный и массовый вид спорта в мире.

Правила игры:

Есть 17 официальных правил игры, каждое из которых содержит список оговорок и руководящих принципов. Эти правила предназначены для применения на всех уровнях футбола, хотя есть некоторые изменения для таких групп, как юниоры, взрослые, женщины и люди с ограниченными физическими возможностями. Законы очень часто формулировались в общих чертах, которые позволяют упростить их применения в зависимости от характера игры. Правила игры публикуются в ФИФА, но поддерживаются Международным советом футбольных ассоциаций (IFAB).

Каждая команда состоит максимум из одиннадцати игроков (без учета запасных), один из которых должен быть вратарём. Правила неофициальных соревнований могут уменьшить количество игроков, максимум до 7. Вратари являются единственными игроками, которым позволено играть руками при условии: они делают это в пределах штрафной площади у своих собственных ворот. Хотя есть различные позиции на поле, эти позиции не обязательны.

Отдельная футбольная игра называется матч, который в свою очередь состоит из двух таймов по 45 минут. Пауза между первым и вторым таймами составляет 15 минут, в течение которой команды отдыхают, а по её окончании меняются воротами.

Цель игры — забить мяч в ворота противника, сделать это как можно большее количество раз и постараться не допустить гола в свои ворота. Матч выигрывает команда, забившая большее количество голов.

В случае, если в течение двух таймов команды забили одинаковое количество голов, то или фиксируется ничья, или победитель выявляется согласно установленному регламенту матча. В этом случае может быть назначено дополнительное время — ещё два тайма по 15 минут каждый. Как правило, между основным и дополнительным временем матча командам предоставляется перерыв. Между дополнительными таймами командам даётся лишь время на смену сторон. Одно время в футболе существовало правило, по которому победителем объявлялась команда, пер-

вой забившая гол (правило «золотого гола») или выигрывавшая по окончании любого из дополнительных таймов (правило «серебряного гола»). В настоящий момент дополнительное время либо не играется вовсе, либо играется в полном объёме (2 тайма по 15 минут). Если в течение дополнительного времени победителя выявить не удаётся, проводится серия послематчевых пенальти, не являющихся частью матча: по воротам противника с расстояния 11 метров пробивается по пять ударов разными игроками. Если количество забитых пенальти у обеих команд будет равным, тогда пробиваются по одной паре пенальти, пока не будет выявлен победитель.

Поле:

Матчи могут проводиться как на полях с естественным, так и на полях с искусственным покрытием. Согласно официальным правилам игры в футбол искусственное покрытие должно быть зелёного цвета. Поле для игры имеет форму прямоугольника. Боковая линия обязательно должна быть длиннее линии ворот. Принималось решение о том, что размер поля должен составлять 100—110 м (110—120 ярдов) в длину и минимум 64—75 м (70—80 ярдов) в ширину, однако затем обязательность этого требования была приостановлена.

Разметка:

Ширина разметки

Разметка поля делается линиями шириной не более 12 см (5 дюймов); эти линии входят в площади, которые они ограничивают. Все линии должны быть одинаковой ширины.

Название линий поля

Две длинные линии, ограничивающие поле для игры, называются боковыми линиями; две короткие линии — лицевыми линиями или линиями ворот, так как на них располагаются ворота.

Средняя линия

Поле делится на две половины с помощью средней линии, соединяющей середины боковых линий. Посередине средней линии делается отметка центра поля — сплошной круг диаметром 0,3 м (1 фут). Вокруг центра поля проводится окружность радиусом 9,15 м (10 ярдов). С отметки центра поля в начале каждого из таймов основного и дополнительного времени, а также после каждого забитого гола, выполняется начальный удар. При исполнении начального удара все игроки должны находиться на своих половинах поля, а соперники выполняющей удар команды — и за

пределами центрального круга.

Площадь ворот

На каждой половине поля размечается *площадь ворот* — зона, из пределов которой выполняется удар от ворот.

Из точек на расстоянии 5,5 м (6 ярдов) от внутренней стороны каждой стойки ворот, под прямым углом к линии ворот, вглубь поля проводятся две линии. На расстоянии 5,5 м (6 ярдов) эти линии соединяются другой линией, параллельной линии ворот. Таким образом, размеры площади ворот — 18,32 м (20 ярдов) на 5,5 м (6 ярдов).

Штрафная площадь

На каждой половине поля размечается штрафная площадь — зона, в которой вратарь может играть руками, а в ворота команды, совершившей в своей штрафной площади нарушение, наказуемое штрафным ударом, будет назначен 11-метровый удар.

Из точек на расстоянии 16,5 м (18 ярдов) от внутренней стороны каждой стойки ворот, под прямым углом к линии ворот, вглубь поля проводятся две линии. На расстоянии 16,5 м (18 ярдов) эти линии соединены другой линией, параллельной линии ворот. Размеры штрафной площади, таким образом — 40,32 м (44 ярда) на 16,5 м (18 ярдов). В пределах штрафной площади, по центру линии ворот и на расстоянии 11 м (12 ярдов) от неё, наносится одиннадцатиметровая отметка — сплошной круг диаметром 0,3 м (1 фут). За пределами штрафной площади проводится дуга окружности радиусом 9,15 м (10 ярдов), центр которой находится на одиннадцатиметровой отметке. Данная дуга используется для расположения игроков команд при пробитии одиннадцатиметрового удара.

Угловые сектора

В каждом из четырёх углов поля проводится дуга радиусом 1 м (или 1 ярд) с центром в углу поля, ограничивающая сектор для исполнения угловых ударов.

На расстоянии 9,15 м (10 ярдов) от границ угловых секторов у боковых линий и линий ворот могут быть нанесены отметки (с внешней стороны линий, примыкающие к ним под прямым углом), используемые для определения расстояния, на котором находятся игроки при исполнении углового.

Также у каждого угла поля обязательно ставятся флаги на флагштоках высотой не менее 1,5 метра (5 футов), не имеющих сверху заострений.

Мяч:

Футбольный мяч должен быть сферической формы, используемый для игры в футбол, параметры которого регламентируются Правилом 2 Правил игры в футбол.

Мяч состоит из 3 частей: покрышки, подкладки и камеры.

Покрышка — верхняя оболочка мяча, по которой наносятся удары.

Подкладка — средняя оболочка мяча, от толщины которой зависит прочность мяча (чем толще, чем прочнее)

Камера — центральная оболочка мяча, в которую закачивается воздух для более удобного нанесения ударов.

Стандартные положения:

В футболе бывают стандартные положения. Стандартными положениями в футболе являются штрафные, свободные, угловые и прочие удары, которые наносятся по сигналу судьи.

Стандартными положениями являются:

- Начальный удар. Наносится в начале каждого тайма, а также — после каждого забитого мяча. Назначается с центральной точки поля (в центральном круге).
- Вбрасывание мяча (аут). Бросается руками из-за боковой линии. Назначается после того, как мяч эту самую боковую линию пересёк. При этом аут бросает соперник игрока, которого мяч коснулся последним перед уходом за боковую линию.
- Удар от ворот. Наносится вратарём, после того, как мяч полностью пересёк линию ворот (вне территории ворот) от игрока нападавшей команды.
- Угловой удар. Наносится игроком нападающей команды из углового сектора. Назначается в случае, если мяч полностью пересекает линию ворот (вне территории ворот) от игрока защищавшейся команды.
- Свободный удар. Назначается в случае опасной игры против соперника (несостоявшееся нарушение) в ворота команды, которая совершила опасную игру. Пробивается с точки, где произошёл момент опасной игры. Гол, забитый прямым ударом со свободного не засчитывается.
- Штрафной удар. Назначается в случае нарушения правил в ворота команды, нарушившей правила. Может быть назначен только за пределами штрафной площадки команды-нарушителя (в случае, если фол произошёл в пределах штрафной, назначается пенальти). Штрафной также, как и свободный удар

пробивается с точки нарушения. Гол, забитый прямым ударом со штрафного засчитывается.

- Пенальти. Наносится со специальной отметки, расположенной в 11 метрах от ворот. Назначается в случае, если игрок нарушает правила в своей штрафной.
- Спорный мяч. Совершается судьёй, скидывающим мяч между двумя игроками-соперниками. Назначается в случае, если игра была остановлена в ситуации, не связанной с правилами. Напоминает хоккейное вбрасывание.

1.2 Краткий анализ уже созданного ПО

В футбольном менеджере «11х11», и надеемся, что игра доставит вам удовольствие! Жанр футбольного менеджера предполагает, что участник выступает в качестве главного тренера, его основная задача — подобрать наиболее эффективную тактику против конкретного соперника, правильно выставить настройки на игру и успешно выступить в турнире, умело распределив силы футболистов на серию матчей.

Игроки команды



















№	Игрок	Нац	Поз	Воз	Мас	Ф/г	Р/м	Мор	Тал	Бонусы
1	Руслан Кареев		Gk	35	72	100	72	0	4	
2	Имре Диани		Gk	30	96	100	96	0	4	
3	Константин Магомедов		Ld	33	108	100	108	0	3	
4	Фратнисек Валичек		Ld	26	89	100	89	0	5	
5	Александр Дубинский		Cd	23	83	100	83	0	1	
6	Мехди Газуани		Cd	33	95	100	95	0	4	
7	Артур Акимов		Cd	24	63	100	63	0	2	
8	Кевин Диллас		Rd	23	69	100	69	0	3	
9	Джамалудин Окрошидзе		Rd	22	70	100	70	0	3	
11	Севериано Битенкурт		Lm	29	66	100	66	0	6	
10	Пётр Большаков		Lm	19	66	100	66	0	4	
13	Владлен Карсаков		Cm	34	78	100	78	0	4	
12	Василий Тувин		Cm	25	71	100	71	0	3	
14	Влад Ляпкин		Rm	30	100	100	100	0	1	
15	Сослан Русинов		Lf	22	64	100	64	0	3	
16	Геннадий Шумилов		Cf	26	94	100	94	0	3	
17	Алексей Котовец		Cf	36	107	100	107	0	1	
18	Валентин Поддубский		Rf	37	99	100	99	0	1	
				28	83		83			

Рисунок 1.2.1 – Список игроков 11х11

Ваша цель в игре — одержать победу в наибольшем количестве матчей, набрать максимальное количество баллов в призовых турнирах и занять достойное

место в рейтинге.



Рисунок 1.2.2 – Перед началом матча 11x11

После регистрации в игре Вы получаете команду, состоящую из 18 футболистов, каждый из которых уже имеет специализацию в соответствии с его позицией на поле. Перейдя в список игроков, Вы можете посмотреть их характеристики.

Для того, чтобы сыграть товарищеский матч, следует либо подать заявку на странице Матчи — Товарищеские матчи, либо там же вызвать соперника, чья заявка уже находится в списке.

После подтверждения заявки открывается окно настройки тактики (билдер) команды. Вы можете выбрать расстановку из списка стандартных или создать собственный вариант, перетаскивая мышкой игроков на поле. При несоответствии позиции и амплуа футболиста, обозначение амплуа подсвечивается красным цветом. Замены могут производиться автоматически, при получении травмы футболист заменяется запасным с тем же амплуа или же, если такого нет, другим полевым игроком. Чтобы замена была возможна, запасных игроков следует поместить на "скамейку запасных" - места сразу под зеленым полем в окне настройки-расстановки состава на матч. По ссылке "Замены" можно выбрать пары футболистов и выставить время замен (число в минутах от начала матча), и тогда они будут происходить во время ближайшей остановки игры после наступления времени замены. Тактика на матч настраивается путем выбора баланса нападение-защита, а также выбирается манера игры из стандартного набора инструкций.

Дополнительно могут быть определены исполнители во время стандартных положений - угловой, штрафной, пенальти.

FIFA — серия симуляторов футбола, которая разрабатывается студией EA Canada, входящей в состав корпорации Electronic Arts. Игры издаются под брендом EA Sports™, каждый год выходит новое издание, включающее в себя изменения, произошедшие в футбольном мире за год. Electronic Arts владеет множеством лицензий на использование в игре футбольных лиг разных стран и игроков, выступающих в этих чемпионатах.



