

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования
«Ижевский государственный технический университет
имени М.Т. Калашникова»

К защите

Руководитель направления
Лялин В.Е.

«_____» _____ 20__ г.

Кошелева Юлия Святославовна

Разработка системы поддержки проведения соревнований по спортивному
бриджу

09.04.04 «Программная инженерия»

09.04.04-2 «Системы мультимедиа и компьютерная графика»

**Диссертация на соискание академической степени
магистра**

Магистрант
Кошелева Ю.С.

Научный руководитель
к.т.н., доцент Коробейников А.В.

Руководитель программы
д.т.н., профессор Мурынов А.И

Ижевск 2015

РЕФЕРАТ

Объем и структура диссертационной работы. Диссертация содержит введение, 6 глав и заключение, изложенные на 168 с. машинописного текста, а также 1 приложения. В работу включены 22 рис., 9 табл., список литературы из 17 наименований. В приложении представлен исходный код программы.

Аннотация. Целью данной работы является автоматизация соревнований по спортивному бриджу.

Для написания соответствующего программного обеспечения были изучены материалы и публикации по предметной области. Результатом разработки является программная система, позволяющая автоматизировать проведение игр, вычисление результатов и поиск системы торговли.

ОГЛАВЛЕНИЕ

РЕФЕРАТ	2
Оглавление	Ошибка! Закладка не определена.
ВВЕДЕНИЕ	5
СПИСОК ТЕРМИНОВ И СОКРАЩЕНИЙ	8
1. АНАЛИТИЧЕСКИЙ ОБЗОР ИССЛЕДУЕМОЙ ОБЛАСТИ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К РАЗРАБАТЫВАЕМОЙ СИСТЕМЕ	9
1.1. Спортивный бридж	9
1.2. Обзор состояния программ по спортивному бриджу	10
1.2.1. BridgeMate	10
1.2.2. BridgeMonitor	11
1.2.3. Ginsberg's Intelligent Bridge Player	11
1.2.4. Bridge Baron	15
1.3. Основные требования к системе	16
1.3.1. Требования к функциональной структуре системы	16
1.3.2. Требования к техническому обеспечению	16
1.3.3. Требования к информационному обеспечению	17
1.4. Постановка цели и задач работы	17
2. РАЗРАБОТКА ПОДСИСТЕМЫ WEBBRIDGEMATE	18
2.1. Описание постановки задачи	18
2.1.1. Характеристика задачи	18
2.1.2. Входная информация	18
2.1.3. Выходная информация	18
2.1.4. Математическая постановка задачи	19
2.2. Описание алгоритма расчета результата	20
2.3. Описание алгоритма перевода очков в IMP	21
2.4. Описание подпрограммы result	22
2.5. Описание подпрограммы convert	25
2.6. Разработка сайта	26
2.7. Описание алгоритма формирования раскладки	26
2.8. Описание алгоритма формирования протокола	27
2.9. Описание подпрограммы r_scheme	27
2.10. Описание подпрограммы res_protocol	27
3. РАЗРАБОТКА КЛИЕНТ-СЕРВЕРНОГО ПРИЛОЖЕНИЯ	31
3.1. Характеристика задачи	31

3.2. Разработка серверной части.....	31
3.2.1. Входная информация.....	31
3.2.4. Выходная информация	31
3.2.5. Описание алгоритмов.....	32
3.3. Разработка клиентской части	34
3.4. Описание контрольного примера.	35
4. РАЗРАБОТКА АНАЛИТИЧЕСКОЙ ПОДСИСТЕМЫ	38
4.1. Постановка задачи.....	38
4.2. Описание построения дерева решений.....	38
4.3. Входная информация	39
4.4. Выходная информация.....	39
4.5. Описание логики алгоритма	40
4.6. Результаты работы подсистемы.....	41
ЗАКЛЮЧЕНИЕ.....	43
ЛИТЕРАТУРА	44
ПРИЛОЖЕНИЕ.....	45

ВВЕДЕНИЕ

Спортивный бридж - единственная карточная игра, признанная международным олимпийским комитетом в качестве вида спорта. По своей сложности, увлекательности и популярности бридж стоит в одном ряду с такими интеллектуальными играми, как шахматы, го или шашки. С 1996 года Американская лига Спортивного бриджа установила официальный Всемирный чемпионат компьютерного бриджа, который проводится ежегодно. Несмотря на такую распространенность, компьютерный бридж находится еще в зачаточном состоянии, в отличие от других перечисленных игр.

В вычислении и анализе результатов турниров по бриджу есть необходимость в большом количестве вычислений. На современном уровне развития вычислительной техники и интернет технологий целесообразно реализовать автоматизированную систему.

Объектом исследования являются программы для игры в спортивный бридж между людьми и компьютерами.

Предметом исследования являются конкретные программы автоматизации и информационной поддержки проведения соревнований по бриджу.

Методы исследования, использованные в работе — теоретические и экспериментальные. В работе применялись теоретические и экспериментальные методы исследования.

К теоретическим методам исследования можно отнести сравнительный анализ различных программных систем, алгоритмов классификации.

Достоверность изложенных положений работы подтверждается результатами практического применения разработанных подходов, алгоритмов, программных средств и технологии анализа игровой ситуации.

Теоретические положения, установленные в работе, обосновываются адекватным выбором исходных посылок и последовательным применением

математического аппарата при получении из них выводов, а также верификацией этих выводов данными систематического исследования полученных аналитических результатов.

Достоверность экспериментальных результатов подтверждается их согласованностью с теоретическими выводами, обоснованным выбором корректных критериев при построении алгоритмов обработки информации, воспроизводимостью результатов на больших объемах экспериментального материала при выполнении серий вычислительных экспериментов с большим количеством изменяемых значений влияющих параметров, наглядностью интерпретации полученных практических результатов обработки информации.

На защиту выносятся результаты разработки и исследования подходов и алгоритмов классификации данных, а также результаты практической реализации этих методов и алгоритмов.

Научной новизной является применение алгоритма построения дерева решений для анализа результатов сыгранных контрактов по бриджу.

Практическая ценность работы заключается в применении подходов и алгоритмов классификации данных.

Разработанный программный комплекс обеспечивает полную поддержку проведения соревнований в интеллектуальной игре «Спортивный бридж».

Объем и структура диссертационной работы. Диссертация содержит введение, 3 главы и заключение, изложенные на 100 с. машинописного текста, а также 2 приложения. В работу включены 5 рис., 2 табл., список литературы из 12 наименований.

В первой главе приводится обзор игры «Спортивный бридж», анализируется существующее на сегодняшний день программное обеспечение.

Во второй главе описан процесс разработки подсистемы для записи и вычисления результатов соревнований.

В третьей главе описана разработка клиент-серверного приложения для игры в бридж.

В четвертой главе рассматриваются алгоритмы построения дерева

решений, используемого для поиска наилучшей системы торговли.

В заключении диссертационной работы сформулированы основные выводы и результаты выполненных исследований.

В приложении помещен текст программы.

СПИСОК ТЕРМИНОВ И СОКРАЩЕНИЙ

Бридж (англ. Bridge) — карточная интеллектуальная командная или парная игра.

Спортивный бридж (англ. Duplicate Bridge) — карточная игра, в которой шансы сторон максимально уравниваются. Чтобы уравнивать шансы играющих, в спортивном бридже одна и та же игровая ситуация разыгрывается разными участниками на нескольких столах, после чего сравниваются результаты.

Сдача — распределение карт в руках игроков после их раздачи в начале партии

Рука — набор из 13 карт, полученный игроком при начале партии бриджа

Дилер — игрок, определенный сдающим карты в данной сдаче (применительно к спортивному бриджу).

Заявка — «ход» игрока в процессе торговли.

Болван — союзник разыгрывающего.

1. АНАЛИТИЧЕСКИЙ ОБЗОР ИССЛЕДУЕМОЙ ОБЛАСТИ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К РАЗРАБАТЫВАЕМОЙ СИСТЕМЕ

1.1. Спортивный бридж

Бридж (*bridge*) – карточная интеллектуальная командная или парная игра. Спортивный бридж – единственная из карточных игр, признанная Международным Олимпийским Комитетом в качестве вида спорта. В спортивном бридже играется определенное количество сдач, причем каждая сдача разыгрывается на двух или более столах разными игроками с последующим сравнением результатов. Задача каждой пары в спортивном бридже – получить лучший результат, чем другие пары с теми же самыми картами. По одной из версий название произошло от старинной русской игры «бирюч», в своё время распространённой в среде русских посыльных (бирючей). Правила современного бриджа, возникли в 1925 г. при участии Вандербилта, который популяризировал игру.

За одним столом играют две пары игроков, игроки одной пары сидят друг напротив друга. Игроки называются по сторонам света: Север (*N*), Восток (*E*), Юг (*S*) и Запад (*W*). Пара *NS* играет против пары *EW*. Для игры используется стандартная колода из 52 карт от двойки до туза четырёх мастей. При раздаче каждый игрок получает по 13 карт. Один из игроков является сдающим (*dealer*). Каждая раздача состоит из двух фаз – торговли и розыгрыша. Заявки в торговле и игра картами производятся игроками поочередно по часовой стрелке.

Целью торговли является определение контракта – обязательства одной пары взять определенное количество взятков в назначенной ею деноминации (козырной масти или в игре без козыря). Во время торговли игроки, начиная со сдающего, по очереди делают заявки. После торговли совершается розыгрыш. Первый ход делает оппонент разыгрывающего, сидящий слева. После этого «болван» (партнер разыгрывающего), кладёт свои карты на стол и его картами

управляет разыгрывающий. Игра ведётся так же, как и в других играх на взятки, бить козырем необязательно. Результатом розыгрыша является количество взяток, полученных каждой парой.

Для создания программ спортивного бриджа необходимо разработать алгоритмы для фаз торговли и розыгрыша. Особенности бриджа: наличие кооперации и конкуренции, неполная информация о раскладе.

1.2. Обзор состояния программ по спортивному бриджу

1.2.1. BridgeMate

В настоящее время подсчет результатов производится либо вручную, либо с использованием аппаратного обеспечения BridgeMate II, выпускаемого голландской фирмой BridgeSystems. У данной системы весьма удобно организован ввод данных, устройство для ввода показано на рисунке 1.1.



Рис. 1.1

Устройство по беспроводному соединению подключается к серверу и передает на него данные. Программное обеспечение Bridgemate Control на сервере рассчитывает результат и заполняет протокол, а данные сохраняются в базе. С помощью клиентского программного обеспечения игроки могут зайти на сервер и посмотреть интересующие результаты.


Система дорогостоящая: 125 евро за 10 аппаратов и программное обеспечение стоимостью в 190 евро. Более того, в России нет распространителей этой

системы, и ее необходимо привозить из-за границы, что накладывает дополнительные расходы на доставку и НДС.

1.2.2. BridgeMonitor

Bridge Monitor позволяет проводить соревнования между различными программами-роботами. Кроме того, позволяет играть по сети (локальной или интернет), может быть использован для обучения бриджу, может использоваться в качестве графического интерфейса в программах-роботах. Поддерживается открытый протокол обмена.



система для соревнования среди программ бриджу (своеобразный «стол») состоит из центрального сервера и узла управления столом, который распределяет сдачи игрового тура по подключенным компьютерам, на каждом из которых установлена программа-игрок в бридж. Перед началом матча операторы противников обмениваются конвенционными картами по системе торговли и вводят эту информацию в базы данных программ. Игра производится при помощи узла управления, через который программы-игроки обмениваются информацией. Программы-игроки установлены на персональных компьютерах офисной производительности. Скорость игры составляет 2 минуты на одну сдачу, что примерно в  2 раза быстрее, чем для игроков-людей

1.2.3. Ginsberg's Intelligent Bridge Player

GIB(Ginsberg's Intelligent Bridge Player)- искусственный интеллект для игры в бридж. **GIB** использует поиск "в лоб", чтобы проанализировать текущую ситуацию. Затем используется метод Монте-Карло, чтобы сгенерировать стратегию, объединяя результаты анализа всех возможных вариантов.

В этом случае, дерево игры - прямое минимаксное дерево, хотя в нем есть несколько минимальных узлов с минимальными листьями, так как игрок, сыгравший во взятке последним, в следующей может быть первым. Ветвление

дерева примерно равно 4, альфа-бета отсечение уменьшает его до 1.7.

Увеличение ходов, используя эвристику уменьшает ветвление до 1.3, если область поиска 10^6 в 6 степени узлов. Поиск разделения уменьшает область приблизительно до 50,000 узлов.

Один путь, по которому мы могли бы продолжить двигаться в реальной ситуации, будет состоять в том, чтобы иметь разыгрывать неизвестные карты случайным образом, оказывая влияние на розыгрыш так, чтобы это было совместимо с результатами торговли и с картами, разыгрываемыми в настоящий момент. Мы могли тогда проанализировать результат игры с двумя «болванами» и решить, какая из стратегий окажется самой эффективной. Усреднение большого количества таких образцов Монте-Карло – один из возможных методов решения проблемы игры с неполной информацией.

Алгоритм Монте-Карло для выбора хода из множества M различных вариантов:

1. Составить множество взяток D , совместимое с контрактом и разыгрываемыми картами
2. Для каждого хода $m \in M$ и $d \in D$ оценить результат игры с двумя «болванами», совершая ход m во взятке d . Запомнить полученный результат.
3. Возвратить значение m , для которого максимально

У метода Монте-Карло есть недостатки. Вариант игры с полной информацией, при котором решение базируется на не изменяющихся данных, предполагает, что информация будет доступна к тому времени, когда следующее решение должно быть принято. Несмотря на это, уровень игры GIB, приравнивается к уровню эксперта.

GIB решает 16 дополнительных задач, используя дополнительные ресурсы в форме дополнительного времени (до 100 секунд за игру), большой образец Монте-Карло (100 взяток вместо 60) и сгенерированные вручную

объяснения контрактов и сдающих карт. Каждый из этих трех факторов, способствует улучшению работы.

Есть два важных технических замечания, которые должны быть сделаны об алгоритме Монте-Карло. Во-первых, отметьте, что мы были бесцеремонны с простым высказыванием: “Выберите множество D взяток, совместимых и с торговлей и с игрой руки к настоящему времени”.

Чтобы выбрать взятки, совместимые с торговлей, мы сначала упрощаем торговлю, ограничивая описывающие каждую из рук. Когда мы разыгрываем карты на руках, используя случайный генератор, у которого есть ограничения по количеству карт у каждого игрока в каждом наборе. Этот набор взяток далее проверяется на наличие элементов, которые не соответствуют ограничению, и каждая из взяток передается модулю торговли, чтобы определить, возможные заявки, которые сделают игроки. Этот процесс как правило занимает одну или две секунды, чтобы сгенерировать полный набор взяток, необходимых алгоритму.

Чтобы соответствовать текущему розыгрышу, непрактично проверять каждое гипотетическое решение против модуля розыгрыша. Вместо этого GIB использует готовый анализ, чтобы определить ошибки, которые могли бы сделать противники. Например, предположите, что GIB играет Д5. Анализ указывает, что в 80% случаев, следующий игрок держит KmS. Это игрока, чтобы не сыграть эту карту. Правило Байеса, чтобы определить вероятность, что у игрока есть KmS. Полученные вероятности используются, чтобы оказать влияние на образец Монте-Карло, используемый алгоритмом.

Второй технический пункт относительно самого алгоритма в том, что необходимо быстро выполнять вычисления, до того как завершится анализ. Есть множество жадных методов, которые могут использоваться, чтобы гарантировать, что ход m не рассматривается, если мы можем показать что $< .$ Алгоритм также использует итеративное расширение, чтобы гарантировать, что

ответ низкой ширины доступен, если поиск высокой ширины не завершится вовремя.

Также относительно скорости, алгоритм требует, чтобы для каждой взятки в образце Монте-Карло и возможного хода, мы оценили получающийся результат точно. Зная, что ход m_1 не такой хороший, как ход m_2 для взятки d ; m_1 может быть лучше, чем m_2 в другом случае, и мы должны сравнить их количественно. Этому подходу поэтому помогает существенно идея разделения поиска, где записи в столе перемещения соответствуют не единственным положениям и их оцененным ценностям, но наборам положений и ценностей.

Торговля основана на использовании базы данных, решения принимаются с помощью моделирования действий каждого кандидата. Отсюда следует необходимость в проектировании, как будет проходить торговля и с каким возможным количеством контрактов начнется розыгрыш.

Второй алгоритм для выбора контракта из множества кандидатов B , используя базу данных Z , которая предлагает контракты в определенных ситуациях.

1. Выбрать множество D заявок, совместимых с текущей торговлей.
2. Для каждого контракта $b \in B$ и каждой заявки $d \in D$, использовать базу данных, чтобы смоделировать, как дальше пойдет торговля. Вычислить результат возможного контракта для игры с двумя болванами, обозначая его $s(b, d)$.
3. Возвратить такое знание b для которого максимально

Как с методом Монте-Карло в розыгрыше, этот подход не принимает во внимание игру с двумя болванами. Игроки-эксперты часто принимают решение не сделать заявку, которая передаст слишком много информации противникам, чтобы сделать задачу защитников максимально трудной. На практике, ошибка в базе данных вряд ли отразится на суждении противников, и попытки GIB восполнить этот недочет будут безуспешными.

Это серьезная проблема относится к любой попытке эвристическим образом смоделировать поведение противника. Трудно отличить хороший ход, потому что у противника нет возможности победить из-за плохого хода, который кажется успешным, потому что эвристика не различает такие варианты.

Есть множество путей, которыми эта проблема могла бы быть рассмотрена подробнее, но ни один из них не идеальный. Самый очевидный - использовать агрессивное поведение GIB для определения ошибки или багов в базе данных, и исправить их. Поскольку база данных большая (приблизительно 6500 правил), это очень медленный процесс.

1.2.4 Bridge Baron

Bridge Baron, 1983 г. – первая полностью автоматизированной программа по бриджу. Розыгрыш реализован на основе эвристик и методов планирования иерархической сети задач (Hierarchical Task-Network – HTN) для решения задачи игры с одним «болваном».

Планирование с ИСЗ это метод планирования с помощью искусственного интеллекта, который создает планы по разложению задачи. Это процесс, в котором система планирования разбивает задачи на мелкие подзадачи, до тех пор, пока задачи не станут примитивными. HTN-системы планирования содержат основные методы. Каждый метод включает в себя описание, как разбивать задачи на подзадачи, с некоторыми ограничениями, которые необходимы для того, чтобы метод был применимым, и некоторые ограничения на подзадачи и отношения между ними. Учитывая задачу, планировщик выбирает применимый метод, иллюстрирует его, чтобы анализировать задачу в подзадачи, и затем выбирает и иллюстрирует примерами другие методы, чтобы анализировать подзадачи еще больше. Если ограничения на подзадачи или взаимодействия среди них будут препятствовать плану быть выполнимыми, то система планирования возвратится и попробует другие методы.

Алгоритм использует процедуру, похожую на разложение задачи чтобы создать игровое дерево, ветки которого представляют перемещения, сгенерированные этими методами. Это применяет все методы, применимые к данному состоянию игры, чтобы произвести новые состояния, и продолжается рекурсивно, пока не остается методов, которые не были уже применены к текущему состоянию игры.

Для дерева игры, сгенерированного таким путем, количество ветвлений от одного состояния к другому не равно количеству ходов, а количеству различных тактических схем, используемых игроком. Чтобы оценить дерево игры в листьях, где происходит ход раздающего, наш алгоритм выбирает игру, которая приводит к самому высокому счету.

Чтобы оценить дерево игры в узлах, где очередь противника разыграть карту, алгоритм берет взвешенное среднее число детей узла, основанных на вероятностях, произведенных доверительной функцией.

1.3. Основные требования к системе

1.3.1. Требования к функциональной структуре системы

1.3.2. Требования к техническому обеспечению

Система будет разбита на две логические части — клиентская и серверная. Серверная сторона предъявляет более жёсткие требования к техническому обеспечению:

1. не менее 1гб оперативной памяти;
2. процессор с частотой работы не менее 1ГГц;
3. наличие доступа в Интернет и до клиентских машин.

У клиентских устройств требования проще. Им требуется иметь на борту установленный веб-браузер с поддержкой изображений и современных веб-технологий. Этому условию удовлетворяют почти все устройства с браузером.

1.3.3. Требования к информационному обеспечению.

Информационное обеспечение системы будет происходить при помощи единой базы данных, которая будет пополняться из различных источников независимо. Всю основную информацию система будет брать оттуда, актуальность хранящихся там данных будут обеспечивать отдельные подсистемы.

1.4. Постановка цели и задач работы

Необходимо разработать программное обеспечение, совмещающее функции системы Bridgemate, программы подсчета результатов турнира, базы игровых сдач. ПО WebBridgeMate реализуется в виде веб-сайта, доступ к которому выполняется со стандартных смартфонов, планшетов или других устройств доступа к интернету. При этом исключается необходимость использования специализированного аппаратного устройства для доступа к системе, как это потребовалось в системе Bridgemate. Кроме того, программная реализация клиента обеспечивает легкость модификации клиента при развитии системы.

Целью работы является автоматизация турниров по бриджу среди компьютеров и людей.

Исходя из цели, были поставлены следующие задачи:

- разработка веб-сайта WebBridgeMate;
- разработка клиент-серверного приложения;
- разработка подсистемы поддержки принятия решений при составлении систем торговли.

2. РАЗРАБОТКА ПОДСИСТЕМЫ WEBBRIDGEMATE

2.1. Описание постановки задачи

2.1.1. Характеристика задачи

Основная задача подсистемы расчета результатов – выполнять все необходимые вычисления. Необходимо обработать полученные данные и рассчитать результат партии.

2.1.2. Входная информация

Входная информация вводится с клавиатуры. Входные данные представлены в таблице 1.

Таблица 1

Входная информация

Описание информации	Тип
Номер сдачи	Целое
Количество взяток	Целое
Зональность	Целое
Контра	Логическое
Реконтра	Логическое
Масть	Строка
Уровень	Целое

2.1.3. Выходная информация

Выходные данные представлены в таблице 2.

Таблица 2

Выходная информация

Описание информации	Тип
---------------------	-----

Результат партии	Целое
------------------	-------

2.1.4. Математическая постановка задачи

В спортивном бридже за каждый сыгранный контракт начисляется заранее известное, строго определенное количество очков. Они зависят от уровня контракта, наличия или отсутствия лишних взяток и была ли заявлена контра или реконтра.

Если разыгрывающая сторона не смогла набрать заказанное количество взяток, то результат вычисляется следующим образом.

Если разыгрывающая сторона не смогла набрать заказанное количество взяток, то результат вычисляется следующим образом:

$$J = -K \cdot (RC + 1), \quad (2.1)$$

где J – результат партии;

K – премия, равная 50 очков, если разыгрывающие были до зоны, и 100 очков, если в зоне;

RC – реконтра.

Результат за сыгранный контракт вычисляется по формуле :

$$J = -S + M + L \cdot I, \quad (2.2)$$

где J – результат партии;

S – штраф, равный 100 очкам за первую недобранную взятку, по 200 за вторую и третью, и по 300 за каждую следующую;

M – премия, начисляемая за набранное количество взяток;

L – уровень карты;

I – масть.

2.2. Описание алгоритма расчета результата

Алгоритм предназначен для расчета результата партии. Подробное описание информации приведено ранее (см. табл. 1). Результатом решения алгоритма является результат партии.

В спортивном бридже за каждый сыгранный контракт начисляется заранее известное, строго определенное количество очков. Они зависят от уровня контракта, наличия или отсутствия лишних взяток и была ли дана контра или реконтра. В отличие от роббера, очки, полученные в прошлых сдачах, не оказывают никакого влияния на подсчет очков в следующих сдачах. Результат каждой сдачи записывается в отдельную колонку и не суммируется с результатами других сдач. За сыгранный контракт в зоне частичной записи сторона получает премию в 50 очков. Если была контра, то премия становится равной 100 очков, и 200, если была реконтра. Выигравшая сторона получает по 20 очков за каждую взятку свыше шести в миноре, по 30 очков в мажоре, и 40 очков за первую взятку в БК (остальные также по 30). Если была контра, эти очки удваиваются, а на реконтре — умножаются на 4

Если контракт геймовый (3БК, 4червы, 4 пики, 5 трэф, 5 бубен или выше), то вместо премии 50 очков начисляется премия 300 очков за сыгранный гейм до зоны и 500 в зоне. Если играется малый шлем (12 взяток), то кроме геймовой премии, сторона получает премию, которая составляет 500 очков до зоны и 750 в зоне. Если играется Большой шлем (13 взяток), то кроме геймовой премии, сторона получает премию за Большой шлем, которая составляет 1000 очков до зоны и 1500 в зоне.

Если разыгрывающая сторона не смогла набрать заказанное количество взяток, вистующие записывают себе над чертой за каждую недобранную взятку 50 очков, если разыгрывающие были до зоны, и 100 очков, если в зоне.

Если была контра, то штраф за подсад становится равным:

- до зоны — 100 очков за первую недобранную взятку, по 200 за вторую и третью, и по 300 за каждую следующую.

- в зоне — 200 очков за первую недобранную взятку, и по 300 за каждую следующую.

При наличии реконтры эти очки удваиваются.