Рисунок 1.2.3 – Скриншот из игры FIFA

Особенности FIFA:

- Переработанные стандарты
- Активная система интеллекта
- Переработанная физическая модель
- Атакующие возможности
- Кроссплатформенный движок FROSTBITE

1.3 Постановка требований к разрабатываемому ПО

Создать интересную игру с интуитивно понятным и красивым интерфейсом, в которой будет присутствовать довольно сильный искусственный интеллект. Так же, необходимо создать возможность прокачивать свою команду и зарабатывать на победах.

Создать:

- удобный и интуитивно понятный интерфейс для пользователя, играющего в футбол
- реализовать систему прокачки команды, каждого игрока в отдельности, своего игрового аккаунта
- создать систему статистики и механизм определения места игрока в ней
- создать понятное описание игры и цен на прокачку своего аккаунта и команды

2 ВЫБОР ПРОГРАММНЫХ СРЕДСТВ

В качестве языка программирования был выбран C++.

C++ — компилируемый, статически типизированный язык программирования общего назначения.

C++ широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования. Область его применения включает создание операционных систем, разнообразных прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также развлекательных приложений (игр). Существует множество реализаций языка C++, как бесплатных, так и коммерческих и для различных платформ.

Платформой была выбрана Windows, так как она лучше других подходит для разработки игр.

В качестве среды программирования была выбрана лицензионная коммерческая среда RAD Studio 10.1 Berlin Starter Edition как наиболее удобная для создания пользовательского интерфейса.

В качестве фреймворка был выбран FireMonkey, который отлично подходит для созданий анимаций, различных эффектов и отрисовки плавной картинки.

3 РАЗРАБОТКА АЛГОРИТМА

В ходе курсового проектирования была создана следующая схема работы программы, отображающая общий алгоритм ее работы.

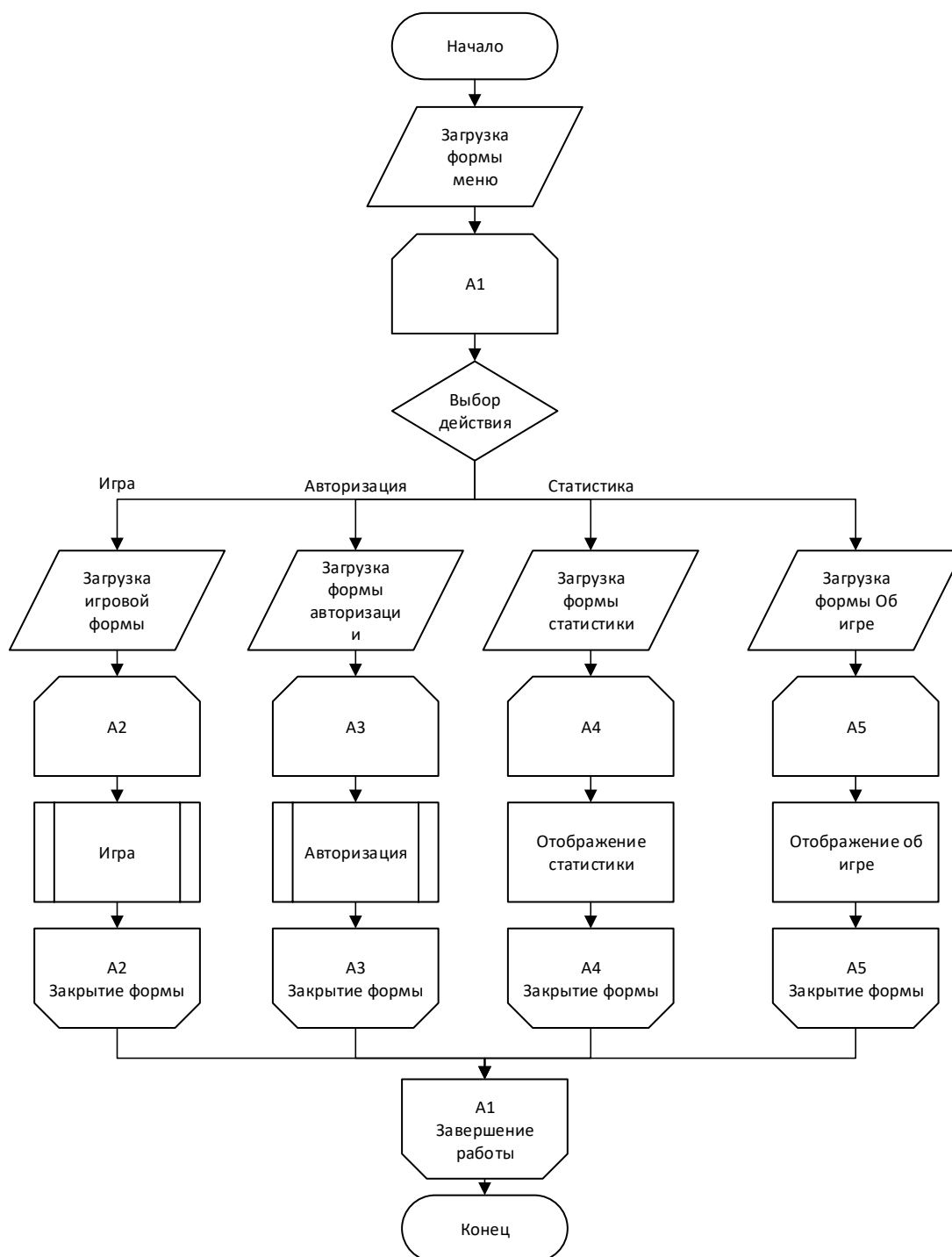


Схема 2.1 – Схема работы программы

4 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

4.1 Выбор структур данных

При разработке приложения было решено использовать объектно-ориентированную парадигму программирования, чтобы создать более наглядную и удобную для модифицирования структуру приложения. Все классы были вынесены в отдельные файлы и подключались к нужным формам только по мере необходимости.

Были созданы следующие классы:

TPlayer

- name
- surname
- level
- speed

name – хранение имени игрока команды

surname – фамилия игрока команды

level – уровень игрока команды, который можно увеличивать по ходу игры

speed – скорость игрока, которую можно прокачивать по ходу игры

TProfile

- Team[3]
- name
- cash
- wins
- write()
- read()

Team[3] – массив из трех игроков, определенных в классе TPlayer

name – имя пользователя (логин)

level – уровень аккаунта пользователя

cash – баланс пользователя

wins – количество побед пользователя

write() и read() – методы для вывода и ввода информации из файла в класс TProfile

TUdb

- vector<TProfile> user
- pid
- filename
- out()
- in()
- SortForStat()
- IsExist()
- Reg()
- Log()
- GetUser()
- AddWin()
- BuyLvl()
- BuyAccountLvl()
- length()

user – динамический массив, хранящий профили всех пользователей игры

pid – объект класса, определяющий текущего пользователя, который на данный момент авторизирован в игре

filename – имя файла с базой данных

out() – метод для вывода содержимого user в файл базы данных с название filename (необходимо при закрытии приложения)

in() – метод для ввода содержимого user из файла в динамический массив (необходимо при запуске приложения)

SortForStat() – метод класса для сортировки пользователей по убыванию количества побед (необходимо для вывода статистики)

IsExist() – метод, проверяющий, существует ли такой пользователь в базе или нет

Reg() – метод для регистрации пользователя

Log() – метод для авторизации пользователя

GetUser() – метод для получения пользователя с заданным именем

AddWin() – метод для добавления победы определенному пользователю

BuyLvl() – метод для покупки скорости определенному игроку определенного пользователя

BuyAccountLvl() – метод для покупки уровня аккаунта пользователя

length() – количество пользователей в базе данных

TPid

- value
- filename
- out()
- in()

value – номер текущего игрока в базе данных

filename – имя файла, хранящего идентификатор текущего пользователя

out() – метод для записи информации из базы value в файл filename

in() – метод для чтения информации из файла filename в value

TGame

- InitGame()
- CanMove
- WinCount
- Count[2]
- TimerCount
- Winner
- GoalTo()

InitGame – инициализация игры при заходе игрока в игру

CanMove – могут ли игроки двигаться или нет (необходимо при тайм-аутах между матчами)

WinCount – количество игр, которые необходимо выиграть для победы

Count[2] – массив, хранящий счет каждой команды в целочисленном виде

TimerCount – обратный отсчет таймера (необходим для регулирования количества секунд, после которых игроки смогут двигаться)

Winner – в случае победы в этой переменной будет номер выигравшей команды

GoalTo() – метод для фиксации гола в ворота одной из команд

TFiPlayer

- Speed
- mouseX
- mouseY
- InGame
- Img
- Txt

Speed – скорость игрока на поле

mouseX, mouseY – точка с координатами на игровом поле, указывающая, куда следует двигаться игроку команды

InGame – переменная, указывающая, в игре ли игрок или нет

Img – указатель на картинку игрока на поле

Txt – указатель на текст с номером игрока на поле (порядковый номер игрока в команде)

TTeam

- Player[TeamSize]
- CurrPlayer
- GetAmount()
- AddPlayer()
- DeletePlayer()

Player[] – массив с игроками команды

CurrPlayer – идентификатор игрока, которым сейчас управляет пользователь

GetAmount() – метод, возвращающий количество игроков в игре

AddPlayer() – метод, позволяющий добавить игрока в команду. В случае успешного добавления метод возвращает true.

DeletePlayer() – метод, позволяющий удалить игрока с поля с заданным идентификатором

TBall

- Speed
- Accelerate
- Degree
- CBall

Speed – текущая скорость мяча

Accelerate – ускорение мяча (для равнозамедленного движения)

Degree – угол направления движения мяча

CBall – указатель на изображения мяча

TField

- Team[2]
- Ball

- Background
- CalcDegree()
- CalcDistance()

Team[2] – указатель на массив из двух команд типа TTeam

Ball – указатель на футбольный мяч

Background – указатель на изображение игрового поля

CalcDegree() – метод для подсчета угла между двумя точками

CalcDistance() – метод для подсчета расстояния в пикселях между двумя точками

4.2 Работа с файлами

Для работы с файлами были использованы потоки ifstream и ofstream. Для хранения пользователей был создан файл db.bin. В программе пользователи хранятся в объекте класса Udb. Ввод и вывод базы данных db.bin осуществляется при помощи методов класса TUdb. Все поля записываются, а затем и считываются последовательно.

Для хранения идентификатора текущего пользователя используется файл pid.bin, а ввод и вывод из файла осуществляется методами класса TPid. Поля записываются и считываются последовательно.

Для хранения информации о текущем выигрыше используется файл win.bin. В случае выигрыша в него записывается сумма выигрыша, которая затем перечисляется на счет победителя.

Работа с картинками и их загрузка производилась методом LoadFromFile() стандартной библиотеки FireMonkey.

5 ОБОСНОВАНИЕ ТЕХНИЧЕСКИХ ПРИЕМОВ ПРОГРАММИРОВАНИЯ

Так как выбранный язык программирования является объектно-ориентированным, было решено описать при помощи классов все сущности, используемые в программе.

Код программы был разбит на следующие модули:

- UnitAuth
- UnitMain
- UnitMenu
- UnitStat
- UnitInfo

В модуле UnitAuth реализована форма авторизации. Можно либо войти в уже существующий аккаунт либо зарегистрировать новый. В процессе проектирования было решено реализовать авторизацию пользователей без использования паролей. Так как все данные о пользователях хранятся в текстовых файлах и никаким образом не защищены, использование пароля не является целесообразным.

В модуле UnitMain реализовано игровое поле, на котором происходят все основные действия.

В модуле UnitMenu реализовано основное меню игры, из которого можно вызывать другие формы, приобрести уровни аккаунта, либо начать игру с компьютером. Открытие некоторых форм возможно только после авторизации или регистрации с авторизацией.

В модуле UnitStat реализован вывод статистики. Для более наглядного вывода статистики было решено создать метод SortForStat() класса TUdb, который позволяет сперва отсортировать по убыванию по уровню аккаунта, затем по количеству побед. Сперва сортировка происходит по уровню аккаунта, чтобы пользователи с одинаковым количеством побед правильно расположились в таблице лидеров. Пользователь, под которым в настоящий момент игрок залогинен выделяется в таблице символом > напротив имени.

В модуле UnitInfo содержится полное описание игры, способов управления, ценах на внутриигровые предметы либо улучшения. Было решено создать данную форму для того, чтобы новым игрокам было проще адаптироваться и сразу понять управление, которое довольно сильно отличается от описанных аналогов.

В обозначенных модулях содержатся функции для работы с окном. Методы,

отвечающие за логику работы программы по максимуму были вынесены в классы. Каждый класс описан в отдельном файле. К модулям форм подключаются в основном один-два заголовочных файла.

Так как в классах TUdb, TPlayer и некоторых других хранятся данные пользователей, данные о команде, перед завершением работы программы необходимо сохранить эти данные. Так как проект довольно небольшой, было решено отказаться от использования реляционных/нереляционных баз данных. В качестве хранилища используются текстовые файлы.

В C++ работать с файлами можно при помощи потоков либо при помощи C-подобного FILE*. В результате нескольких экспериментов было установлено, что под современные версии Windows и компилятор CLANG (RAD Studio 10.1) работа с файлами посредством потоков более безопасна и при этом программа работает стабильнее.

Так как количество пользователей может быть теоретически неограниченно, было решено использовать динамические массивы. В C++ в качестве динамического массива может выступить `vector<type>`. Так как прямой доступ к `vector<> user` в классе TUdb недопустимо, массиву `user` был выставлен модификатор доступа `private`. В виду очевидной необходимости в работе с массивом `user`, были реализованы следующие функции:

- `SortForStat()`
- `IsExist()`
- `Reg()`
- `Log()`
- `GetUser()`
- `AddWin()`
- `BuyLvl()`
- `BuyAccountLvl()`
- `length()`

`SortForStat` – сортировка вектора для вывода статистики на форму `UnitStat`

Перед регистрацией методом класса `reg()` производится проверка при помощи метода `IsExist()`, который, если вернул `false`, можно зарегистрировать пользователя. Для входа используется метод `log()`, который так же использует `IsExist()`.

Так как в приложении реализована возможность покупки как уровня аккаунта,

так и прокачки игроков путём покупки им скорости, необходимы методы для модификации базы данных TUdb и внесения изменений в соответствующие поля. Метод BuyLvl() позволяет приобрести +1 скорость для определенного игрока за 20,000 игровой валюты. Метод ButAccountLvl() позволяет приобрести +1 уровень аккаунта за 15,000 внутриигровой валюты.

Управление:

Перемещение игроков в игре происходит при помощи мыши (необходимо направить курсор в точку, в которую следует двигаться игроку). Чтобы переключаться между игроками используются цифровые клавиши 1-2-3. Чтобы ударить по воротам противника нужно нажать клавишу E(y), а чтобы просто ударить мяч в сторону от себя нужно нажать W(ц). Игроки команды противника управляются при помощи элементарного искусственного интеллекта.

6 ПРОВЕРКА РАБОТЫ ПРОГРАММЫ

В ходе создания курсового проекта была проведена проверка работы программы. Найденные ошибки и неточности были исправлены. Приложение функционирует исправно и готово к работе.

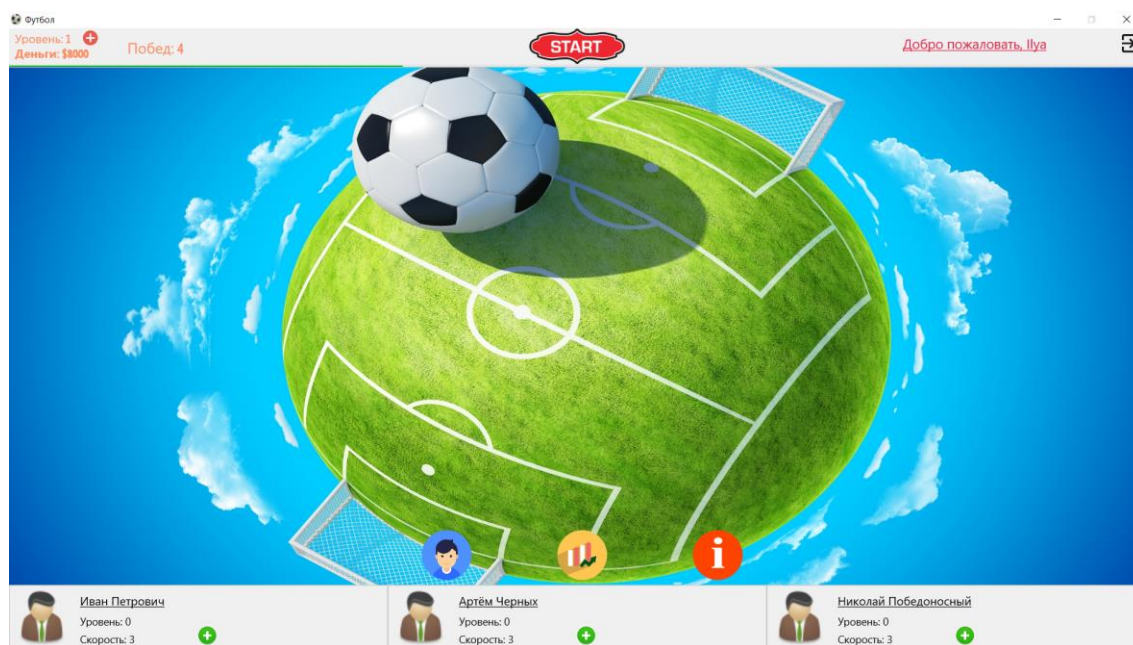


Рисунок 6.1 – Главное меню программы

7 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

7.1 Авторизация

Для регистрации необходимо:

- 1) Запустить программу.
- 2) Нажать кнопку «Авторизация».
- 3) Ввести уникальный логин.
- 4) Нажать кнопку «Зарегистрироваться».

Для входа необходимо:

- 1) Запустить программу.
- 2) Нажать кнопку «Авторизация».
- 3) Ввести существующий логин.
- 4) Нажать кнопку «Войти».

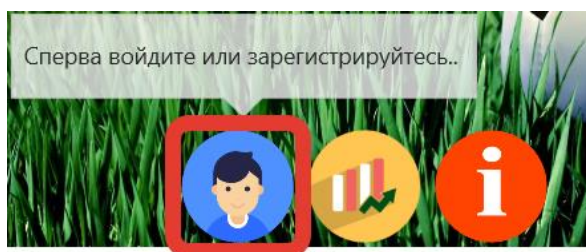


Рисунок 6.1.1 – Выбор кнопки «Авторизация»

7.2 Просмотр статистики

Для просмотра статистики необходимо:

- 1) Запустить программу.
- 2) Авторизироваться.
- 3) Нажать кнопку «Статистика».

7.3 Просмотр информации об игре

Для просмотра информации об игре необходимо:

- 1) Запустить программу.
- 2) Нажать кнопку «Об игре».

7.4 Игра

Для того чтобы сыграть матч против компьютера необходимо:

- 1) Запустить программу.
- 2) Авторизироваться.
- 3) Нажать кнопку «START».
- 4) Играть.

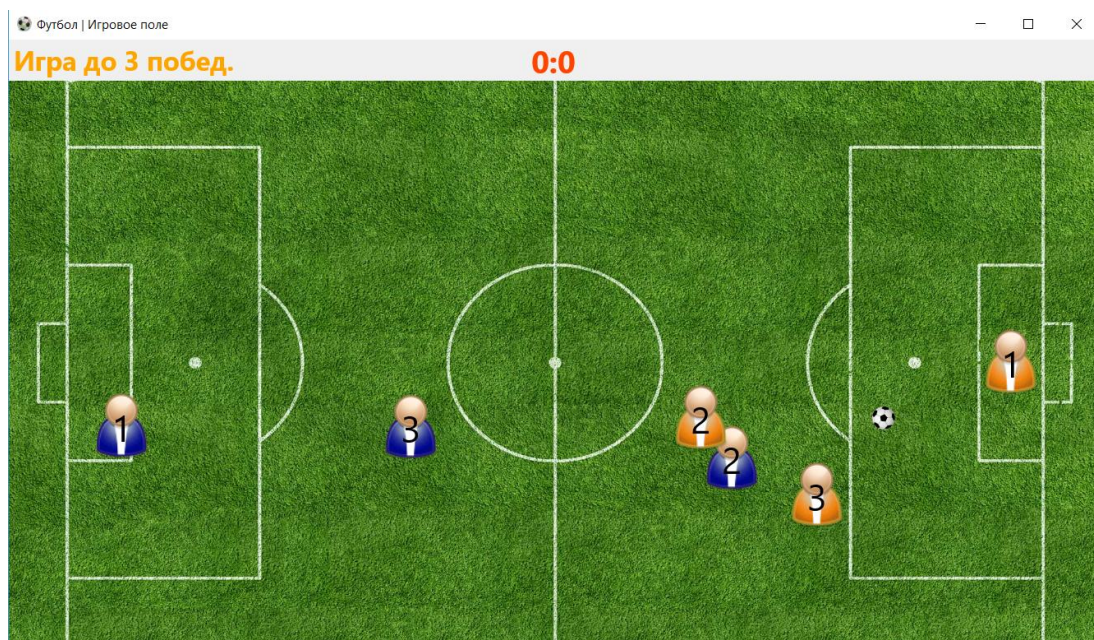


Рисунок 6.4.1 – Игровое поле

7.5 Покупка улучшений

Для покупки уровня аккаунта необходимо:

- 1) Запустить программу.
- 2) Авторизироваться.
- 3) Нажать кнопку покупки уровня аккаунта. (с игрового счета спишется 15,000 валюты)

Для покупки скорости игроку необходимо:

- 1) Запустить программу.
- 2) Авторизироваться.
- 3) Нажать кнопку покупки уровня аккаунта соответствующему игроку. (с игрового счета спишется 20,000 валюты)

ЗАКЛЮЧЕНИЕ

В результате курсового проектирования было создано приложение, полностью соответствующее заранее определенным требованиям.

Для написания приложения понадобилось знание классических алгоритмов и структур данных.

Были изучены материалы по языку C++, среде RAD Studio и фреймворку Fire-Monkey. В ходе изучения литературы были получены новые знания.

Изначально поставленные задачи достигнуты. Приложения получилось простым и понятным для пользователя.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Футбол википедия <https://ru.wikipedia.org/wiki/Футбол>
- [2] Игра 11x11 <http://11x11.ru>
- [3] FIFA википедия [https://ru.wikipedia.org/wiki/FIFA_\(серия_игр\)](https://ru.wikipedia.org/wiki/FIFA_(серия_игр))
- [4] C++ википедия <https://ru.wikipedia.org/wiki/C%2B%2B>
- [5] RAD Studio <https://www.embarcadero.com/ru/products/rad-studio>

ПРИЛОЖЕНИЕ А

(обязательное)