При игре менее 2 пар используется таблица компенсации, имитирующая игру противников (табл. 3)

Таблица 3

Таблица компенсации

РС	до зоны	в зоне
21	50	50
22	70	70
23	110	110
24	200	200
25	300	440
26	350	520
27	400	600
28	430	630
29	460	660
30	490	690
31	600	900
32	750	1050
33	900	1350
34	1000	1500
35	1100	1650
36+	1200	1800

2.3. Описание алгоритма перевода очков в IMP

Результат каждой сдачи переводится в IMP(International Match Point). Очки переводятся следующим образом:

Таблица 4

Перевод результата партии в IMP

Результат	IMP
20-40	1
50-80	2
90-120	3
130-160	4
170-210	5

220-260	6
270-310	7
320-360	8
370-420	9
430-490	10
500-590	11
600-740	12

Продолжение табл.4

Результат	IMP
750-890	13
900-1090	14
1100-1290	15
1300-1490	16
1500-1740	17
1750-1990	18
2000-2240	19
2250-2490	20
2500-2990	21
3000-3490	22
3500-3990	23

Алгоритм предназначен для перевода результата партии в IMP.

Результатом решения алгоритма является результат партии в IMP.

2.4. Описание подпрограммы result

Подпрограмма вычисляет результат партии одной пары.

Входными данными являются номер сдачи, количество взяток, контра, реконтра, масть и уровень (см. табл. 1). Входные данные вводятся с клавиатуры.

Выходные данные представляют собой результат партии для одной пары, победившей в раунде. Результат проигравшей пары такой же, но с отрицательным знаком.

Входные данные и полученные результаты выводятся на экран и заносятся в базу данных в таблицу ses_result для возможности их дальнейшего анализа.

Логика подпрограммы приведена на рис. 2.1.

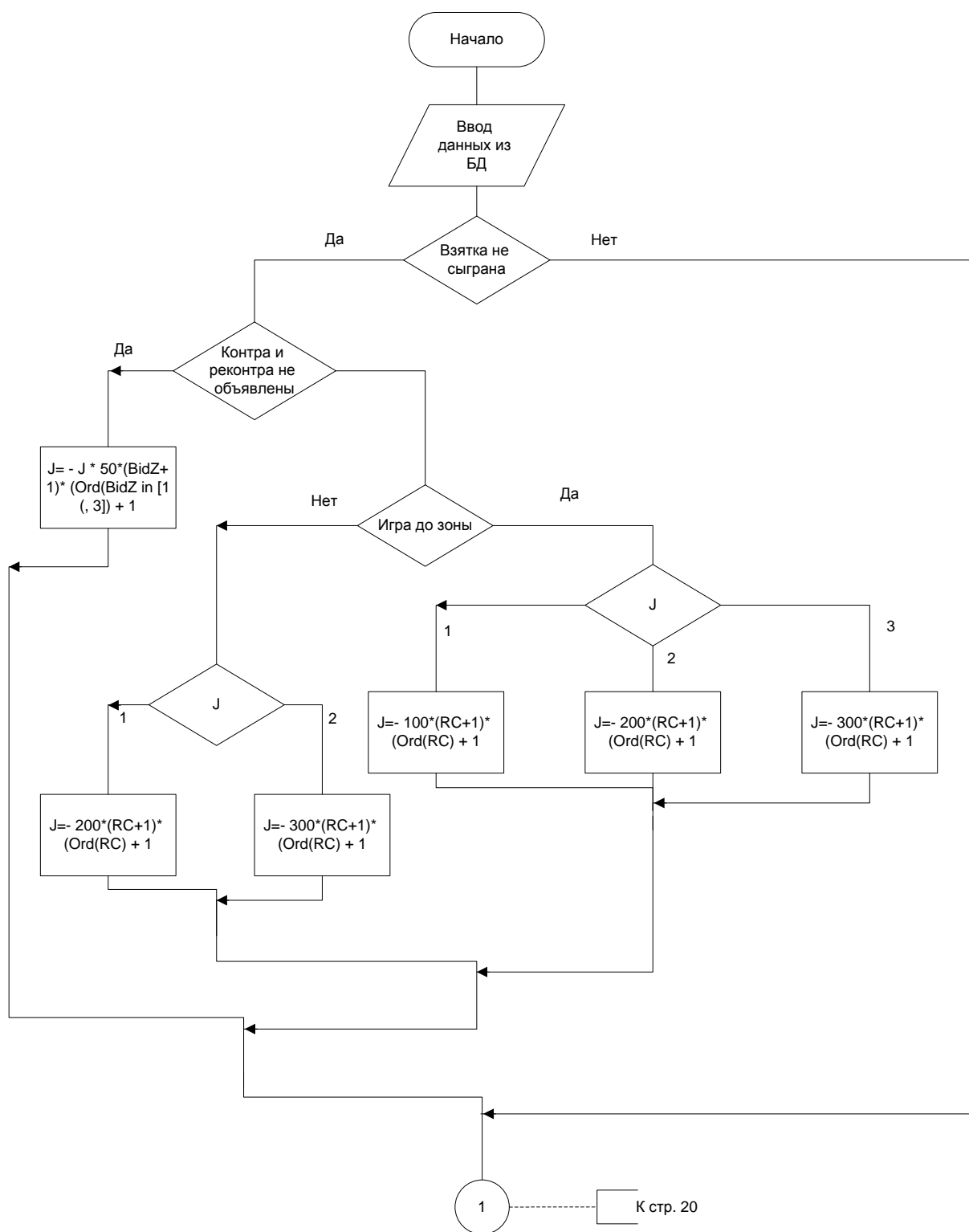
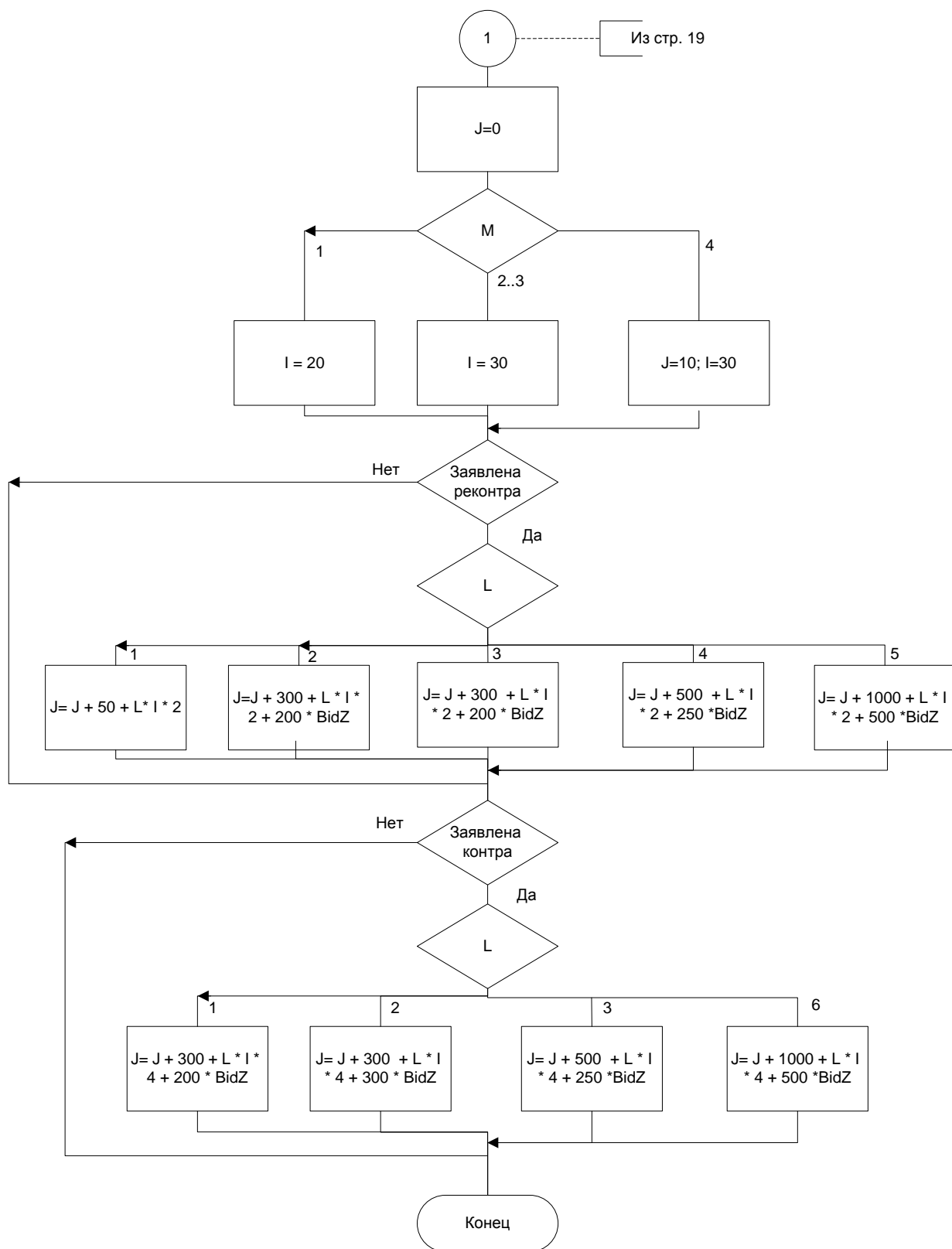


Рис. 2.1



Продолжение рис.2.1

2.5. Описание подпрограммы convert

Подпрограмма конвертирует результат партии в IMP. Входные параметры – результат партии победившей пары и зональность. На выходе получается результат в IMP.

Логика подпрограммы приведена на рис. 2.2.

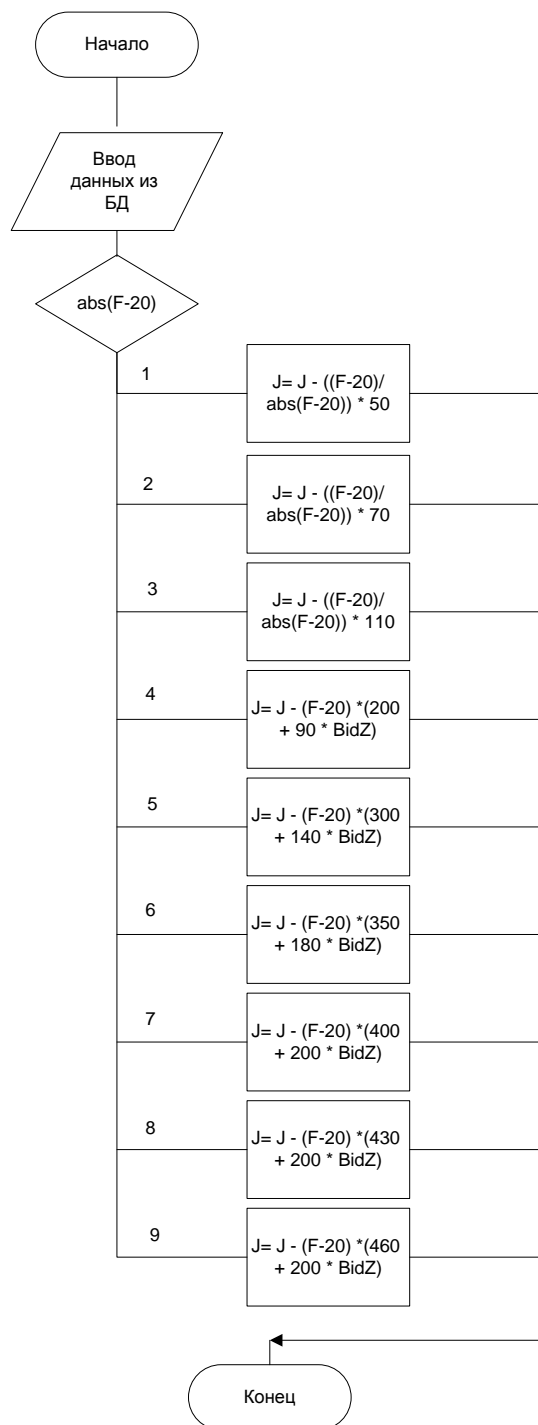


Рис. 2.2

2.6.Разработка сайта

Необходимо генерировать HTML-страницы со всеми необходимыми для проведения турнира формами и протоколами.

Данные о сдаче записываются в протокол, представленный в таблице 5.

Таблица 5

Игровой протокол

Сдача Стол		Коробка:	В зоне: ... Открывает: ...			Дата:		
очки	Номера пар на линии С– Ю	Контракт	Разыгры- вающий	1-й ход	Взят- ки	Резуль- тат	номера пар на линии 3–В	очки

Результаты записываются в таблицу 6:

Таблица 6

Результаты игры

Место	Номер пары	Пара	Разряд	Результат	IMP

2.7. Описание алгоритма формирования раскладки

Для формирования схемы раскладки игроков за столом на время тура, необходимо с помощью запроса считать из базы данных количество пар.

Номер стола генерируется случайно в промежутке чисел:

$$1..N/2, \quad (3.1)$$

где N – номер пары.

На выходе получается сгенерированная таблица, которая выводится в окне браузера.

2.8. Описание алгоритма формирования протокола

Для расчета результата в протоколе используется алгоритм result, описанный ранее в подразделе 2.4. Данные для расчета результата вводятся с клавиатуры в форму протокола.

2.9. Описание подпрограммы r_scheme

На рисунке 3.1 представлена логика подпрограммы.