Текст модуля UnitAuth

```
//UnitAuth.h

#ifndef UnitAuthH
#define UnitAuthH
//-----
#include <System.Classes.hpp>
#include <FMX.Controls.hpp>
#include <FMX.Forms.hpp>
#include <FMX.ComboEdit.hpp>
#include <FMX.Controls.Presentation.hpp>
#include <FMX.Edit.hpp>
#include <FMX.StdCtrls.hpp>
#include <FMX.Types.hpp>
//-----
#include "ClassTUdb.h"
//-----
class TFormAuth : public TForm
{
__published:      // IDE-managed Components
    TButton *ButtonLogin;
    TButton *ButtonRegister;
    TEdit *EditUsername;
    TLabel *LabelUsername;
    TLabel *Label1;
    void __fastcall ButtonLoginClick(TObject *Sender);
    void __fastcall FormShow(TObject *Sender);
    void __fastcall ButtonRegisterClick(TObject *Sender);
    void __fastcall FormCloseQuery(TObject *Sender, bool &CanClose);
    void __fastcall EditUsernameKeyDown(TObject *Sender, WORD &Key,
System::WideChar &KeyChar,
    TShiftState Shift);
private:      // User declarations
public:      // User declarations
    __fastcall TFormAuth(TComponent* Owner);
};
//-----
extern PACKAGE TFormAuth *FormAuth;
//-----
#endif

//UnitAuth.cpp

#include <fmx.h>
#pragma hdrstop

#include "UnitAuth.h"
//-----
#pragma package(smart_init)
#pragma resource "*.fmx"
//-----
TFormAuth *FormAuth;
TUdb Udb;
//-----
__fastcall TFormAuth::TFormAuth(TComponent* Owner)
    : TForm(Owner)
{
}
}
```

```

//-----
string UnicodeToString(UnicodeString us) {
    string result = AnsiString(us.t_str()).c_str();
    return result;
}

//-----
void __fastcall TFormAuth::ButtonLoginClick(TObject *Sender)
{
    string username = UnicodeToString(EditUsername->Text);
    Udb.pid.value = Udb.log(username);
    if (Udb.pid.value > - 1) {
        string msg = "Вы были успешно авторизированы под ником " +
username + ".";
        ShowMessage(msg.c_str());
        //Udb.pid.out();
        FormAuth->Close();
    } else {
        string msg = "Пользователя с ником " + username + " не
существует. Сперва нужно зарегистрироваться.";
        ShowMessage(msg.c_str());
    }
}

//-----
void __fastcall TFormAuth::FormShow(TObject *Sender)
{
    Udb.pid.in();
    Udb.in();
    Label1->Text = ("Всего пользователей: " +ToStr(Udb.length()))>.c_str();
}

//-----
void __fastcall TFormAuth::ButtonRegisterClick(TObject *Sender)
{
    TProfile profile;
    profile.name = UnicodeToString(EditUsername->Text);
    profile.level = 0;
    profile.cash = 3000;
    profile.wins = 0;
    profile.Team[0].name = "Иван";
    profile.Team[0].surname = "Петрович";
    profile.Team[0].level = 0;
    profile.Team[0].speed = 3;

    profile.Team[1].name = "Артём";
    profile.Team[1].surname = "Черных";
    profile.Team[1].level = 0;
    profile.Team[1].speed = 3;

    profile.Team[2].name = "Николай";
    profile.Team[2].surname = "Победоносный";
    profile.Team[2].level = 0;
    profile.Team[2].speed = 3;

    if (Udb.reg(profile)) {
        string msg = "Вы были успешно зарегистрированы под ником " +
profile.name + ".";
        ShowMessage(msg.c_str());
    } else {
        string msg = "Пользователя с ником " + profile.name + " уже
существует. Можете войти.";

```

```

        ShowMessage(msg.c_str());
    }
    Udb.out();
    Udb.in();
    Label1->Text = ("Всего пользователей: " + ToString(Udb.length()) +
".").c_str();
}
//-----

void __fastcall TFormAuth::FormCloseQuery(TObject *Sender, bool &CanClose)
{
    Udb.pid.out();
    Udb.out();
    CanClose = true;
}
//-----

void __fastcall TFormAuth::EditUsernameKeyDown(TObject *Sender, WORD &Key,
System::WideChar &KeyChar,
TShiftState Shift)
{
    if (Key != 8) {
        if (!isalnum(KeyChar))
        {
            if (KeyChar == ' ')
            {
                KeyChar = '_';
                return;
            }
            KeyChar = 0;
            return;
        }
    }
}
//-----

```

ПРИЛОЖЕНИЕ Б

(обязательное)

Текст модуля UnitMain

```
//UnitMain.h

#ifndef UnitMainH
#define UnitMainH
//-----
#include <System.Classes.hpp>
#include <FMX.Controls.hpp>
#include <FMX.Forms.hpp>
#include <FMX.Objects.hpp>
#include <FMX.Types.hpp>
#include <math.h>
#include <FMX.Controls.Presentation.hpp>
#include <FMX.StdCtrls.hpp>
#include <FMX.Ani.hpp>
#include <FMX.ImgList.hpp>
#include <System.ImageList.hpp>
#include <FMX.MaterialSources.hpp>
#include <FMX.Effects.hpp>
#include <FMX.Viewport3D.hpp>
#include <FMX.Controls3D.hpp>
#include <FMX.Objects3D.hpp>
#include <System.Math.Vectors.hpp>
#include <FMX.Edit.hpp>
#include <IdBaseComponent.hpp>
#include <IdComponent.hpp>
#include <IdCustomTCPServer.hpp>
#include <IdTCPClient.hpp>
#include <IdTCPConnection.hpp>
#include <IdTCPServer.hpp>
#include <IdUDPBase.hpp>
#include <IdUDPClient.hpp>
#include <IdUDPServer.hpp>
#include <FMX.Menus.hpp>
#include <FMX.ComboEdit.hpp>
#include <IdContext.hpp>

#include "ClassTUdb.h"
#include <FMX.Media.hpp>
//-----
class TFormMain : public TForm
{
__published:      // IDE-managed Components
    TTimer *TimerMoveUser;
    TImage *ImageBackground;
    TCircle *Ball;
    TTimer *TimerMoveBall;
    TPanel *PanelPlayArea;
    TTimer *TimerCheckBall;
    TLabel *LabelFPS;
    TToolBar *ToolBarMenu;
    TLabel *LabelCount;
    TTimer *TimerCountDown;
    TTimer *TimerAI;
    TTimer *TimerMoveAI;
    TLabel *LabelCountDown;
    TLabel *LabelGameTo;
    TMediaPlayer *MediaPlayer1;
```



```

        void __fastcall TimerMoveUserTimer(TObject *Sender);
        void __fastcall TimerMoveBallTimer(TObject *Sender);
        void __fastcall FormKeyDown(TObject *Sender, WORD &Key, System::WideChar
&KeyChar,
                                TShiftState Shift);
        void __fastcall PanelPlayAreaMouseMove(TObject *Sender, TShiftState
Shift, float X, float Y);
        void __fastcall TimerCheckBallTimer(TObject *Sender);
        void __fastcall FormShow(TObject *Sender);
        void __fastcall FormCloseQuery(TObject *Sender, bool &CanClose);
        void __fastcall TimerCountDownTimer(TObject *Sender);
        void __fastcall TimerAITimer(TObject *Sender);
        void __fastcall TimerMoveAITimer(TObject *Sender);
private:    // User declarations
public:     // User declarations
        void __fastcall ATick();
        void __fastcall InitField();
        void __fastcall DestroyField();
        void __fastcall PrintCount();
        void __fastcall InitTimer(int Count);
        void __fastcall PlayMusic(UnicodeString s);
        TImage *tmpImage;
        TLabel *tmpLabel;
        __fastcall TFormMain(TComponent* Owner);
};

//-----
extern PACKAGE TFormMain *FormMain;
int UniqNum = 0;
//-----
#endif

//UnitMain.cpp

#include <fmx.h>
#pragma hdrstop

#include "UnitMain.h"
#include "ClassTField.h"
#include "ClassTGame.h"
//-----
#pragma package(smart_init)
#pragma resource "*.fmx"
#pragma resource ("*.Surface.fmx", _PLAT_MSWINDOWS)

// Константы, важные объявления
TFormMain *FormMain;
TField Field;
TUdb Udb;
TGame Game;
const int MainID = 1;
const int SecondID = 1 - MainID;

int DefSpeed[3];
double GoalKeeperDelta[2] = { (double)88/1062, (double)(1062 - 88)/1062};
double AttackerDelta[2] = { (double)472/1062, (double)(1062 - 472)/1062};
double DefenderDelta[2] = { (double)340/1062, (double)(1062 - 340)/1062};
double BallDelta[2] = { (double)182/1062, (double)(1062 - 182)/1062};
// Константы, важные объявления

//-----
__fastcall TFormMain::TFormMain(TComponent* Owner)

```

```

        : TForm(Owner)
    {
    }

void __fastcall TFormMain::PlayMusic(UnicodeString str)
{
    MediaPlayer1->Clear();
    MediaPlayer1->FileName = "music/" + str;
    MediaPlayer1->Play();
}

//-----
void __fastcall TFormMain::InitTimer(int Count)
{
    Game.TimerCount = Count;
    Game.CanMove = false;
    TimerCountDown->Enabled = true;
}

//-----
void __fastcall TFormMain::PrintCount()
{
    LabelCount->Text = (ToStr(Game.Count[0]) + ":" +
ToStr(Game.Count[1])).c_str();
}

//-----
void __fastcall TFormMain::InitField()
{
    InitTimer(5);
    PrintCount();

    // Background Start
    Field.Background = ImageBackground;
    Field.Background->Bitmap->LoadFromFile("img/field.jpg");
    // Background end

    // Ball Start
    Field.Ball.CBall = Ball;
    Field.Ball.Speed = 0;
    Field.Ball.Degree = 0;
    Field.Ball.CBall->Position->X = (PanelPlayArea->Width - Field.Ball.CBall-
>Width)/2;
    Field.Ball.CBall->Position->Y = (PanelPlayArea->Height -
Field.Ball.CBall->Height)/2;
    // Ball end

    // Players Start
    // - Team0 (Robotics)
    Field.Team[0].AddPlayer((GoalKeeperDelta[0]*PanelPlayArea->Width) -
PlayerSize/2, (PanelPlayArea->Height - PlayerSize)/2, 'b');
    Field.Team[0].AddPlayer((AttackerDelta[0]*PanelPlayArea->Width) -
PlayerSize/2, (PanelPlayArea->Height - PlayerSize)/2, 'b');
    Field.Team[0].AddPlayer((DefenderDelta[0]*PanelPlayArea->Width) -
PlayerSize/2, (PanelPlayArea->Height - PlayerSize)/2, 'b');
    // - Team1 (Main)
    Field.Team[1].AddPlayer((GoalKeeperDelta[1]*PanelPlayArea->Width) -
PlayerSize/2, (PanelPlayArea->Height - PlayerSize)/2, 'o');
    Field.Team[1].AddPlayer((AttackerDelta[1]*PanelPlayArea->Width) -
PlayerSize/2, (PanelPlayArea->Height - PlayerSize)/2, 'o');
    Field.Team[1].AddPlayer((DefenderDelta[1]*PanelPlayArea->Width) -
PlayerSize/2, (PanelPlayArea->Height - PlayerSize)/2, 'o');
    // Curr

```

```

        Field.Team[0].CurrPlayer = 1;
        Field.Team[1].CurrPlayer = 1;
        // Players end

        // Timers Start
        TimerMoveBall->Enabled = true;
        TimerMoveUser->Enabled = true;
        TimerMoveAI->Enabled = true;
        TimerCheckBall->Enabled = true;
        TimerAI->Enabled = true;
        // Timers end
    }

//-----
void __fastcall TFormMain::DestroyField()
{
    // Players Start
    for (int i = 0; i < 2; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            Field.Team[i].DeletePlayer(j);
        }
    }
    // Players end

    // Timers Start
    TimerMoveBall->Enabled = false;
    TimerMoveUser->Enabled = false;
    TimerMoveAI->Enabled = false;
    TimerCheckBall->Enabled = false;
    TimerAI->Enabled = false;
    // Timers end
}

//-----
void __fastcall TFormMain::TimerMoveUserTimer(TObject *Sender)
{
    if (Game.CanMove) {
        int TID = 1;
        if
(Field.CalcDistance(Field.Team[TID].Player[Field.Team[TID].CurrPlayer].Img-
>Position->X + Field.Team[TID].Player[Field.Team[TID].CurrPlayer].Img->Width/2,
Field.Team[TID].Player[Field.Team[TID].CurrPlayer].Img->Position->Y +
Field.Team[TID].Player[Field.Team[TID].CurrPlayer].Img->Height/2,
Field.Team[TID].Player[Field.Team[TID].CurrPlayer].mouseX,
Field.Team[TID].Player[Field.Team[TID].CurrPlayer].mouseY) > 3)
        {
            float Degree =
Field.CalcDegree(Field.Team[TID].Player[Field.Team[TID].CurrPlayer].Img->Position->X
+ Field.Team[TID].Player[Field.Team[TID].CurrPlayer].Img->Width/2,
Field.Team[TID].Player[Field.Team[TID].CurrPlayer].Img->Position->Y +
Field.Team[TID].Player[Field.Team[TID].CurrPlayer].Img->Height/2,
Field.Team[TID].Player[Field.Team[TID].CurrPlayer].mouseX,
Field.Team[TID].Player[Field.Team[TID].CurrPlayer].mouseY);
            for (int j = 0; j <=
Field.Team[TID].Player[Field.Team[TID].CurrPlayer].Speed; j++)
            {

                Field.Team[TID].Player[Field.Team[TID].CurrPlayer].Img->Position->X +=
(cos(Degree * pi / 180) * 1);
            }
        }
    }
}