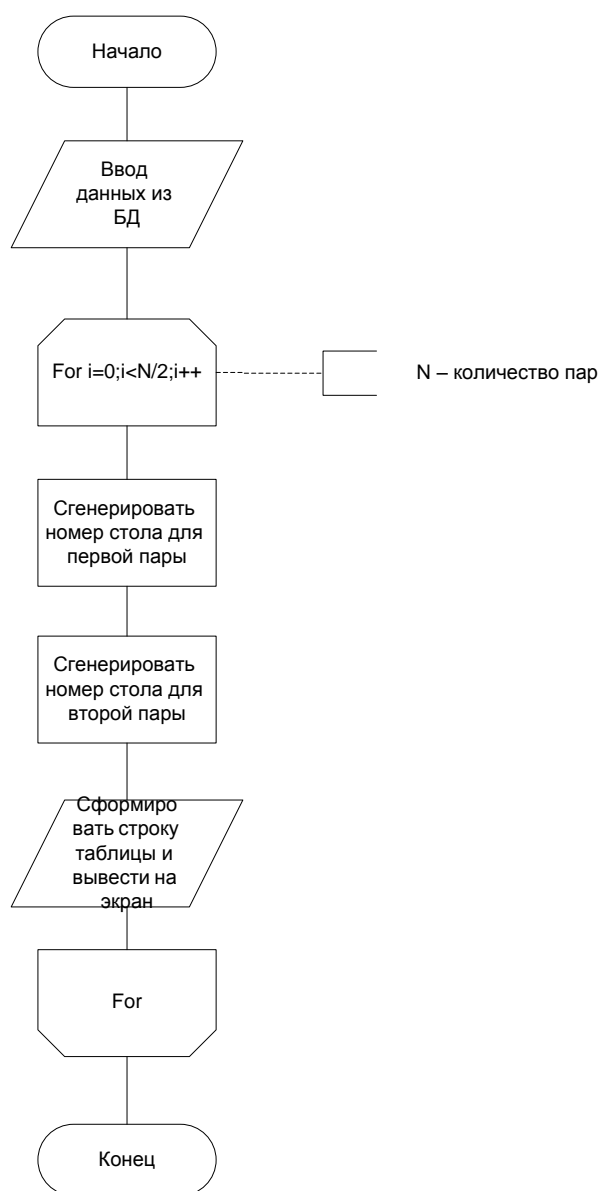


Рис. 2.3

2.10. Описание подпрограммы res_protocol

На рисунке 2.4 представлена логика подпрограммы.

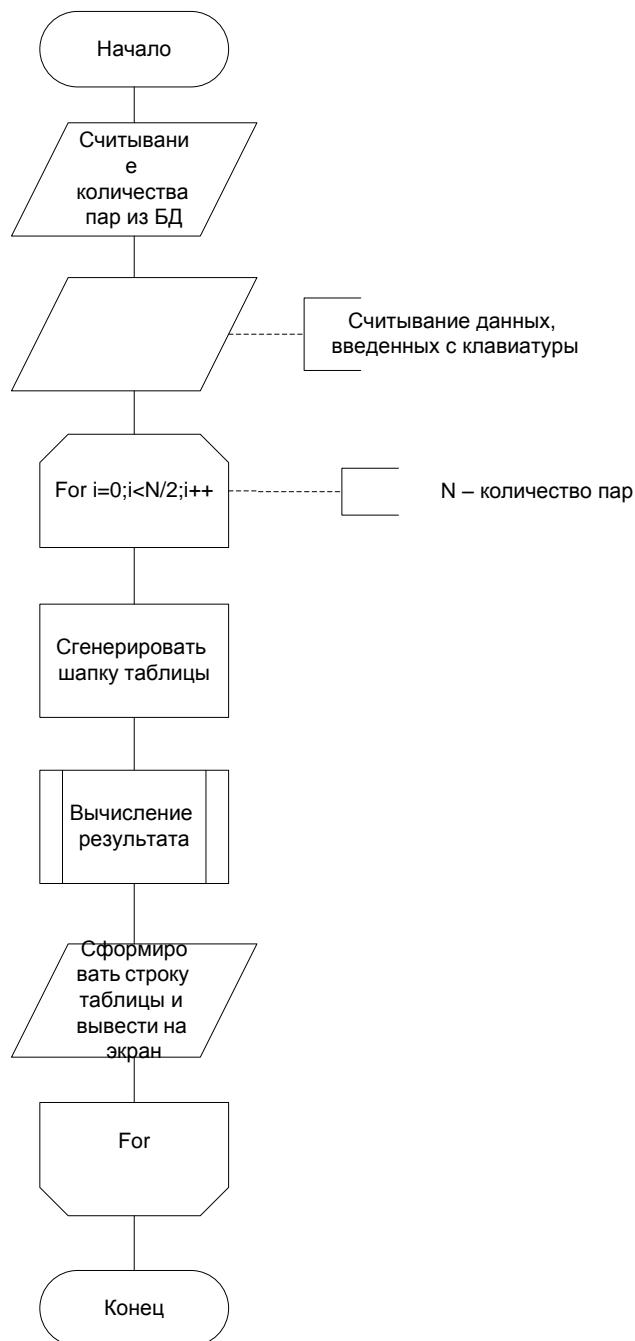


Рис. 2.4

Данный контрольный пример служит для проверки корректности работы подсистемы клиентского приложения системы.

Для проверки корректности работы добавим в базу 12 игроков, сформируем 6 пар и введем произвольные результаты сдачи.

Форма добавления игрока в базу представлена на рис. 2.5

Введите имя игрока:

Рис. 2.5

Пример формы для добавления и редактирования пары игроков представлен на рис.2.6

Игрок 1 Игрок 2 Номер

Рис. 2.6

Схема рассадки игроков за столами показана на рис. 2.7

Стол №	NS	EW
1	1	4
2	2	5
3	3	6

Рис. 2.7

Пример игрового протокола с результатами партии представлен на рис. 2.8

Комплект № 2				Дата 10.06.13			
Сдача № 1				Сдающий N		Зональность None	
№NS	№EW	Контракт	Атака	Взято	Рез	Рез NS	Рез EW
6	1	2 mS	10 mH	1	140	-8	8
2	4	3 mC	10 mD	2	800	8	-8
3	5						

Рис. 2.8

3. РАЗРАБОТКА КЛИЕНТ-СЕРВЕРНОГО ПРИЛОЖЕНИЯ

3.1. Характеристика задачи

Для сравнения эффективности разработанных алгоритмов возникла необходимость в разработке ПО для проведения онлайн соревнований по спортивному бриджу, в которых могли бы участвовать в качестве игроков как люди, так и программы-роботы. Дополнительным требованием является возможность генерации как случайных игровых сдач, так и подключения к базе игровых сдач. Необходимость разработки оригинального ПО вместо использования существующего обусловлено необходимостью использования системы проведения соревнований по бриджу в исследовательских целях и необходимостью интеграции с различными существующими базами игровых сдач, в том числе с базой WebBridgeMate.

3.2. Разработка серверной части

3.2.1. Входная информация

Входная информация представлена в таблице 7

Таблица 7

Описание информации	Тип данных
IP адрес клиента	Строковый
Сторона клиента	Символьный
Информация о раскладе	Строка в базе данных, либо генерируемая случайным образом

3.2.4. Выходная информация

Выходной информацией являются данные о сыгранной сдаче, которые в дальнейшем записываются в базе данных.

3.2.5. Описание алгоритмов

Программа-сервер запускается на отдельном компьютере и играет роль стола, за которым играют бриджисты. Сервер работает с помощью протокола *TCP/IP* и выполняет функции:

- обеспечение процесса торговли и розыгрыша между игрока;
- взаимодействие с базой данных игровых сдач.

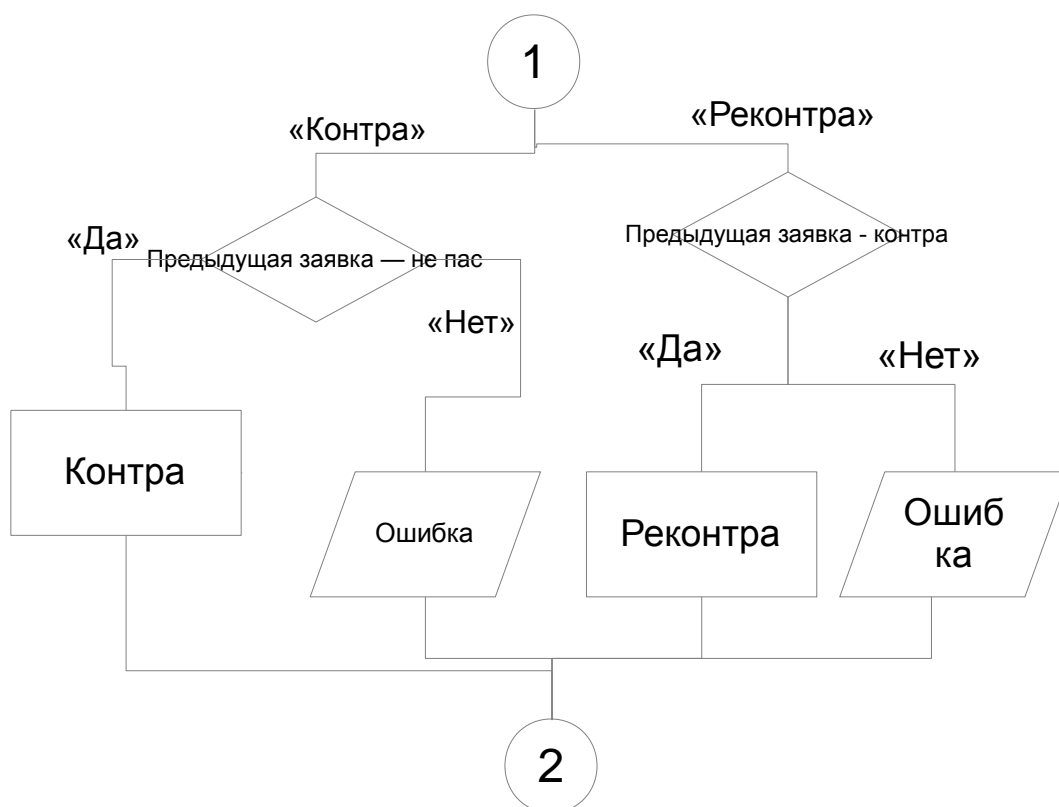
После запуска программа ожидает подключения четырех клиентов.

После подключения всех игроков производится раздача по 13 карт соответствующей сдачи из базы сдач или случайно сгенерированной.

Затем составляется очередность ходов игроков и начинается этап торговли. Сервер проверяет поступающие на торговлю заявки следующим образом: каждое последующее назначение должно быть выше предыдущего либо по уровню, либо по деноминации. В случае нарушения порядка торговли выдается сообщение об ошибке. Торговля прекращается после трёх пасов подряд, последовавших после значащей заявки. Последняя значащая заявка становится контрактом.

Обработка заявок на торговлю представлена на рисунке 3.1.





После определения контракта начинается процесс розыгрыша. В соответствии с очередью выбирается следующий после разыгрывающего игрок, который сделает первый ход. Игроки по очереди отправляют на сервер по одной карте. Картами «болвана» управляет разыгрывающий.

Все взаимодействия между клиентами и сервером выводится в окно сервера и записывается в протокол игры (лог-файл).

По окончании игры результаты записываются в базу игровых сдач.

3.3. Разработка клиентской части

Клиентская программа предназначена для обмена информацией между игроком (бриджистом или роботом) и сервером.

Используя заданный *IP*-адрес и порт сервера, клиент выбирает сторону игрока на столе и подключается к серверу. Процедура подключения представлена на рисунке 3.2.

После подключения всех игроков и получения карт начинается торговля.



3.4. Описание контрольного примера.

На рисунке 3.3. изображено стартовое окно сервера.

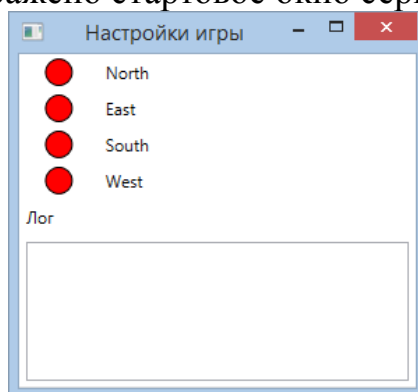


Рис. 3.3

Окно протокола игры представлено на рисунке 3.4.

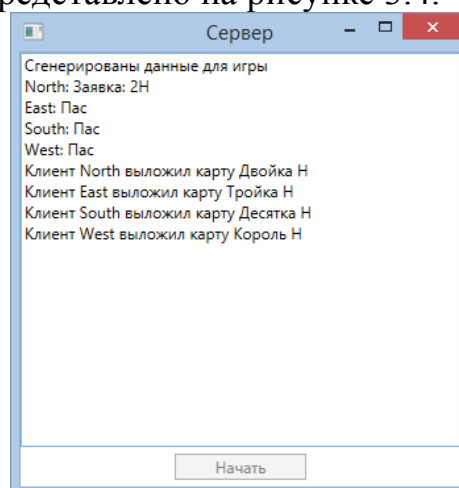


Рисунок 3.4

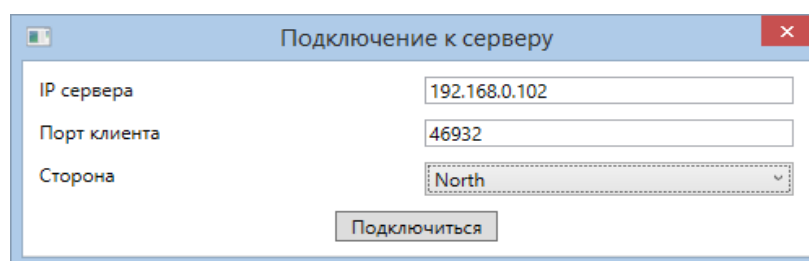


Рисунок 3.5. – Клиентское окно подключения к серверу



Рисунок 3.6. – Клиентское окно торговли



Рисунок 3.5. – Клиентское окно розыгрыша

4. РАЗРАБОТКА АНАЛИТИЧЕСКОЙ ПОДСИСТЕМЫ

4.1. Постановка задачи

Необходимо разработать аналитическую подсистему для поддержки принятия решений при выборе системы торговли в игре.

4.2. Описание построения дерева решений

Для решения задачи выбран алгоритм ID3 — один из алгоритмов для построения дерева принятия решений. Алгоритм начинает работу со всеми обучающими примерами в корневом узле дерева. Для разделения множества примеров корневого узла выбирается один из атрибутов и для каждого значения принимаемого этим атрибутом строится ветвь и создается дочерний узел. Затем все примеры распределяются по дочерним узлам в соответствии со значением атрибута. Пусть атрибут X принимает три значения : A , B и C . Тогда при разбиении исходного множества T по атрибуту X алгоритм создаст три дочерних узла, в первый из которых будут помещены все записи со значением A , во второй — со значением B , а в третий— со значением C .

Алгоритм повторяется рекурсивно до тех пор, пока в узлах не останутся только примеры одного класса, после чего узлы будут объявлены листьями и разбиение прекратится. Наиболее проблемным этапом алгоритма является выбор атрибута, по которому будет производиться разбиение в каждом узле.

Для выбора атрибута разбиения ID3 использует критерий, называемый приростом информации (information gain), или уменьшением энтропии. Введем в рассмотрение меру прироста информации, вычисляемую как:

$$Gain(S) = Info(T) - Info_S(T) \quad (1)$$

где $Info(T)$ -энтропия множества T до разбиения; $Info_S(T)$ - энтропия после разбиения S .

Информационная энтропия находится по формуле:

$$Info(S) = - \sum_i p(x_i) \log_2 p(x_i) \quad (2)$$

где, S - набор данных, для которого вычисляется энтропия; x - количество классов в S ; $p(x)$ - количество элементов класса x , входящих в набор S .

В качестве наилучшего атрибута для использования в разбиении S выбирается тот атрибут, который обеспечивает наибольший прирост информации

4.3. Входная информация

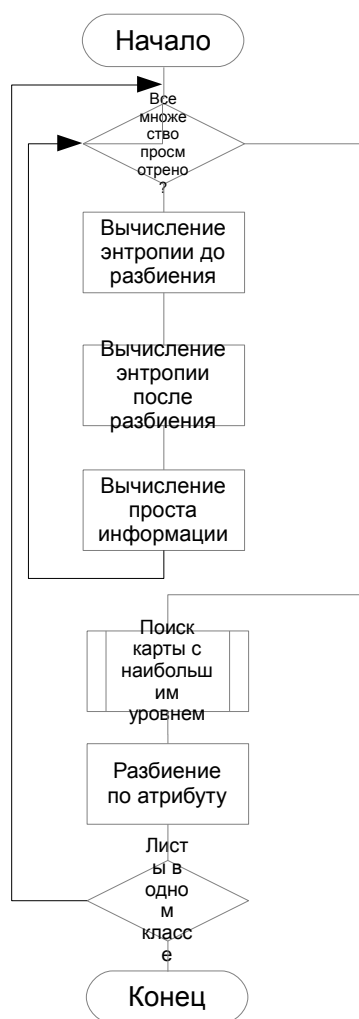
Входная информация представлена в таблице 8

Таблица 8

Описание информации	Тип данных
Карты игрока «Север»	Строковый
Карты игрока «Юг»	Строковый
Карты игрока «Восток»	Строковый
Карты игрока «Запад»	Строковый
Контракт	Строковый
Результат	Целый

4.4. Выходная информация

Выходной информацией служит дерево решений, представляющее собой набор различных условий, по которым можно определить — был контракт сыгран или нет.



4.5. Описание логики алгоритма

Для корректной работы алгоритма для начала следует выделить несколько атрибутов: сумма сил рук и длины мастей на линиях. Расчетная сила выражается в очках НРС (High Point Card) по шкале Милтона Уорка, согласно которой туз оценивается в 4 очка, король – 3, дама – 2, валет – 1. Длина некоторой масти означает число полученных игроком карт в этой масти. По полученной таблице атрибутов строится дерево решений.

Логика алгоритма построения дерева представлена на рисунке 4.1.

4.6. Результаты работы подсистемы

В таблице 9 представлены данные, выбранные из базы по контракту 6 червы.

Таблица 9

KQ JT 7	J	A6 5	KT 93	43 2	98 63	9	QJ 76 4	A9 6	KQ T7 2	Q7	A5 2	85	A5 4	KJ T8 43 2	8	-1
Q	T9 87 2	AT 86 3	84	AT 65		KJ 95	AK Q9 7	72	K6 54 3	Q7 4	T6 5	KJ 98 43	AQ J		J3 2	1
Q	T9 87 2	AT 86 3	84	AT 65		KJ 95	AK Q9 7	72	K6 54 3	Q7 4	T6 5	KJ 98 43	AQ J		J3 2	1
Q	T9 87 2	AT 86 3	84	AT 65		KJ 95	AK Q9 7	72	K6 54 3	Q7 4	T6 5	KJ 98 43	AQ J		J3 2	1
98 2	KQ 74 2	98	86 3	Q6 4	A9 53	KJ 65	AQ	53	85	T7 32	J7 54 2	AK JT 7	JT	AQ 4	KT 9	0
98 2	KQ 74 2	98	86 3	Q6 4	A9 53	KJ 65	AQ	53	85	T7 32	J7 54 2	AK JT 7	JT	AQ 4	KT 9	1
AK 86	5	J6 54	AJ 74	JT 7	A8 64		KT 86 53	Q5 43	KQ 93	AK 83	2	92	JT 72	QT 97 2	J2	-1
84	95 42	K9 3	AT 76	KQ T3 2	A8 63		98 53	95	J7	QJ 87 6	QJ 42	AJ 76	KQ T	AT 54 2	K	0
T5 42	AK 75	AK 64	3	AJ	T4	J9 75 3	86 54	KQ 76 3	8	Q	AK JT 97	98	QJ 96 32	T8 2	Q2	0
T5 42	AK 75	AK 64	3	AJ	T4	J9 75 3	86 54	KQ 76 3	8	Q	AK JT 97	98	QJ 96 32	T8 2	Q2	0
32	KQ	KT 5	Q9 87 43	AJ 86 5	2	Q7 62	KJ 5	97 4	87 43	AJ 4	T6 2	KQ T	AJ T9 65	J9 3	A	-1
54 3	T7 62	J9 86	J8	JT 86	AK 85 4	A4	K2	7	QJ 3	KT 52	QT 74 3	AK Q9 2	9	Q7 3	A9 65	0
54 3	T7 62	J9 86	J8	JT 86	AK 85 4	A4	K2	7	QJ 3	KT 52	QT 74 3	AK Q9 2	9	Q7 3	A9 65	1
AQ J9	J6 3	KJ T6	AQ	T8 54	AT 5	A4	95 32	73 2	KQ 74 2	98	JT 4	K6	98	Q7 53 2	K8 76	-2
9	T9 53 2	J2	Q7 53 2	K7 5	J8 7	98 64	T6 4	AQ J6 42	A6	AK T5	A	T8 3	KQ 4	Q7 3	KJ 98	-1

Подсистема обрабатывает полученные данные и преобразует их к виду, представленному на рисунке 4.2.

Обработка данных

Дерево ID3

Дерево C4.5

☐ Применить пороговые значения для чисел

Разыгрывающий сила	Разыгрывающий длина пик	Разыгрывающий длина черов	Разыгрывающий длина бубен	Разыгрывающий длина трефов	Вистующий сила	Вистующий длина пик	Вистующий длина черов	Вистующий длина бубен	Вистующий длина трефов	Валют вздох	Ключ сдачи	Контракт	Разыгрывающий	Карта атаки	Раздающий сдачи	Зональность сдачи	Карты стороны Север	Карты стороны Юг	
14	7	5	8	6	26	6	8	5	7	-1	81	6Sxx	E	Dx	E	NS	2♥3♥7♥Q♥7♠A+2♠3♥7♠J♠K♠4♠6♠	6♥3♠5♠6♠8♠9♠J♠8♠Q♣2♥7♠9♠J♠	4♥8
24	5	8	6	7	16	8	5	7	6	-2	6	6Sx	N	DA	E	NS	3♥6♥10♥6♠10♠J♠K♠Q♠A♠9♠J♠Q♠A♠	2♥4♥7♥Q♥K♥8♥9♥4♥10♥10♥2♥3♥7♠	8♥9
29	7	6	5	8	11	6	7	8	5	-1	24	6S	N	Dx	W	ALL	1♥5♠6♠A♠3♠9♠10♠K♥7♠10♠J♠Q♠K♠	2♥7♥10♥Q♥K♥7♥4♥Q♠2♥5♥A♠6♥9♥A♠	4♥5
11	5	10	8	3	29	8	3	5	10	1	32	6S	W	HT	W	ALL	2♥7♥8♥9♥10♥3♠6♠8♠10♠A♠4♠8♠Q♠	3♥4♥5♥6♥K♥4♥7♥Q♠5♥6♥10♥2♥7♠	J♥Q
11	5	10	8	3	29	8	3	5	10	1	32	6S	W	Hx	W	ALL	2♥7♥8♥9♥10♥3♠6♠8♠10♠A♠4♠8♠Q♠	3♥4♥5♥6♥K♥4♥7♥Q♠5♥6♥10♥2♥7♠	J♥Q
11	5	10	8	3	29	8	3	5	10	1	32	6S	W	Sx	W	ALL	2♥7♥8♥9♥10♥3♠6♠8♠10♠A♠4♠8♠Q♠	3♥4♥5♥6♥K♥4♥7♥Q♠5♥6♥10♥2♥7♠	J♥Q
6	8	7	6	5	34	5	6	7	8	0	37	6S	W	Sx	W	NS	2♥4♥7♥Q♥K♥8♥9♥3♠6♠8♠2♠8♠9♠	5♥8♥2♥3♥7♥10♥2♥4♥5♥7♥10♥3♥5♠	10♥
6	8	7	6	5	34	5	6	7	8	1	37	6S	W	--	W	NS	2♥4♥7♥Q♥K♥8♥9♥3♠6♠8♠2♠8♠9♠	5♥8♥2♥3♥7♥10♥2♥4♥5♥7♥10♥3♥5♠	10♥
27	5	5	8	8	12	8	8	5	5	-1	45	6S	N	HA	E	NS	5♥4♠5♠6♠J♠4♥7♥10♥A♠6♠8♠K♠A♠	3♥9♥Q♥K♥3♥8♥K♥A♠2♥3♥4♥5♥Q♠	2♥7
14	8	6	8	4	26	5	7	5	9	0	67	6S	E	DQ	S	EW	2♥4♥5♥9♥3♥9♥K♥6♥7♥10♥A♠4♠8♠	7♥J♥6♥7♥8♥J♥Q♠2♥4♥J♥Q♠5♥9♠	10♥
29	7	5	5	9	11	6	8	8	4	0	70	6S	S	HQ	N	EW	5♥7♥K♥A♠4♠6♠K♥A♠3♠2♠4♠5♠10♠	8♥Q♠7♥9♥10♥J♥K♥A♠3♠6♥7♥Q♥K♠	2♥3
29	7	5	5	9	11	6	8	8	4	0	70	6S	S	HQ	N	EW	5♥7♥K♥A♠4♠6♠K♥A♠3♠2♠4♠5♠10♠	8♥Q♠7♥9♥10♥J♥K♥A♠3♠6♥7♥Q♥K♠	2♥3
15	9	6	6	5	26	4	7	7	8	-1	82	6S	E	DA	S	EW	Q♥K♥5♥10♥K♥3♠4♠7♥8♥9♥Q♠2♠3♠	3♥4♥7♥8♥4♥J♥A♠2♠6♠10♥4♥7♥9♠	5♥6

Рис. 4.2.

На рисунке 4.3. показано построенное дерево решений.