```

```

        Field.Team[TID].Player[Field.Team[TID].CurrPlayer].Img->Position->Y -=
(sin(Degree * pi / 180) * 1);

        Field.Team[TID].Player[Field.Team[TID].CurrPlayer].Txt->Position->X +=
(cos(Degree * pi / 180) * 1);

        Field.Team[TID].Player[Field.Team[TID].CurrPlayer].Txt->Position->Y -=
(sin(Degree * pi / 180) * 1);

                                ImageBackground->Repaint();
                                }
                                }
        }

//-----
void __fastcall TFormMain::TimerMoveAITimer(TObject *Sender)
{
    if (Game.CanMove) {
        int TID = 0;
        for (int i = 0; i < 3; i++) {
            if
(Field.CalcDistance(Field.Team[TID].Player[i].Img->Position->X +
Field.Team[TID].Player[i].Img->Width/2, Field.Team[TID].Player[i].Img->Position->Y +
Field.Team[TID].Player[i].Img->Height/2, Field.Team[TID].Player[i].mouseX,
Field.Team[TID].Player[i].mouseY) > 3)
            {
                float Degree =
Field.CalcDegree(Field.Team[TID].Player[i].Img->Position->X +
Field.Team[TID].Player[i].Img->Width/2, Field.Team[TID].Player[i].Img->Position->Y +
Field.Team[TID].Player[i].Img->Height/2, Field.Team[TID].Player[i].mouseX,
Field.Team[TID].Player[i].mouseY);

                for (int j = 0; j <=
Field.Team[TID].Player[i].Speed; j++)
                {

                    Field.Team[TID].Player[i].Img->Position->X += (cos(Degree * pi / 180) *
1);

                    Field.Team[TID].Player[i].Img->Position->Y -= (sin(Degree * pi / 180) *
1);

                    Field.Team[TID].Player[i].Txt->Position->X += (cos(Degree * pi / 180) *
1);

                    Field.Team[TID].Player[i].Txt->Position->Y -= (sin(Degree * pi / 180) *
1);

                                ImageBackground->Repaint();
                                }
                                }
                }
            }
        }

//-----
void __fastcall TFormMain::TimerMoveBallTimer(TObject *Sender)
{
    if (Game.CanMove)
    {
        if (Field.Ball.Speed > 0)
        {

```

```

        for (int i = 0; i < (Field.Ball.Speed -
Field.Ball.Accelerate); i++)
        {
            if (!((cos(int(Field.Ball.Degree) *
pi / 180) * 1) + Ball->Position->X + Ball->Width/2 < ImageBackground->Width) ||
!((cos(int(Field.Ball.Degree) * pi / 180) * 1) + Ball->Position->X + Ball->Width/2 >
0))
            {
                // Отражение на 180
                Field.Ball.Degree = 180 -
Field.Ball.Degree;
                Field.Ball.Speed *= 0.8;
            }
            Ball->Position->X +=
(cos(Field.Ball.Degree * pi / 180) * 1);
            if (!(Ball->Position->Y + Ball-
>Height/2 - (sin(int(Field.Ball.Degree) * pi / 180) * 1) < ImageBackground->Height)
|| !(Ball->Position->Y + Ball->Height/2 - (sin(int(Field.Ball.Degree) * pi / 180) *
1) > 0))
            {
                // Отражение на 180
                Field.Ball.Degree = (180
- Field.Ball.Degree) * 2 + Field.Ball.Degree;
                Field.Ball.Speed *= 0.8;
            }
            Ball->Position->Y -=
(sin(Field.Ball.Degree * pi / 180) * 1);
            Ball->RotationAngle += 2 *
((cos(Field.Ball.Degree * pi / 180) * 1) - (sin(Field.Ball.Degree * pi / 180) * 1));
            Field.Ball.Speed -=
Field.Ball.Accelerate;
            Field.Background->Repaint();
        }
    }
}

//-----
void __fastcall TFormMain::TimerAITimer(TObject *Sender)
{
    if (Game.CanMove)
    {
        double BallX = Ball->Position->X + Ball->Width/2;
        double BallY = Ball->Position->Y + Ball->Height/2;

        // Player1 Нападающий
        double PlayX = Field.Team[SecondID].Player[1].Img->Position-
>X + Field.Team[SecondID].Player[1].Img->Width/2;
        double PlayY = Field.Team[SecondID].Player[1].Img->Position-
>Y + Field.Team[SecondID].Player[1].Img->Height/2;
        if ((Field.CalcDistance(Ball->Width/2 + Ball->Position->X,
Ball->Height/2 + Ball->Position->Y, Field.Team[SecondID].Player[1].Img->Position->X
+ Field.Team[SecondID].Player[1].Img->Width/2, Field.Team[SecondID].Player[1].Img-
>Position->Y + Field.Team[SecondID].Player[1].Img->Height/2) <= 12)) {
            Field.Ball.Speed += 3;
            Field.Ball.Degree = Field.CalcDegree(BallX,
BallY, ((1062 - 54)*PanelPlayArea->Width/1062), ((274 + rand() % 50 +
1)*PanelPlayArea->Height/545));
        }
        if (Field.CalcDistance(BallX, BallY, PlayX, PlayY) > 5)
        {
            Field.Team[SecondID].Player[1].Speed = 4;
            Field.Team[SecondID].Player[1].mouseX = BallX;

```

```

        Field.Team[SecondID].Player[1].mouseY = Bally;
    } else {
        Field.Team[SecondID].Player[1].Speed = -1;
    }

    // Player2 Воротарь
    PlayX = Field.Team[SecondID].Player[0].Img->Position->X +
Field.Team[SecondID].Player[0].Img->Width/2;
    PlayY = Field.Team[SecondID].Player[0].Img->Position->Y +
Field.Team[SecondID].Player[0].Img->Height/2;
    if ((Field.CalcDistance(Ball->Width/2 + Ball->Position->X,
Ball->Height/2 + Ball->Position->Y, Field.Team[SecondID].Player[0].Img->Position->X
+ Field.Team[SecondID].Player[0].Img->Width/2, Field.Team[SecondID].Player[0].Img-
>Position->Y + Field.Team[SecondID].Player[0].Img->Height/2) <= 15)) {
        Field.Ball.Speed += 3;
        Field.Ball.Degree = Field.CalcDegree(BallX,
Bally, ((1062 - 54)*PanelPlayArea->Width/1062), ((274 + rand() % 50 +
1)*PanelPlayArea->Height/545));
    }
    if (Field.CalcDistance(BallX, Bally, PlayX, PlayY) > 5 &&
(Bally > ((180)*PanelPlayArea->Height/545)) && (Bally < ((367)*PanelPlayArea-
>Height/545)))
    {
        Field.Team[SecondID].Player[0].Speed = 4;
        Field.Team[SecondID].Player[0].mouseX = PlayX;
        Field.Team[SecondID].Player[0].mouseY = Bally;
    } else {
        Field.Team[SecondID].Player[0].Speed = -1;
    }

    // Player3 Защитник
    PlayX = Field.Team[SecondID].Player[2].Img->Position->X +
Field.Team[SecondID].Player[2].Img->Width/2;
    PlayY = Field.Team[SecondID].Player[2].Img->Position->Y +
Field.Team[SecondID].Player[2].Img->Height/2;
    if ((Field.CalcDistance(Ball->Width/2 + Ball->Position->X,
Ball->Height/2 + Ball->Position->Y, Field.Team[SecondID].Player[2].Img->Position->X
+ Field.Team[SecondID].Player[2].Img->Width/2, Field.Team[SecondID].Player[2].Img-
>Position->Y + Field.Team[SecondID].Player[2].Img->Height/2) <= 35)) {
        Field.Ball.Speed += 3;
        Field.Ball.Degree = Field.CalcDegree(BallX,
Bally, ((1062 - 54)*PanelPlayArea->Width/1062), ((274 + rand() % 50 +
1)*PanelPlayArea->Height/545));
    }
    if (Field.CalcDistance(BallX, Bally, PlayX, PlayY) > 5)
    {
        Field.Team[SecondID].Player[2].Speed = 4;
        Field.Team[SecondID].Player[2].mouseX = PlayX;
        Field.Team[SecondID].Player[2].mouseY = Bally;
    } else {
        Field.Team[SecondID].Player[2].Speed = -1;
    }
}

//-----
void __fastcall TFormMain::FormKeyDown(TObject *Sender, WORD &Key, System::WideChar
&KeyChar,
        TShiftState Shift)
{
    if (!Game.CanMove) {return;}
    // Обработка ударов
    if (isalpha(KeyChar))

```

```

        {
            double dWidth =
Field.Team[SecondID].Player[Field.Team[SecondID].CurrPlayer].Img->Width/2;
            double dHeight =
Field.Team[SecondID].Player[Field.Team[SecondID].CurrPlayer].Img->Height/2;
            double dX =
Field.Team[SecondID].Player[Field.Team[SecondID].CurrPlayer].Img->Position->X;
            double dY =
Field.Team[SecondID].Player[Field.Team[SecondID].CurrPlayer].Img->Position->Y;
            int dSpeed = 5;

            switch (KeyChar) {
                case 'w': case 'W':

                    Field.Team[SecondID].Player[Field.Team[SecondID].CurrPlayer].mouseX = dX
+ dWidth + 0.001;

                                if ((Field.CalcDistance(Ball->Width/2
+ Ball->Position->X, Ball->Height/2 + Ball->Position->Y,
Field.Team[MainID].Player[Field.Team[MainID].CurrPlayer].Img->Position->X +
Field.Team[MainID].Player[Field.Team[MainID].CurrPlayer].Img->Width/2,
Field.Team[MainID].Player[Field.Team[MainID].CurrPlayer].Img->Position->Y +
Field.Team[MainID].Player[Field.Team[MainID].CurrPlayer].Img->Height/2) <=
Field.Team[MainID].Player[Field.Team[MainID].CurrPlayer].Img->Height)) {
                                    Field.Ball.Speed += 3;
                                    Field.Ball.Degree =
Field.CalcDegree(Field.Team[MainID].Player[Field.Team[MainID].CurrPlayer].Img-
>Width/2 + Field.Team[MainID].Player[Field.Team[MainID].CurrPlayer].Img->Position-
>X, Field.Team[MainID].Player[Field.Team[MainID].CurrPlayer].Img->Height/2 +
Field.Team[MainID].Player[Field.Team[MainID].CurrPlayer].Img->Position->Y, Ball-
>Width/2 + Ball->Position->X, Ball->Height/2 + Ball->Position->Y);
                                    Ball->Position->X += 5;
                                }

                            break;

                case 'e': case 'E':

                    Field.Team[SecondID].Player[Field.Team[SecondID].CurrPlayer].mouseX = dX
+ dWidth + 0.001;

                                if ((Field.CalcDistance(Ball->Width/2
+ Ball->Position->X, Ball->Height/2 + Ball->Position->Y,
Field.Team[MainID].Player[Field.Team[MainID].CurrPlayer].Img->Position->X +
Field.Team[MainID].Player[Field.Team[MainID].CurrPlayer].Img->Width/2,
Field.Team[MainID].Player[Field.Team[MainID].CurrPlayer].Img->Position->Y +
Field.Team[MainID].Player[Field.Team[MainID].CurrPlayer].Img->Height/2) <=
Field.Team[MainID].Player[Field.Team[MainID].CurrPlayer].Img->Height)) {
                                    Field.Ball.Speed += 3;
                                    Field.Ball.Degree =
Field.CalcDegree(Field.Team[MainID].Player[Field.Team[MainID].CurrPlayer].Img-
>Width/2 + Field.Team[MainID].Player[Field.Team[MainID].CurrPlayer].Img->Position-
>X, Field.Team[MainID].Player[Field.Team[MainID].CurrPlayer].Img->Height/2 +
Field.Team[MainID].Player[Field.Team[MainID].CurrPlayer].Img->Position->Y,
((54)*PanelPlayArea->Width/1062), ((274)*PanelPlayArea->Height/545));
                                    Ball->Position->X += 5;
                                }

                            break;

            }
        }
// Обработка переключений между игроками
    } else if (isdigit(KeyChar)) {
        KeyChar -= '0';
        switch(KeyChar) {
            case 1: case 2: case 3:

```

```

        Field.Team[MainID].Player[Field.Team[MainID].CurrPlayer].Speed =
DefSpeed[Field.Team[MainID].CurrPlayer]/2;
        Field.Team[MainID].CurrPlayer =
KeyChar - 1;
        break;
    }
}

//-----
void __fastcall TFormMain::PanelPlayAreaMouseMove(TObject *Sender, TShiftState
Shift, float X,
        float Y)
{
    Field.Team[MainID].Player[Field.Team[MainID].CurrPlayer].mouseX = X;
    Field.Team[MainID].Player[Field.Team[MainID].CurrPlayer].mouseY = Y;
    Field.Team[MainID].Player[Field.Team[MainID].CurrPlayer].Speed =
DefSpeed[Field.Team[MainID].CurrPlayer];
}

//-----
void __fastcall TFormMain::TimerCheckBallTimer(TObject *Sender)
{
    // Проверка на ГОЛ
    double BallX = Field.Ball.CBall->Width/2 + Field.Ball.CBall->Position->X;
    double BallY = Field.Ball.CBall->Height/2 + Field.Ball.CBall->Position-
>Y;
    int GT = 0;
    if ((BallX < (54*PanelPlayArea->Width/1062)) && (BallY >
(236*PanelPlayArea->Height/545)) && (BallY < (311*PanelPlayArea->Height/545)))
    {
        GT = 1;
    } else if ((BallX > ((1062 - 54)*PanelPlayArea->Width/1062)) && (BallY >
(236*PanelPlayArea->Height/545)) && (BallY < (311*PanelPlayArea->Height/545)))
    {
        GT = 2;
    }
    if (GT > 0)
    {
        Game.GoalTo(GT);
        PrintCount();
        DestroyField();
        if (Game.Winner > -1) {
            ofstream ss("db/win.bin", std::ofstream::out |
std::ofstream::trunc);
            if (Game.Winner == 0)
            {
                PlayMusic("loose.mp3");
                string msg = "Вы проиграли";
                ShowMessage(msg.c_str());
                ss << 0;
            } else {
                PlayMusic("win.mp3");
                string msg = "Вы ПОБЕДИЛИ! На ваш
счет будет зачислено $5000.";
                ShowMessage(msg.c_str());
                ss << 5000;
            }
            ss.close();
            FormMain->Close();
        } else {

```



```

        PlayMusic("goal.wav");
        InitField();
    }

    Field.Ball.Speed = 0;
    Field.Ball.Degree = 0;
    return;
}

// Проверка на выход из поля
if ((Field.Ball.CBall->Width/2 + Field.Ball.CBall->Position->X) <
(54*PanelPlayArea->Width/1062))
{
    PlayMusic("svist.mp3");
    InitTimer(0);
    Field.Ball.CBall->Position->X = BallDelta[0]*PanelPlayArea-
>Width - Field.Ball.CBall->Width/2;
    Field.Ball.CBall->Position->Y = PanelPlayArea->Height/2 -
Field.Ball.CBall->Height/2;
    Field.Ball.Speed = 0;
    Field.Ball.Degree = 0;
    TimerMoveUser->Enabled = false;
    Field.Team[0].DeletePlayer(0);
    Field.Team[0].AddPlayer((GoalKeeperDelta[0]*PanelPlayArea-
>Width) - PlayerSize/2, (PanelPlayArea->Height - PlayerSize)/2, 'b');
    TimerMoveUser->Enabled = true;
    return;
} else if ((Field.Ball.CBall->Width/2 + Field.Ball.CBall->Position->X) >
((1062-54)*PanelPlayArea->Width/1062))
{
    PlayMusic("svist.mp3");
    InitTimer(0);
    Field.Ball.CBall->Position->X = BallDelta[1]*PanelPlayArea-
>Width - Field.Ball.CBall->Width/2;
    Field.Ball.CBall->Position->Y = PanelPlayArea->Height/2 -
Field.Ball.CBall->Height/2;
    Field.Ball.Speed = 0;
    Field.Ball.Degree = 0;
    TimerMoveUser->Enabled = false;
    Field.Team[1].DeletePlayer(0);
    Field.Team[1].AddPlayer((GoalKeeperDelta[1]*PanelPlayArea-
>Width) - PlayerSize/2, (PanelPlayArea->Height - PlayerSize)/2, 'o');
    TimerMoveUser->Enabled = true;
    return;
}

// Проверка на столкновение с игроком
for (int i = 1; i < 2; i++)
{
    TImage* TestPlayer;
    for (int j = 0; j < 3; j++) {
        TestPlayer = Field.Team[i].Player[j].Img;
        if (sqrt(((Ball->Height/2 + Ball->Position->Y) -
(TestPlayer->Height/2 + TestPlayer->Position->Y))*((Ball->Height/2 + Ball->Position-
>Y) - (TestPlayer->Height/2 + TestPlayer->Position->Y)) + ((Ball->Width/2 + Ball-
>Position->X) - (TestPlayer->Width/2 + TestPlayer->Position->X))*((Ball->Width/2 +
Ball->Position->X) - (TestPlayer->Width/2 + TestPlayer->Position->X)))) <=
TestPlayer->Width/4/(2-i)) {
            Field.Ball.Speed = 0;
            return;
        }
    }
}
}

```

```

}
//-----

void __fastcall TFormMain::FormShow(TObject *Sender)
{
    Udb.in();
    Udb.pid.in();
    for (int i = 0; i < 3; i++)
        DefSpeed[i] = Udb.GetUser(Udb.pid.value).Team[i].speed;
    LabelGameTo->Text = ("Игра до " + ToStr(Game.WinCount) + "
побед.").c_str();
    Udb.pid.out();
    Udb.out();
    InitField();
    Game.InitGame();
}
//-----

void __fastcall TFormMain::FormCloseQuery(TObject *Sender, bool &CanClose)
{
    if (Game.Winner == -1) {
        ofstream ss("db/win.bin", std::ofstream::out |
std::ofstream::trunc);
        ss << 0;
        ss.close();
    }
    DestroyField();
}
//-----

void __fastcall TFormMain::TimerCountDownTimer(TObject *Sender)
{
    if (!Game.TimerCount) {
        LabelCountDown->Text = "";
        TimerCountDown->Enabled = false;
        PrintCount();
        Game.CanMove = true;
    } else {
        LabelCountDown->Text = ("Таймер: " + ToStr(Game.TimerCount--)
+ " секунд").c_str();
    }
}
//-----

```

ПРИЛОЖЕНИЕ В

(обязательное)

Текст модуля UnitMenu

```
//UnitMenu.h

#ifndef UnitMenuH
#define UnitMenuH
//-----
#include <System.Classes.hpp>
#include <FMX.Controls.hpp>
#include <FMX.Forms.hpp>
#include <FMX.Colors.hpp>
#include <FMX.Controls.Presentation.hpp>
#include <FMX.StdCtrls.hpp>
#include <FMX.Types.hpp>
#include <FMX.Objects.hpp>
#include <FMX.ImgList.hpp>
#include <System.ImageList.hpp>
#include <FMX.Ani.hpp>
//-----
#include "ClassTUdb.h"
#include <FMX.Media.hpp>
#include <IdBaseComponent.hpp>
#include <IdComponent.hpp>
#include <IdSntp.hpp>
#include <IdUDPBase.hpp>
#include <IdUDPClient.hpp>
//-----
class TFormMenu : public TForm
{
__published:      // IDE-managed Components
    TLabel *LabelUser;
    TToolBar *ToolBarPlayers;
    TPanel *PanelPlayer0;
    TPanel *PanelPlayer1;
    TPanel *PanelPlayer2;
    TTimer *TimerCheckUserLogin;
    TLabel *LabelName0;
    TLabel *LabelLevel0;
    TLabel *LabelSpeed0;
    TImage *ImageUser0;
    TLabel *LabelSpeed1;
    TLabel *LabelLevel1;
    TLabel *LabelName1;
    TImage *ImageUser1;
    TLabel *LabelSpeed2;
    TLabel *LabelLevel2;
    TLabel *LabelName2;
    TImage *ImageUser2;
    TImage *ImageBackground;
    TToolBar *ToolBarTop;
    TLabel *LabelCash;
    TLabel *LabelLvl;
    TLabel *LabelWin;
    TSpeedButton *SpeedButtonProfile;
    TFloatAnimation *FloatAnimationPanelHeight;
    TImage *ImageProfile;
    TSpeedButton *SpeedButtonAbout;
    TImage *ImageAbout;
    TSpeedButton *SpeedButtonStats;
```

```

TImage *ImageStatistics;
TFloatAnimation *FloatAnimationMouseProfile;
TFloatAnimation *FloatAnimationMouseStat;
TFloatAnimation *FloatAnimationMouseInfo;
TSpeedButton *SpeedButtonStartGame;
TImage *ImageStartGame;
TSpeedButton *SpeedButtonAdd0;
TImage *ImageAdd0;
TSpeedButton *SpeedButtonAdd1;
TImage *ImageAdd1;
TSpeedButton *SpeedButtonAdd2;
TImage *ImageAdd2;
TSpeedButton *SpeedButtonExit;
TImage *ImageExit;
TSpeedButton *SpeedButtonAddLvl;
TImage *ImageAddLvl;
TCalloutPanel *CalloutPanelToReg;
TLabel *LabelNeedRegister;
TFloatAnimation *FloatAnimationPanelTopHeight;
TFloatAnimation *FloatAnimationStartGameHeight;
TFloatAnimation *FloatAnimationUserNameHeight;
TTimer *TimerChangePhoto;
TProgressBar *ProgressBarPhoto;
TFloatAnimation *FloatAnimationChangePhoto;
TFloatAnimation *FloatAnimationAddLvl;
TMediaPlayer *MediaPlayerMenu;
TIdSNTP *IdSNTP1;
TTimer *TimerCheckLicense;
void __fastcall ButtonProfileClick(TObject *Sender);
void __fastcall TimerCheckUserLoginTimer(TObject *Sender);
void __fastcall FormCreate(TObject *Sender);
void __fastcall FormCloseQuery(TObject *Sender, bool &CanClose);
void __fastcall FormResize(TObject *Sender);
void __fastcall SpeedButtonProfileClick(TObject *Sender);
void __fastcall SpeedButtonAboutClick(TObject *Sender);
void __fastcall SpeedButtonStatsClick(TObject *Sender);
void __fastcall SpeedButtonProfileMouseEnter(TObject *Sender);
void __fastcall SpeedButtonStatsMouseEnter(TObject *Sender);
void __fastcall SpeedButtonAboutMouseEnter(TObject *Sender);
void __fastcall SpeedButtonStartGameClick(TObject *Sender);
void __fastcall SpeedButtonAdd0Click(TObject *Sender);
void __fastcall SpeedButtonAdd1Click(TObject *Sender);
void __fastcall SpeedButtonAdd2Click(TObject *Sender);
void __fastcall SpeedButtonExitClick(TObject *Sender);
void __fastcall SpeedButtonAddLvlClick(TObject *Sender);
void __fastcall TimerChangePhotoTimer(TObject *Sender);
void __fastcall SpeedButtonAddLvlMouseEnter(TObject *Sender);
void __fastcall TimerCheckLicenseTimer(TObject *Sender);
private:    // User declarations
public:    // User declarations
    void __fastcall SetUserVisible(bool state);
    void __fastcall TFormMenu(TComponent* Owner);
};
//-----
extern PACKAGE TFormMenu *FormMenu;
//-----
#endif

//UnitMenu.cpp

#include <fmh.h>
#pragma hdrstop

```

```

#include "UnitMenu.h"
#include "UnitAuth.h"
#include "UnitMain.h"
#include "UnitStat.h"
#include "UnitInfo.h"
//-----
#pragma package(smart_init)
#pragma resource "*.fmx"
#pragma resource ("*.Windows.fmx", _PLAT_MSWINDOWS)
#pragma resource ("*.Surface.fmx", _PLAT_MSWINDOWS)
//-----
TFormMenu *FormMenu;
TDateTime ExpireDate("01.12.2016 15:09:00");
TUdb Udb;
int AmountOfImages = 4;
int NumOfImage = 0;
int x = 5;
//-----
__fastcall TFormMenu::TFormMenu(TComponent* Owner)
    : TForm(Owner)
{
}
//-----

void __fastcall TFormMenu::ButtonProfileClick(TObject *Sender)
{
    FormAuth->ShowModal();
    Udb.in();
    Udb.pid.in();
}
//-----

void __fastcall TFormMenu::SetUserVisible(bool state)
{
    ImageUser0->Visible = state;
    ImageUser1->Visible = state;
    ImageUser2->Visible = state;
    LabelName0->Visible = state;
    LabelSpeed0->Visible = state;
    LabelLevel0->Visible = state;
    LabelName1->Visible = state;
    LabelSpeed1->Visible = state;
    LabelLevel1->Visible = state;
    LabelName2->Visible = state;
    LabelSpeed2->Visible = state;
    LabelLevel2->Visible = state;
    SpeedButtonAddLvl->Visible = state;
    SpeedButtonAdd0->Visible = state;
    SpeedButtonAdd1->Visible = state;
    SpeedButtonExit->Visible = state;
    SpeedButtonAdd2->Visible = state;
    CalloutPanelToReg->Visible = !state;
    if (state) {
        FloatAnimationPanelHeight->StopValue = 89;
        FloatAnimationPanelTopHeight->StopValue = 49;
        FloatAnimationStartGameHeight->StopValue = 49;
        FloatAnimationUserNameHeight->StopValue = 25;
    } else {
        FloatAnimationPanelHeight->StopValue = 0;
        FloatAnimationPanelTopHeight->StopValue = 0;
        FloatAnimationStartGameHeight->StopValue = 0;
        FloatAnimationUserNameHeight->StopValue = 0;
    }
}

```

```

    }
    FloatAnimationPanelHeight->Start();
    FloatAnimationPanelTopHeight->Start();
    FloatAnimationStartGameHeight->Start();
    FloatAnimationUserNameHeight->Start();
}

void __fastcall TFormMenu::TimerCheckUserLoginTimer(TObject *Sender)
{
    if (Udb.pid.value > -1)
    {
        LabelUser->Text = ("Добро пожаловать, " +
Udb.GetUser(Udb.pid.value).name).c_str();
        LabelCash->Text = ("Деньги: $" +
ToStr(Udb.GetUser(Udb.pid.value).cash)).c_str();
        LabelLvl->Text = ("Уровень: " +
ToStr(Udb.GetUser(Udb.pid.value).level)).c_str();
        LabelWin->Text = ("Побед: " +
ToStr(Udb.GetUser(Udb.pid.value).wins)).c_str();

        for (int i = 0; i < 3; i++)
        {
            TImage* Img;
            TLabel* LabelName;
            TLabel* LabelSpeed;
            TLabel* LabelLevel;
            switch(i)
            {
                case 0:
                    Img = ImageUser0;
                    LabelName = LabelName0;
                    LabelSpeed = LabelSpeed0;
                    LabelLevel = LabelLevel0;

                    break;
                case 1:
                    Img = ImageUser1;
                    LabelName = LabelName1;
                    LabelSpeed = LabelSpeed1;
                    LabelLevel = LabelLevel1;

                    break;
                case 2:
                    Img = ImageUser2;
                    LabelName = LabelName2;
                    LabelSpeed = LabelSpeed2;
                    LabelLevel = LabelLevel2;

                    break;
            }
            Img->Bitmap->LoadFromFile("img/user.png");
            LabelName->Text =
(Udb.GetUser(Udb.pid.value).Team[i].name + " " +
Udb.GetUser(Udb.pid.value).Team[i].surname).c_str();
            LabelSpeed->Text = ("Скорость: " +
ToStr(Udb.GetUser(Udb.pid.value).Team[i].speed)).c_str();
            LabelLevel->Text = ("Уровень: " +
ToStr(Udb.GetUser(Udb.pid.value).Team[i].level)).c_str();
        }
        SetUserVisible(true);
    } else {
        LabelUser->Text = "Войдите или зарегистрируйтесь";
        LabelCash->Text = "";
        LabelLvl->Text = "";
        LabelWin->Text = "";
        SetUserVisible(false);
    }
}

```

```

    }
}
//-----

void __fastcall TFormMenu::FormCreate(TObject *Sender)
{
    MediaPlayerMenu->Play();
    ImageBackground->Bitmap->LoadFromFile(("img/menu/" + ToString(NumOfImage++)
+ ".jpg").c_str());
    PanelPlayer0->Width = ToolBarPlayers->Width/3;
    PanelPlayer1->Width = ToolBarPlayers->Width/3;
    PanelPlayer2->Width = ToolBarPlayers->Width/3;

    Udb.in();
    Udb.pid.in();
    Udb.pid.value = -1;
    Udb.pid.out();
    ToolBarPlayers->Height = 0;
    ToolBarTop->Height = 0;
}
//-----

void __fastcall TFormMenu::FormCloseQuery(TObject *Sender, bool &CanClose)
{
    Udb.out();
    CanClose = true;
}
//-----

void __fastcall TFormMenu::FormResize(TObject *Sender)
{
    PanelPlayer0->Width = ToolBarPlayers->Width/3;
    PanelPlayer1->Width = ToolBarPlayers->Width/3;
    PanelPlayer2->Width = ToolBarPlayers->Width/3;
}
//-----

void __fastcall TFormMenu::SpeedButtonProfileClick(TObject *Sender)
{
    FormAuth->ShowModal();
    Udb.in();
    Udb.pid.in();
}
//-----

void __fastcall TFormMenu::SpeedButtonAboutClick(TObject *Sender)
{
    FormMenu->Hide();
    FormInfo->ShowModal();
    FormMenu->Show();
}
//-----

void __fastcall TFormMenu::SpeedButtonStatsClick(TObject *Sender)
{
    if (Udb.pid.value == -1)
    {
        string msg = "Для просмотра статистики нужно
зарегистрироваться.";
        ShowMessage(msg.c_str());
        return;
    }
}

```

```

    }
    Udb.pid.out();
    Udb.out();
    Udb.in();
    Udb.pid.in();
    FormMenu->Hide();
    FormStat->ShowModal();
    FormMenu->Show();
}
//-----

void __fastcall TFormMenu::SpeedButtonProfileMouseEnter(TObject *Sender)
{
    if (!FloatAnimationMouseProfile->Running)
        FloatAnimationMouseProfile->Start();
}
//-----

void __fastcall TFormMenu::SpeedButtonStatsMouseEnter(TObject *Sender)
{
    if (!FloatAnimationMouseStat->Running)
        FloatAnimationMouseStat->Start();
}
//-----

void __fastcall TFormMenu::SpeedButtonAboutMouseEnter(TObject *Sender)
{
    if (!FloatAnimationMouseInfo->Running)
        FloatAnimationMouseInfo->Start();
}
//-----

void __fastcall TFormMenu::SpeedButtonStartGameClick(TObject *Sender)
{
    if (Udb.pid.value == -1)
    {
        string msg = "Перед началом игры нужно зарегистрироваться.";
        ShowMessage(msg.c_str());
        return;
    }
    Udb.pid.out();
    Udb.out();
    FormMenu->Hide();
    FormMain->ShowModal();
    FormMenu->Show();
    Udb.in();
    Udb.pid.in();
    int MoneyToAdd;
    ifstream ss("db/win.bin");
    ss >> MoneyToAdd;
    Udb.AddWin(Udb.pid.value, MoneyToAdd);
    ss.close();
}
//-----

void __fastcall TFormMenu::SpeedButtonAdd0Click(TObject *Sender)
{
    if (!Udb.BuyLvl(Udb.pid.value, 0))
    {
        string msg = "Недостаточно средств на счету.";
        ShowMessage(msg.c_str());
    }
}

```



```

//-----
void __fastcall TFormMenu::SpeedButtonAdd1Click(TObject *Sender)
{
    if (!Udb.BuyLvl(Udb.pid.value, 1))
    {
        string msg = "Недостаточно средств на счету.";
        ShowMessage(msg.c_str());
    }
}
//-----

void __fastcall TFormMenu::SpeedButtonAdd2Click(TObject *Sender)
{
    if (!Udb.BuyLvl(Udb.pid.value, 2))
    {
        string msg = "Недостаточно средств на счету.";
        ShowMessage(msg.c_str());
    }
}
//-----

void __fastcall TFormMenu::SpeedButtonExitClick(TObject *Sender)
{
    Udb.pid.value = -1;
    Udb.pid.out();
    Udb.pid.in();
}
//-----

void __fastcall TFormMenu::SpeedButtonAddLvlClick(TObject *Sender)
{
    if (!Udb.BuyAccountLvl(Udb.pid.value))
    {
        string msg = "Недостаточно средств на счету.";
        ShowMessage(msg.c_str());
    }
}
//-----

void __fastcall TFormMenu::TimerChangePhotoTimer(TObject *Sender)
{
    ImageBackground->Bitmap->LoadFromFile(("img/menu/" + ToString(NumOfImage) +
".jpg").c_str());
    NumOfImage = (NumOfImage + 1) % AmountOfImages;
}
//-----

void __fastcall TFormMenu::SpeedButtonAddLvlMouseEnter(TObject *Sender)
{
    if (!FloatAnimationAddLvl->Running)
        FloatAnimationAddLvl->Start();
}
//-----

void __fastcall TFormMenu::TimerCheckLicenseTimer(TObject *Sender)
{
    IdSNTP1->SyncTime();
    if (IdSNTP1->DateTime > ExpireDate)
    {
        throw Exception("License expried.");
    }
}

```

ПРИЛОЖЕНИЕ Г
(обязательное)
Текст модуля UnitStat

```
//-----
#ifndef UnitStatH
#define UnitStatH
//-----
#include <System.Classes.hpp>
#include <FMX.Controls.hpp>
#include <FMX.Forms.hpp>
#include <FMX.Controls.Presentation.hpp>
#include <FMX.Layouts.hpp>
#include <FMX.ListBox.hpp>
#include <FMX.StdCtrls.hpp>
#include <FMX.Types.hpp>
#include <FMX.ListView.Adapters.Base.hpp>
#include <FMX.ListView.Appearances.hpp>
#include <FMX.ListView.hpp>
#include <FMX.ListView.Types.hpp>
//-----
#include "ClassTUdb.h"
//-----
class TFormStat : public TForm
{
__published:      // IDE-managed Components
    TListBox *ListBox1;
    TLabel *LabelStat;
    TToolBar *ToolBar1;
    void __fastcall FormShow(TObject *Sender);
private:          // User declarations
public:            // User declarations
    __fastcall TFormStat(TComponent* Owner);
};
//-----
extern PACKAGE TFormStat *FormStat;
//-----
#endif
//-----

#include <fmx.h>
#pragma hdrstop

#include "UnitStat.h"
//-----
#pragma package(smart_init)
#pragma resource "*.fmx"
TFormStat *FormStat;
TUdb Udb;
//-----
__fastcall TFormStat::TFormStat(TComponent* Owner)
    : TForm(Owner)
{
}
//-----

void __fastcall TFormStat::FormShow(TObject *Sender)
{
    Udb.in();
    Udb.pid.in();
}
```

```

k = 1;
ListBox1->Items->Clear();

string CurrentUser = Udb.GetUser(Udb.pid.value).name;
Udb.SortForStat();
ListBox1->Columns = 6;

ListBox1->Items->Add("Место");
ListBox1->Items->Add("Игрок");
ListBox1->Items->Add("Уровень");
ListBox1->Items->Add("Деньги");
ListBox1->Items->Add("Уровень команды");
ListBox1->Items->Add("Победы");

for (int i = 0; i < Udb.length(); i++)
{
    ListBox1->Items->Add(("#" + ToString(k++)).c_str());
    if (Udb.GetUser(i).name == CurrentUser) {
        ListBox1->Items->Add(("> " +
Udb.GetUser(i).name).c_str());
    } else {
        ListBox1->Items-
>Add(Udb.GetUser(i).name.c_str());
    }
    ListBox1->Items->Add(ToString(Udb.GetUser(i).level).c_str());
    ListBox1->Items->Add((" $" +
ToString(Udb.GetUser(i).cash)).c_str());
    ListBox1->Items->Add(ToString(Udb.GetUser(i).Team[0].level +
Udb.GetUser(i).Team[1].level + Udb.GetUser(i).Team[2].level).c_str());
    ListBox1->Items->Add(ToString(Udb.GetUser(i).wins).c_str());
}
}
//-----