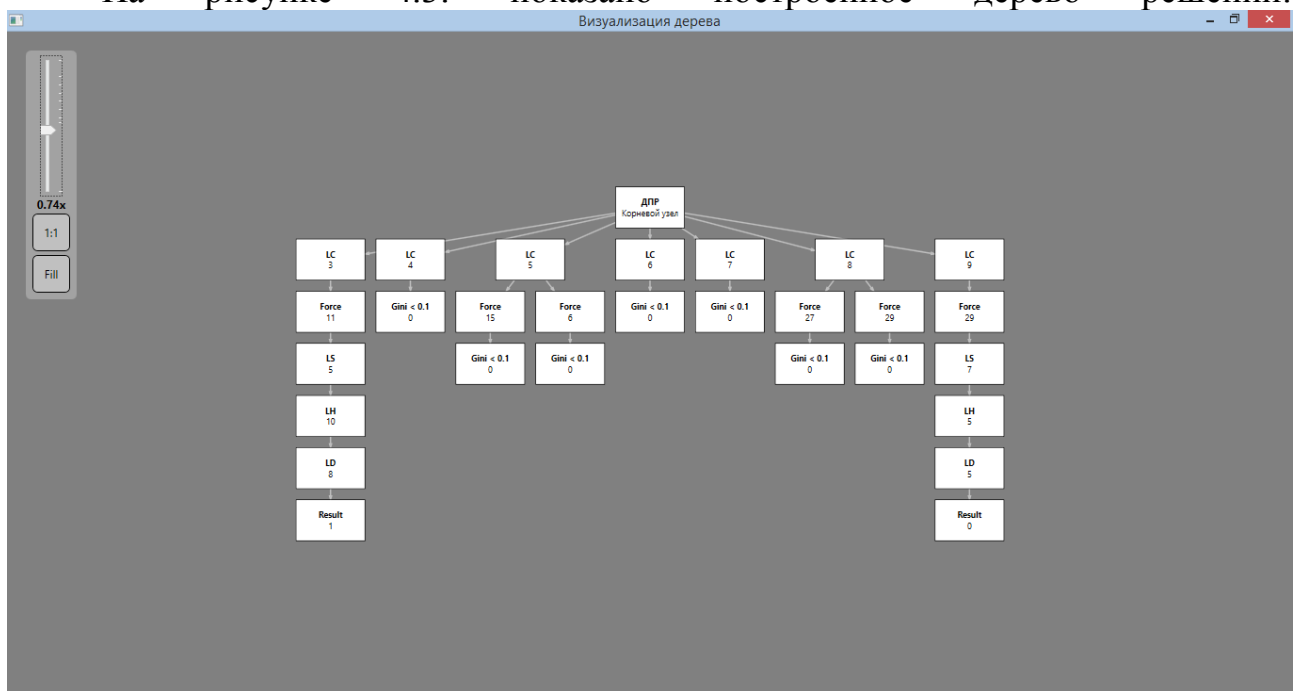


Рис.4.3.

ЗАКЛЮЧЕНИЕ

По итогам выполненной работы разработана система, позволяющая автоматизировать игры по спортивному бриджу и значительно сократить время, расходуемое на раздачу карт, заполнение протоколов и вычисление результата.

Для этого в работе успешно выполнены следующие задачи:

Направления дальнейшего развития системы:

1. Использование алгоритмов шифрования при подключении клиентов и при передаче данных для обеспечения информационной безопасности системы.
2. Модификация системы с целью соответствия существующим открытым стандартам в ПО бриджа: формат *.rbn для протоколирования игры; протокол обмена между сервером и клиентами.
3. Подключение к общедоступным базам сдач бриджа.

ЛИТЕРАТУРА

1. Коробейников А.В., С.И. Зыкин, Р.Х. Судуров, И.С. Ефремова Обзор состояния программ спортивного бриджа. // Информационные системы в промышленности и образовании: Сборник трудов молодых ученых. - Ижевск: Изд-во ИжГТУ, - 2012. - С. 69-77.
2. Бабкин Э.А., Козырев О.Р., Куркина И.В. Принципы и алгоритмы искусственного интеллекта.// – Н. Новгород: НГТУ, 2006. -132 с.
3. Градовский О.Т., Феранчук И.Д., Шадыро О.Л. Играйте в бридж.— Минск: Мет, 1997. — 176 с.
4. Люгер Д.Ф. Искусственный интеллект. Стратегии и методы решения сложных проблем. // М.: Вильямс, 2003. – 864 с.
5. Риз. Бридж для начинающих. Искусство побеждать. // М.: Центрполиграф. – 2010 г. – 160 с.
6. Сайт проекта Bridgemate (bridgemate.com). Дата обращения 01.05.2015.
7. Сайт проекта WBridge5 (wbridge5.com). Дата обращения 01.05.2015.
8. Официальный сайт Всемирного чемпионата по компьютерному (allevybridge.com/allevy/computerbridge). Дата обращения 01.05.2015.
9. Al Levy. Participating Computer Bridge Software for the World Computer Bridge Championship and the History of these Software Programs. Summary of Participants at Computer Olympiads (bridgeguys.com/CGlossary/Computer/CBProgrammers.pdf). Дата обращения 01.05.2015.
10. Коробейников А.В., Кошелева Ю.С., Система проведения онлайн соревнований по спортивному бриджу
11. Сайт проекта BridgeBaseOnline (bridgebase.com). Дата обращения 01.05.2015.
12. The State of Automated Bridge Play // New York University, 2010. – 8 p. (ephman.org/bridgeReview200908.pdf). Дата обращения 01.04.2012.
13. Бабкин Э.А., Козырев О.Р., Куркина И.В. Принципы и алгоритмы искусственного интеллекта.// – Н. Новгород: НГТУ, 2006. -132 с.
14. Питер Норвиг, Стьюарт Рассел. Искусственный интеллект: современный подход. .:Вильямс, 2006. – 1408 с.

Класс Gamer

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Server
{
    /// <summary>
    /// Игрок
    /// </summary>
    [Serializable]
    public class Gamer
    {
        /// <summary>
        /// Ip адрес
        /// </summary>
        public string Ip { get; set; }

        /// <summary>
        /// Номер порта
        /// </summary>
        public int Port { get; set; }

        /// <summary>
        /// Карты на руках
        /// </summary>
        public List<Card> Cards = new List<Card>();

        /// <summary>
        /// Выкладываемая карта
        /// </summary>
        public Card LayOutCard { get; set; }

        /// <summary>
        /// Сторона
        /// </summary>
        public WorldPart WorldPart { get; set; }

        /// <summary>
        /// Заявка
        /// </summary>
        public BiddingCall BiddingCall { get; set; }
    }
}

```

```

using System;
using System.Collections.Generic;
using System.Linq;
using Server;

namespace DecisionTree.BasicData
{
    /// <summary>
    /// Информация о картах игрока
    /// </summary>

```

```

public class PlayerCardInfo
{
    /// <summary>
    /// Карты в масти пики
    /// </summary>
    private string _spades { get; set; }

    /// <summary>
    /// Карты в масти червы
    /// </summary>
    private string _hearts { get; set; }

    /// <summary>
    /// Карты в масти бубны
    /// </summary>
    private string _diamonds { get; set; }

    /// <summary>
    /// Карты в масти трефы
    /// </summary>
    private string _clubs { get; set; }

    /// <summary>
    /// Сила
    /// </summary>
    public int Force { get; set; }

    /// <summary>
    /// Длина масти пики
    /// </summary>
    public int SpadeLength { get; set; }

    /// <summary>
    /// Длина масти червы
    /// </summary>
    public int HeartLength { get; set; }

    /// <summary>
    /// Длина масти бубны
    /// </summary>
    public int DiamondLength { get; set; }

    /// <summary>
    /// Длина масти трефы
    /// </summary>
    public int ClubLength { get; set; }

    /// <summary>
    /// Карты
    /// </summary>
    public List<Card> Cards { get; set; }

    /// <summary>
    /// Карты (в виде текста)
    /// </summary>
    public string CardsAsText { get; set; }

    /// <summary>
    /// Конструктор
    /// </summary>
    public PlayerCardInfo(string spades, string hearts, string diamonds, string clubs)
    {
        _spades = spades;
        _hearts = hearts;
    }
}

```

```

_diamonds = diamonds;
_clubs = clubs;

// Конвертируем карты
Cards = new List<Card>();
ConvertCard(_spades, Suite.S);
ConvertCard(_hearts, Suite.H);
ConvertCard(_diamonds, Suite.D);
ConvertCard(_clubs, Suite.C);

// Сортируем по масти и рангу
Cards = Cards.OrderBy(x => x.Suite).ThenBy(y => y.Rank).ToList();

// Получаем силу
Force = Cards.Sum(x => GetForce(x));

// Получаем длины
SpadeLength = Cards.Count(x => x.Suite == Suite.S);
HeartLength = Cards.Count(x => x.Suite == Suite.H);
DiamondLength = Cards.Count(x => x.Suite == Suite.D);
ClubLength = Cards.Count(x => x.Suite == Suite.C);

// Заполняем текст
List<Card> cards = Cards.FindAll(x => x.Suite == Suite.H);
foreach (Card card in cards)
    CardsAsText += ConvertRank(card.Rank) + "♥";
cards = Cards.FindAll(x => x.Suite == Suite.D);
foreach (Card card in cards)
    CardsAsText += ConvertRank(card.Rank) + "♦";
cards = Cards.FindAll(x => x.Suite == Suite.S);
foreach (Card card in cards)
    CardsAsText += ConvertRank(card.Rank) + "♠";
cards = Cards.FindAll(x => x.Suite == Suite.C);
foreach (Card card in cards)
    CardsAsText += ConvertRank(card.Rank) + "♣";
}

/// <summary>
/// Конвертировать карты в формат программы
/// </summary>
/// <param name="cards">Карты</param>
/// <param name="suite">Масть</param>
private void ConvertCard(string cards, Suite suite)
{
    char[] ranks = cards.ToCharArray();
    foreach (char rank in ranks)
    {
        Cards.Add(new Card()
        {
            Rank = ConvertRank(rank),
            Suite = suite,
        });
    }
}

/// <summary>
/// Преобразование ранга карты в формат программы
/// </summary>
/// <param name="rank">Ранг</param>
/// <returns>Ранг в формате программы</returns>
private Rank ConvertRank(char rank)
{
    switch (rank)
    {

```

```

        case '2':
            return Rank.Двойка;
        case '3':
            return Rank.Тройка;
        case '4':
            return Rank.Четвёрка;
        case '5':
            return Rank.Пятёрка;
        case '6':
            return Rank.Шестёрка;
        case '7':
            return Rank.Семёрка;
        case '8':
            return Rank.Восьмёрка;
        case '9':
            return Rank.Девятка;
        case 'T':
            return Rank.Десятка;
        case 'J':
            return Rank.Валет;
        case 'Q':
            return Rank.Дама;
        case 'K':
            return Rank.Король;
        case 'A':
            return Rank.Туз;
        default:
            throw new ArgumentException();
    }
}

/// <summary>
/// Получаем силу карты
/// </summary>
/// <param name="card">Карта</param>
/// <returns>Сила карты</returns>
private int GetForce(Card card)
{
    switch (card.Rank)
    {
        case Rank.Валет:
            return 1;
        case Rank.Дама:
            return 2;
        case Rank.Король:
            return 3;
        case Rank.Туз:
            return 4;
        case Rank.Восьмёрка:
        case Rank.Двойка:
        case Rank.Девятка:
        case Rank.Десятка:
        case Rank.Пятёрка:
        case Rank.Семёрка:
        case Rank.Тройка:
        case Rank.Четвёрка:
        case Rank.Шестёрка:
            return 0;
        default:
            throw new ArgumentException();
    }
}

/// <summary>

```



```

    /// Преобразование ранга карты в формат для чтения
    /// </summary>
    /// <param name="rank">Ранг в формате программы</param>
    /// <returns>Ранг в формате для чтения</returns>
    private string ConvertRank(Rank rank)
    {
        switch (rank)
        {
            case Rank.Валет:
                return "J";
            case Rank.Восьмёрка:
                return "8";
            case Rank.Дама:
                return "Q";
            case Rank.Двойка:
                return "2";
            case Rank.Девятка:
                return "9";
            case Rank.Десятка:
                return "10";
            case Rank.Король:
                return "K";
            case Rank.Пятёрка:
                return "5";
            case Rank.Семёрка:
                return "7";
            case Rank.Тройка:
                return "3";
            case Rank.Туз:
                return "A";
            case Rank.Четвёрка:
                return "4";
            case Rank.Шестёрка:
                return "6";
            default:
                throw new ArgumentException();
        }
    }
}

```

Класс Network

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.IO;
using System.Linq;
using System.Net;
using System.Net.Sockets;
using System.Runtime.Serialization.Formatters.Binary;
using System.Text;
using System.Threading.Tasks;

namespace Server
{
    /// <summary>
    /// Сеть
    /// </summary>
    public static class Network
    {
        /// <summary>
        /// Размер буфера
    }
}

```

```

/// </summary>
static int _bufferSize = 65536;

/// <summary>
/// Сокеты для входящих (список из-за генератора игры)
/// </summary>
public static List<Socket> ListenerSockets = new List<Socket>();

/// <summary>
/// Получить Ip-адрес
/// </summary>
/// <param name="ipAddress">Ip-адрес в виде строки</param>
/// <returns>Ip-адрес</returns>
public static IPAddress GetIpAddress(string ipAddress = null)
{
    IPEndPoint ipHost = null;
    if (string.IsNullOrEmpty(ipAddress))
        ipHost = Dns.GetHostEntry(Dns.GetHostName());
    else
        ipHost = Dns.GetHostEntry(ipAddress.Trim());

    IPAddress ip = ipHost.AddressList.FirstOrDefault(x =>
x.ToString().StartsWith("192"));
    if(ip==null)
        ip = ipHost.AddressList.FirstOrDefault(x => x.ToString().StartsWith("127"));
    return ip;
}

/// <summary>
/// Создать сокет и конечную точку
/// </summary>
/// <param name="numPort">Номер порта</param>
/// <param name="ipAddress">Ip-адрес в виде строки</param>
/// <returns>Сокет</returns>
public static Tuple<Socket, IPEndPoint> CreateSocket(int numPort, string ipAddress =
null)
{
    // Получаем адрес Ip
    IPAddress ipAddr = Network.GetIpAddress(ipAddress);

    // Создаём конечную точку
    IPEndPoint ipEndPoint = new IPEndPoint(ipAddr, numPort);

    // Создаем сокет и привязываем его к конечной точке
    Socket socket = new Socket(ipAddr.AddressFamily, SocketType.Stream,
ProtocolType.Tcp);

    return new Tuple<Socket, IPEndPoint>(socket, ipEndPoint);
}

/// <summary>
/// Ожидание входящих данных
/// </summary>
/// <typeparam name="T">Тип входящих данных</typeparam>
/// <param name="numPort">Номер порта</param>
/// <param name="backgroundWorker">BackgroundWorker, в котором выполняется
прослушка</param>
public static void Listen<T>(int numPort, BackgroundWorker backgroundWorker)
{
    // Создаём сокет для прослушивания
    Tuple<Socket, IPEndPoint> connectionInfo = Network.CreateSocket(numPort);
    Socket socket = connectionInfo.Item1;
    ListenerSockets.Add(socket);
}

```

```

// Привязываем сокет к конечной точке
socket.Bind(connectionInfo.Item2);

// Устанавливаем сокет в состояние прослушивания
socket.Listen(10);

// Начинаем слушать соединения
while (true)
{
    try
    {
        // Ожидание входящих данных
        Socket handler = socket.Accept();

        // Если есть данные, то принимаем их
        byte[] bytes = new byte[_bufferSize];
        int bytesRec = handler.Receive(bytes);

        // Освобождаем ресурсы
        handler.Shutdown(SocketShutdown.Both);
        handler.Close();

        if (!(backgroundWorker as BackgroundWorker).CancellationPending)
        {
            // Распаковываем данные
            BinaryFormatter formatter = new BinaryFormatter();
            object data = (T)formatter.Deserialize(new MemoryStream(bytes));

            // Вызовем обработчик данных
            (backgroundWorker as BackgroundWorker).ReportProgress(0, data);
        }
    }
    catch (Exception ex)
    {
        // В случае исключения прекращаем прослушку
        break;
    }
}

/// <summary>
/// Отправить данных
/// </summary>
/// <param name="numPort">Номер порта</param>
/// <param name="ipAddress">Ip-адрес в виде строки</param>
/// <param name="data">Данные</param>
/// <param name="isGetAnswer">Флаг прихода ответа</param>
public static void Send(int numPort, string ipAddr, object data)
{
    // Создаём сокет
    Tuple<Socket, IPEndPoint> connectionInfo = Network.CreateSocket(numPort,
ipAddr);

    Socket socket = connectionInfo.Item1;

    // Соединяемся с удалённой точкой
    socket.Connect(connectionInfo.Item2);

    // Упаковываем данные
    BinaryFormatter formatter = new BinaryFormatter();
    MemoryStream memoryStream = new MemoryStream();
    formatter.Serialize(memoryStream, data);

    // Отправляем данные
    socket.Send(memoryStream.ToArray());
}

```

```

        // Освобождаем ресурсы
        socket.Shutdown(SocketShutdown.Both);
        socket.Close();
    }

    /// <summary>
    /// Закрыть сокет
    /// </summary>
    /// <param name="numPort">Номер порта</param>
    public static void CloseSocket(int numPort)
    {
        Socket socket = ListenerSockets.First(x => ((IPEndPoint)x.LocalEndPoint).Port ==
numPort);
        socket.Close();
        ListenerSockets.Remove(socket);
    }
}

using System;
using System.Collections.Generic;
using System.Configuration;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Server
{
    /// <summary>
    /// Игра
    /// </summary>
    [Serializable]
    public class Game
    {
        /// <summary>
        /// Список игроков
        /// </summary>
        public List<Gamer> Gamers { get; set; }

        /// <summary>
        /// Игрок, который делает ход
        /// </summary>
        public WorldPart CurrentGamer { get; set; }

        /// <summary>
        /// Порт сервера
        /// </summary>
        public static int Port
        {
            get { return Convert.ToInt32(ConfigurationManager.AppSettings["ServerPort"]); }
        }

        /// <summary>
        /// Число карт у каждого игрока вначале
        /// </summary>
        public static int PlayerBeginCard = 13;

        /// <summary>
        /// Колода карт
        /// </summary>
        public Deck Deck { get; set; }

        /// <summary>
        /// Текстовое сообщение для лога

```

```

/// </summary>
public string LogMessage { get; set; }

/// <summary>
/// Сообщение об успешности подключения
/// </summary>
public static string SuccessMessage = "Успешно";

/// <summary>
/// Счетчик пасов
/// </summary>
public int passCounter { get; set; }

/// <summary>
/// Состояние игры
/// </summary>
public GameState GameState { get; set; }

/// <summary>
/// Очередь заявок
/// </summary>
public List<BiddingCall> BiddingList { get; set; }

/// <summary>
/// Контракт
/// </summary>
public static BiddingCall Contract { get; set; }

/// <summary>
/// Конструктор
/// </summary>
public Game()
{
    Deck = new Deck();
    Deck.Mix();
    Gamers = new List<Gamer>();
    BiddingList = new List<BiddingCall>();
    CurrentGamer = WorldPart.North;
    passCounter = 0;
}

/// <summary>
/// Переключить игрока
/// </summary>
public void ChangeCurrentGamer()
{
    switch (CurrentGamer)
    {
        case WorldPart.North:
            CurrentGamer = WorldPart.East;
            break;
        case WorldPart.East:
            CurrentGamer = WorldPart.South;
            break;
        case WorldPart.South:
            CurrentGamer = WorldPart.West;
            break;
        case WorldPart.West:
            CurrentGamer = WorldPart.North;
            break;
    }
}

/// <summary>

```

```

    /// Торговля
    /// </summary>
    public void Bidding()
    {
        if (GameBeginCheck())
        {
            GameState = GameState.Game;
            LogMessage = "Торговля закончена";
            return;
        }
    }

    /// <summary>
    /// Проверка на начало игры
    /// </summary>
    /// <returns>Результат проверки</returns>
    public bool GameBeginCheck()
    {
        // Игра начинается, когда три последние ставки - пас
        if (BiddingList.Count < 3)
            return false;
        for (int x = BiddingList.Count - 1; x >= BiddingList.Count - 3; x--)
            if (BiddingList[x].BCall != Call.Пас)
                return false;

        return true;
    }

    public void CardPlay()
    {
    }

    /// <summary>
    /// Раздать карты
    /// </summary>
    public void DistributeCards()
    {
        foreach (Gamer gamer in Gamers)
        {
            gamer.Cards = this.Deck.GetRange(0, Game.PlayerBeginCard);
            this.Deck.RemoveRange(0, Game.PlayerBeginCard);
        }
    }
}

using System;
using System.Collections.Generic;
using System.Configuration;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Server
{
    /// <summary>
    /// Игра
    /// </summary>
    [Serializable]
    public class Game
    {

```

```

/// <summary>
/// Список игроков
/// </summary>
public List<Gamer> Gamers { get; set; }

/// <summary>
/// Игрок, который делает ход
/// </summary>
public WorldPart CurrentGamer { get; set; }

/// <summary>
/// Порт сервера
/// </summary>
public static int Port
{
    get { return Convert.ToInt32(ConfigurationManager.AppSettings["ServerPort"]); }
}

/// <summary>
/// Число карт у каждого игрока вначале
/// </summary>
public static int PlayerBeginCard = 13;

/// <summary>
/// Колода карт
/// </summary>
public Deck Deck { get; set; }

/// <summary>
/// Текстовое сообщение для лога
/// </summary>
public string LogMessage { get; set; }

/// <summary>
/// Сообщение об успешности подключения
/// </summary>
public static string SuccessMessage = "Успешно";

/// <summary>
/// Счетчик пасов
/// </summary>
public int passCounter { get; set; }

/// <summary>
/// Состояние игры
/// </summary>
public GameState GameState { get; set; }

/// <summary>
/// Очередь заявок
/// </summary>
public List<BiddingCall> BiddingList { get; set; }

/// <summary>
/// Контракт
/// </summary>
public static BiddingCall Contract { get; set; }

/// <summary>
/// Конструктор
/// </summary>
public Game()
{
    Deck = new Deck();
}

```

```

        Deck.Mix();
        Gamers = new List<Gamer>();
        BiddingList = new List<BiddingCall>();
        CurrentGamer = WorldPart.North;
        passCounter = 0;
    }

    /// <summary>
    /// Переключить игрока
    /// </summary>
    public void ChangeCurrentGamer()
    {
        switch (CurrentGamer)
        {
            case WorldPart.North:
                CurrentGamer = WorldPart.East;
                break;
            case WorldPart.East:
                CurrentGamer = WorldPart.South;
                break;
            case WorldPart.South:
                CurrentGamer = WorldPart.West;
                break;
            case WorldPart.West:
                CurrentGamer = WorldPart.North;
                break;
        }
    }

    /// <summary>
    /// Торговля
    /// </summary>
    public void Bidding()
    {
        if (GameBeginCheck())
        {
            GameState = GameState.Game;
            LogMessage = "Торговля закончена";
            return;
        }
    }

    /// <summary>
    /// Проверка на начало игры
    /// </summary>
    /// <returns>Результат проверки</returns>
    public bool GameBeginCheck()
    {
        // Игра начинается, когда три последние ставки - пас
        if (BiddingList.Count < 3)
            return false;
        for (int x = BiddingList.Count - 1; x >= BiddingList.Count - 3; x--)
            if (BiddingList[x].BCall != Call.Пас)
                return false;

        return true;
    }

    public void CardPlay()
    {
    }

```



```

    /// <summary>
    /// Раздать карты
    /// </summary>
    public void DistributeCards()
    {
        foreach (Gamer gamer in Gamers)
        {
            gamer.Cards = this.Deck.GetRange(0, Game.PlayerBeginCard);
            this.Deck.RemoveRange(0, Game.PlayerBeginCard);
        }
    }
}

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.IO;
using System.Linq;
using System.Net;
using System.Net.Sockets;
using System.Runtime.Serialization.Formatters.Binary;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace Server
{
    /// <summary>
    /// Окно сервера
    /// </summary>
    public partial class ServerWindow : Window
    {
        /// <summary>
        /// Игpa
        /// </summary>
        public Game Game { get; set; }

        /// <summary>
        /// Конструктор
        /// </summary>
        /// <param name="game">Игpa</param>
        public ServerWindow(Game game)
        {
            InitializeComponent();
            Game = game;

            BackgroundWorker backgroundWorker = new BackgroundWorker();
            backgroundWorker.WorkerReportsProgress = true;
            backgroundWorker.DoWork += backgroundWorker_DoWork;
            backgroundWorker.ProgressChanged += backgroundWorker_ProgressChanged;
            backgroundWorker.RunWorkerAsync();
        }
    }
}

```

```

        LogTextBox.AppendText("Сгенерированы данные для игры\n");
    }

    /// <summary>
    /// Запуск нового потока для ожидания входящих данных
    /// </summary>
    void backgroundWorker_DoWork(object sender, DoWorkEventArgs e)
    {
        Network.Listen<Game>(Game.Port, (BackgroundWorker)sender);
    }

    /// <summary>
    /// Обработка входящих данных
    /// </summary>
    private void backgroundWorker_ProgressChanged(object sender,
ProgressChangedEventArgs e)
    {
        Game = (Game) e.UserState;

        // Добавляем в лог
        LogTextBox.AppendText(Game.LogMessage + "\n");

        // Действия в зависимости от состояния игры
        switch (Game.GameState)
        {
            // Сама игра
            case GameState.Game:
                Game.CardPlay();
                break;
            // Торговля
            case GameState.Bidding:
                Game.Bidding();
                break;
            default:
                throw new ArgumentException();
        }

        // Отправляем новые данные клиентам
        SendToClients();
    }

    /// <summary>
    /// Нажатие на кнопку "Начать игру"
    /// </summary>
    private void StartGameButton_Click(object sender, RoutedEventArgs e)
    {
        // Раздать карты
        Game.DistributeCards();

        // Отправляем новые данные клиентам
        Game.LogMessage = "Сервер: произведена раздача карт";
        SendToClients();
        ((Button)sender).IsEnabled = false;
        Game.Bidding();
    }

    /// <summary>
    /// Отправить данные клиента
    /// </summary>
    void SendToClients()
    {
        foreach (Gamer gamer in Game.Gamers)
            Network.Send(gamer.Port, gamer.Ip, Game);
    }