```

ПРИЛОЖЕНИЕ Д (обязательное) Текст модуля UnitInfo

```
//-----  
  
#ifndef UnitInfoH  
#define UnitInfoH  
//-----  
#include <System.Classes.hpp>  
#include <FMX.Controls.hpp>  
#include <FMX.Forms.hpp>  
#include <FMX.Controls.Presentation.hpp>  
#include <FMX.Memo.hpp>  
#include <FMX.ScrollBox.hpp>  
#include <FMX.StdCtrls.hpp>  
#include <FMX.Types.hpp>  
//-----  
class TFormInfo : public TForm  
{  
    __published:          // IDE-managed Components  
        TToolBar *ToolBar1;  
        TLabel *LabelInfo;  
        TMemo *MemoInfo;  
private:                // User declarations  
public:                 // User declarations  
    __fastcall TFormInfo(TComponent* Owner);  
};  
//-----  
extern PACKAGE TFormInfo *FormInfo;  
//-----  
#endif  
//-----  
  
#include <fmx.h>  
#pragma hdrstop  
  
#include "UnitInfo.h"  
//-----  
#pragma package(smart_init)  
#pragma resource "*.fmx"  
TFormInfo *FormInfo;  
//-----  
__fastcall TFormInfo::TFormInfo(TComponent* Owner)  
    : TForm(Owner)  
{  
}  
//-----
```

ПРИЛОЖЕНИЕ Е

(обязательное)

Текст модуля классов

```
class TProfile {
public:
    TPlayer Team[3];
    string name;
    int level;
    int cash;
    int wins;
    void write(ofstream& ss);
    void read(ifstream& ss);
};
#include "ClassTProfile.h"
void TProfile::write(ofstream& ss)
{
    ss << name << " " << level << " " << cash << " " << wins << " ";
    for (int i = 0; i < 3; i++) {
        ss << Team[i].name << " " << Team[i].surname << " " <<
Team[i].level << " " << Team[i].speed;
        if (i < 2) ss << " ";
    }
}

void TProfile::read(ifstream& ss)
{
    string str;
    string tmp;
    ss >> name >> level >> cash >> wins;
    for (int i = 0; i < 3; i++)
        ss >> Team[i].name >> Team[i].surname >> Team[i].level >>
Team[i].speed;
}

class TPlayer {
public:
    TPlayer();
    string name;
    string surname;
    int level;
    int speed;
};
TPlayer::TPlayer()
{
    name = "";
    surname = "";
    level = 0;
    speed = 0;
}

class TUdb {
private:
    vector<TProfile> user;
public:
    TUdb();
    TPid pid;
    string filename;
    void out();
    void in();
    void SortForStat();
    bool IsExist(string name);
};
```

```

        bool reg(TProfile prfl);
        int log(string name);
        TProfile GetUser(int PID);
        void AddWin(int PID, int Funds);
        bool BuyLvl(int PID, int PlayerID);
        bool BuyAccountLvl(int PID);
        int length();
};
#include "ClassTUdb.h"
TUdb::TUdb()
{
    filename = "db/db.bin";
}

void TUdb::out()
{
    ofstream ss(filename.c_str(), std::ofstream::out | std::ofstream::trunc);
    for (int i = 0; i < user.size(); i++)
    {
        user[i].write(ss);
        if (i != user.size() - 1) ss << endl;
    }
    user.clear();
    ss.close();
}

void TUdb::in()
{
    ifstream ss(filename.c_str());
    TProfile tmp;
    user.clear();
    while (!ss.eof())
    {
        tmp.read(ss);
        user.push_back(tmp);
    }
    ss.close();
}

void TUdb::SortForStat()
{
    TProfile tmp;

    for (int i = 0; i < user.size(); i++)
    {
        for (int j = i + 1; j < user.size(); j++)
        {
            if (user[j].level > user[i].level) {
                tmp = user[i];
                user[i] = user[j];
                user[j] = tmp;
            }
        }
    }

    for (int i = 0; i < user.size(); i++)
    {
        for (int j = i + 1; j < user.size(); j++)
        {
            if (user[j].wins > user[i].wins) {
                tmp = user[i];
                user[i] = user[j];
                user[j] = tmp;
            }
        }
    }
}

```

```

        }
    }
}

bool TUdb::IsExist(string name)
{
    for (int i = 0; i < user.size(); i++)
        if (user[i].name == name)
            return true;

    return false;
}

bool TUdb::reg(TProfile prfl)
{
    if (!IsExist(prfl.name))
    {
        user.push_back(prfl);
        return true;
    } else {
        return false;
    }
}

int TUdb::log(string name)
{
    for (int i = 0; i < user.size(); i++)
        if (user[i].name == name)
            return i;

    return -1;
}

int TUdb::length()
{
    return user.size();
}

TProfile TUdb::GetUser(int PID)
{
    return user[PID];
}

void TUdb::AddWin(int PID, int Funds)
{
    if (Funds > 0) {
        user[PID].cash += Funds;
        user[PID].wins++;
    }
}

bool TUdb::BuyLvl(int PID, int PlayerID)
{
    if (user[PID].cash >= 20000) {
        user[PID].cash -= 20000;
        user[PID].Team[PlayerID].speed++;
        user[PID].Team[PlayerID].level =
user[PID].Team[PlayerID].speed/3;
        return true;
    }
    return false;
}

bool TUdb::BuyAccountLvl(int PID)

```

```

{
    if (user[PID].cash >= 15000) {
        user[PID].cash -= 15000;
        user[PID].level++;
        return true;
    }
    return false;
}

class TPid {
public:
    int value;
    TPid();
    string filename;
    void out();
    void in();
};
#include "ClassTPid.h"
TPid::TPid()
{
    filename = "db/pid.bin";
}

void TPid::out()
{
    ofstream ss(filename.c_str(), std::ofstream::out | std::ofstream::trunc);
    ss << value;
    ss.close();
}

void TPid::in()
{
    ifstream ss(filename.c_str());
    ss >> value;
    ss.close();
}

class TGame {
public:
    void InitGame();
    TGame();
    bool CanMove;
    int WinCount; // Количество игр, которые необходимо выиграть
    int Count[2]; // Счет каждой команды
    int TimerCount;
    int Winner; // Кто выиграл матч
    void GoalTo(int Team); // Кто забил мяч = 1/2
};
TGame::TGame()
{
    InitGame();
}

void TGame::InitGame()
{
    CanMove = false;
    Winner = -1;
    WinCount = 3;
    TimerCount = 0;
    Count[0] = Count[1] = 0;
}

void TGame::GoalTo(int Team)

```



```

{
    if (Team == 1 || Team == 2)
        Count[2 - Team]++;
    if (WinCount == Count[0] || WinCount == Count[1])
        (WinCount == Count[0]) ? (Winner = 0) : (Winner = 1);
}

class TFiPlayer {
public:
    int Speed;
    float mouseX, mouseY;
    bool InGame;
    TImage* Img;
    TText* Txt;
};

class TTeam {
public:
    TFiPlayer Player[TeamSize];
    int CurrPlayer;
    int GetAmount();
    bool AddPlayer(float PosX, float PosY, char ImgColor);
    bool DeletePlayer(int N);
};
#include "ClassTTeam.h"
int TTeam::GetAmount()
{
    int Amount = 0;
    for (int i = 0; i < TeamSize; i++)
        if (Player[i].InGame)
            Amount++;
    return Amount;
}

bool TTeam::AddPlayer(float PosX, float PosY, char ImgColor)
{
    if (GetAmount() < TeamSize)
    {
        int i = 0;
        while (Player[i].InGame) i++;
        Player[i].Speed = -1;
        Player[i].mouseX = 0;
        Player[i].mouseY = 0;
        Player[i].InGame = true;

        // Создаем картинку игрока
        Player[i].Img = new TImage(FormMain);
        Player[i].Img->Parent = FormMain->ImageBackground;
        Player[i].Img->Name = "Image" + AnsiString(i) + "Team" +
++UniqNum;

        Player[i].Img->Width = PlayerSize;
        Player[i].Img->Height = PlayerSize;
        Player[i].Img->Position->X = PosX;
        Player[i].Img->Position->Y = PosY;
        switch(ImgColor)
        {
            case 'b':
                Player[i].Img->Bitmap-
>LoadFromFile("img/player0.png");
                break;
            case 'o':
                Player[i].Img->Bitmap-
>LoadFromFile("img/player1.png");

```

```

                                break;
                                default:
                                Player[i].Img->Bitmap-
>LoadFromFile("img/player.png");
                                break;
                                }

                                // Номер игрока, поверх картинки
                                Player[i].Txt = new TText(FormMain);
                                Player[i].Txt->Parent = FormMain->ImageBackground;
                                Player[i].Txt->Height = PlayerSize;
                                Player[i].Txt->Width = PlayerSize;
                                Player[i].Txt->Name = "Text" + AnsiString(i) + "Team" +
++UniqNum;

                                Player[i].Txt->Text = AnsiString(i + 1);
                                Player[i].Txt->Font->Size = 36;
                                Player[i].Txt->TextSettings->FontColor =
TAlphaColor(claBlack);

                                Player[i].Txt->Position->X = PosX;
                                Player[i].Txt->Position->Y = PosY;
                                return true;
                                } else {
                                return false;
                                }
                                }

bool TTeam::DeletePlayer(int N)
{
    if (Player[N].InGame)
    {
        Player[N].Speed = 0;
        Player[N].mouseX = 0;
        Player[N].mouseY = 0;
        Player[N].InGame = false;
        delete Player[N].Img;
        delete Player[N].Txt;
        return true;
    } else {
        return false;
    }
}

class TBall {
public:
    TBall();

    float Speed, Accelerate, Degree;
    TCircle* CBall;
};
#include "ClassTBall.h"
TBall::TBall()
{
    Degree = 0;
    Accelerate = 0.002;
    Speed = 0;
}

const double pi = 3.14159265358979323;

class TField {
public:
    TTeam Team[2];
    TBall Ball;
    TImage* Background;

```

```

float CalcDegree(float FromX, float FromY, float ToX, float
ToY);
float CalcDistance(float FromX, float FromY, float ToX, float
ToY);
};

#include "ClassTField.h"
float TField::CalcDegree(float FromX, float FromY, float ToX, float ToY)
{
    if ((ToX - (FromX) > 0) && (ToY - (FromY) > 0)) { // X+ Y+
        return ceil(atan(fabs(ToX - (FromX))/fabs(ToY - (FromY))) *
180 / pi) + 270;
    } else if ((ToX - (FromX) < 0) && (ToY - (FromY) > 0)) { // X- Y+
        return ceil(atan(fabs(ToY - (FromY))/fabs(ToX - (FromX))) *
180 / pi) + 180;
    } else if ((ToX - (FromX) > 0) && (ToY - (FromY) < 0)) { // X+ Y-
        return ceil(atan(fabs(ToY - (FromY))/fabs(ToX - (FromX))) *
180 / pi);
    } else if ((ToX - (FromX) < 0) && (ToY - (FromY) < 0)) { // X- Y-
        return 180 - ceil(atan(fabs(ToY - (FromY))/fabs(ToX -
(FromX))) * 180 / pi);
    } else if ((ToX - (FromX) == 0) && (ToY - (FromY) < 0)) { // X- Y+
        return 180;
    } else {
        return 0;
    }
}

float TField::CalcDistance(float FromX, float FromY, float ToX, float ToY)
{
    return sqrt(((FromY) - (ToY))*((FromY) - (ToY)) + ((FromX) -
(ToX))*((FromX) - (ToX)));
}

```

Обозначение					Наименование					Дополнительные сведения				
					<u>Текстовые документы</u>									
БГУИР КП 1–40 01 01 551005 014 ПЗ					Пояснительная записка					60 с.				
					<u>Графические документы</u>									
ГУИР 551005 014 ПД					Схема программы					Формат А1				