```

```

    }
}
Стартовое окно сервера

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.IO;
using System.Linq;
using System.Net;
using System.Net.Sockets;
using System.Runtime.Serialization.Formatters.Binary;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace Server
{
    /// <summary>
    /// Окно настроек сервера
    /// </summary>
    public partial class ServerStartWindow : Window
    {
        /// <summary>
        /// Игра
        /// </summary>
        public Game _game { get; set; }

        /// <summary>
        /// Асинхронная операция
        /// </summary>
        BackgroundWorker _backgroundWorker { get; set; }

        /// <summary>
        /// Конструктор
        /// </summary>
        public ServerStartWindow()
        {
            InitializeComponent();
            _game = new Game();

            _backgroundWorker = new BackgroundWorker();
            _backgroundWorker.WorkerReportsProgress = true;
            _backgroundWorker.WorkerSupportsCancellation = true;
            _backgroundWorker.DoWork += backgroundWorker_DoWork;
            _backgroundWorker.ProgressChanged += backgroundWorker_ProgressChanged;
            _backgroundWorker.RunWorkerAsync();
        }

        /// <summary>
        /// Запуск нового потока для ожидания входящих данных
        /// </summary>
        void backgroundWorker_DoWork(object sender, DoWorkEventArgs e)
        {
            Network.Listen<Gamer>(Game.Port, (BackgroundWorker)sender);
        }
    }
}

```

```

/// <summary>
/// Обработка входящих данных
/// </summary>
void backgroundWorker_ProgressChanged(object sender, ProgressChangedEventArgs e)
{
    string message = Game.SuccessMessage;
    Gamer gamer = (Gamer)e.UserState;

    // Проверки
    if (_game.Gamers.Any(x => x.WorldPart == gamer.WorldPart))
        message = "Уже есть зарегистрированный игрок с данной стороной";

    if (message == Game.SuccessMessage)
    {
        _game.Gamers.Add(gamer);
        LogTextBox.AppendText(string.Format("Подключился клиент {0} {1}:{2}\n",
gamer.WorldPart, gamer.Ip, gamer.Port));

        Ellipse gamerEllipse = null;
        switch (gamer.WorldPart)
        {
            case WorldPart.North:
                gamerEllipse = NorthEllipse;
                break;
            case WorldPart.East:
                gamerEllipse = EastEllipse;
                break;
            case WorldPart.South:
                gamerEllipse = SouthEllipse;
                break;
            case WorldPart.West:
                gamerEllipse = WestEllipse;
                break;
        }

        gamerEllipse.Fill = Brushes.Green;
    }

    Network.Send(gamer.Port, gamer.Ip, Encoding.UTF8.GetBytes(message));

    // Если в игре все игроки, то запускаем игру
    if (_game.Gamers.Count == 4)
    {
        Network.CloseSocket(Game.Port);

        // Открываем окно сервера
        ServerWindow serverWindow = new ServerWindow(_game);
        serverWindow.Show();
        this.Hide();
    }
}
}
}
}

```

Клиентская часть

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;

```

```

using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using Server;
using System.Net;
using System.Net.Sockets;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;
using System.Configuration;
using System.ComponentModel;

namespace Client
{
    /// <summary>
    /// Стартовое окно клиента
    /// </summary>
    public partial class StartClientWindow : Window
    {
        /// <summary>
        /// Игрок
        /// </summary>
        Gamer _gamer = new Gamer();

        /// <summary>
        /// IP сервера
        /// </summary>
        string _ipServer;

        /// <summary>
        /// Асинхронная операция
        /// </summary>
        BackgroundWorker _backgroundWorker { get; set; }

        /// <summary>
        /// Конструктор
        /// </summary>
        public StartClientWindow()
        {
            InitializeComponent();

            WorldPartComboBox.ItemsSource = Enum.GetValues(typeof(WorldPart));
            Random rand = new Random();
            _gamer.Port = rand.Next(2000, 65000);
            _gamer.Ip = Network.GetIpAddress().ToString();
            ServerIpTextBox.Text = _ipServer = ConfigurationManager.AppSettings["ServerIp"];
            this.DataContext = _gamer;

            _backgroundWorker = new BackgroundWorker();
            _backgroundWorker.WorkerReportsProgress = true;
            _backgroundWorker.WorkerSupportsCancellation = true;
            _backgroundWorker.DoWork += backgroundWorker_DoWork;
            _backgroundWorker.ProgressChanged += backgroundWorker_ProgressChanged;
            _backgroundWorker.RunWorkerAsync(_gamer.Port);
        }

        /// <summary>
        /// Обработка входящих данных
        /// </summary>
        void backgroundWorker_ProgressChanged(object sender, ProgressChangedEventArgs e)
        {

```

```

        // Получаем сообщение
        string message = Encoding.UTF8.GetString((byte[])e.UserState);
        if (message != Game.SuccessMessage)
            throw new ApplicationException(message);
        //тут нужна обработка исключения

        // Закрываем сокет
        Network.CloseSocket(_gamer.Port);

        // Открываем окно с торговлей
        BiddingClientWindow biddingWindow = new BiddingClientWindow(_gamer.WorldPart,
_gamer.Port, _ipServer);
        biddingWindow.Show();
        this.Hide();
    }

    /// <summary>
    /// Запуск нового потока для ожидания входящих данных
    /// </summary>
    void backgroundWorker_DoWork(object sender, DoWorkEventArgs e)
    {
        Network.Listen<byte[]>((int)e.Argument, (BackgroundWorker)sender);
    }

    /// <summary>
    /// Нажатие на кнопку "Подключиться"
    /// </summary>
    private void SendButton_Click(object sender, RoutedEventArgs e)
    {
        try
        {
            _ipServer = ServerIpTextBox.Text.Trim();

            if (string.IsNullOrEmpty(_ipServer))
                throw new ApplicationException("Введите IP сервера");
            if (_gamer.Port == 0)
                throw new ApplicationException("Введите порт клиента");
            if (_gamer.WorldPart == WorldPart.Nothing)
                throw new ApplicationException("Выберите сторону");

            Network.Send(Game.Port, _ipServer, _gamer);
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message, "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
        }
    }

    /// <summary>
    /// Закрытие окна
    /// </summary>
    private void Window_Closing(object sender, System.ComponentModel.CancelEventArgs e)
    {
        Application.Current.Shutdown();
    }
}

using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Windows;

```

```

using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Shapes;
using Server;

namespace Client
{
    /// <summary>
    /// Окно для клиента
    /// </summary>
    public partial class ClientWindow
    {
        /// <summary>
        /// Текущий игрок
        /// </summary>
        Gamer CurrentGamer
        {
            get
            {
                if (_game != null)
                    return _game.Gamers.First(x => x.WorldPart == _worldPart);
                else
                    return null;
            }
        }

        /// <summary>
        /// Сторона, за которую играет игрок
        /// </summary>
        WorldPart _worldPart { get; set; }

        /// <summary>
        /// Игра
        /// </summary>
        static Game _game { get; set; }

        /// <summary>
        /// Колода карт (для визуального представления)
        /// </summary>
        List<VisualCard> _visualCards { get; set; }

        /// <summary>
        /// Стил для карт
        /// </summary>
        Style _style { get; set; }

        /// <summary>
        /// IP сервера
        /// </summary>
        readonly string _ipServer;

        /// <summary>
        /// Конструктор
        /// </summary>
        public ClientWindow(): this(WorldPart.Nothing, 0, null, null)
        {
        }

        /// <summary>
        /// Конструктор
        /// </summary>
        /// <param name="worldPart">Сторона, за которую играет игрок</param>
        /// <param name="port">Порт</param>
        /// <param name="ipServer">Ip-адрес в виде строки</param>
    }
}

```

```

/// <param name="game">Игра</param>
public ClientWindow(WorldPart worldPart, int port, string ipServer, Game game)
{
    InitializeComponent();

    _worldPart = worldPart;
    _style = (Style)FindResource("CardStyle");
    _ipServer = ipServer;
    _game = game;

    this.Title = string.Format("Клиент. Сторона {0}, порт {1}", _worldPart, port);
    _visualCards = VisualCard.GenerateVisualDeck(_style);
    UpdateGame();

    BackgroundWorker backgroundWorker = new BackgroundWorker {WorkerReportsProgress
= true};
    backgroundWorker.DoWork += backgroundWorker_DoWork;
    backgroundWorker.ProgressChanged += backgroundWorker_ProgressChanged;
    backgroundWorker.RunWorkerAsync(port);
}

/// <summary>
/// Обработка входящих данных
/// </summary>
void backgroundWorker_ProgressChanged(object sender, ProgressChangedEventArgs e)
{
    _game = (Game)e.UserState;
    UpdateGame();
}

/// <summary>
/// Запуск нового потока для ожидания входящих данных
/// </summary>
void backgroundWorker_DoWork(object sender, DoWorkEventArgs e)
{
    Network.Listen<Game>((int)e.Argument, (BackgroundWorker)sender);
}

/// <summary>
/// Клик по карте
/// </summary>
void VisualRectangle_PreviewMouseDown(object sender, MouseButtonEventArgs e)
{
    // Если это карта игрока, то скрываем её у игрока и помещаем в центр стола
    Rectangle rectangle = ((Rectangle)sender);
    VisualCard visualCard = _visualCards.First(x => x.VisualRectangle == rectangle);
    Card card = CurrentGamer.Cards.FirstOrDefault(x => x.Rank == visualCard.Rank &&
x.Suite == visualCard.Suite);
    if (card != null)
    {
        CurrentGamer.Cards.Remove(card);
        CurrentGamer.LayOutCard = card;
        _game.LogMessage = string.Format("Клиент {0} выложил карту {1} {2}",
CurrentGamer.WorldPart, card.Rank, card.Suite);

        // Отправляем состояние игры серверу
        SendToServer();
    }
}

/// <summary>
/// Отправляем серверу
/// </summary>
private void SendToServer()

```



```

{
    Network.Send(Game.Port, _ipServer, _game);
}

/// <summary>
/// Обновить игру
/// </summary>
public void UpdateGame()
{
    foreach (Gamer gamer in _game.Gamers)
    {
        // Рисуем карты игроков
        Canvas canvas = null;
        switch (gamer.WorldPart)
        {
            case WorldPart.North:
                canvas = NorthCanvas;
                break;
            case WorldPart.South:
                canvas = SouthCanvas;
                break;
            case WorldPart.West:
                canvas = WestCanvas;
                break;
            case WorldPart.East:
                canvas = EastCanvas;
                break;
        }

        double leftPosition = 30.0;
        double topPosition = -20.0;
        bool isTurned = (CurrentGamer == gamer) ? false : true;
        canvas.Children.Clear();
        foreach (Card card in gamer.Cards)
        {
            VisualCard visualCard = _visualCards.First(x => x.Suite == card.Suite &&
x.Rank == card.Rank);
            visualCard.IsTurned = isTurned;
            visualCard.UpdateTexture();
            visualCard.VisualRectangle.PreviewMouseDown +=
VisualRectangle_PreviewMouseDown;
            canvas.Children.Add(visualCard.VisualRectangle);
            if (gamer.WorldPart == WorldPart.North || gamer.WorldPart ==
WorldPart.South)
            {
                Canvas.SetLeft(visualCard.VisualRectangle, leftPosition);
                Canvas.SetTop(visualCard.VisualRectangle, this.Height / 8 -
visualCard.VisualRectangle.Height / 2);
                leftPosition += visualCard.VisualRectangle.Width / 3;
            }
            else
            {
                Canvas.SetTop(visualCard.VisualRectangle, topPosition);
                Canvas.SetLeft(visualCard.VisualRectangle, (this.Width / 4 -
visualCard.VisualRectangle.Width) / 2);
                topPosition += visualCard.VisualRectangle.Height / 4;
            }
        }

        // Рисуем карты в центре
        if (gamer.LayOutCard != null)
        {
            VisualCard centerVisualCard = new VisualCard(gamer.LayOutCard, _style);
            centerVisualCard.UpdateTexture();
        }
    }
}

```



```

private Style _style { get; set; }

/// <summary>
/// Колода карт (для визуального представления)
/// </summary>
private List<VisualCard> _visualCards { get; set; }

/// <summary>
/// Конструктор
/// </summary>
/// <param name="worldPart">Сторона, за которую играет игрок</param>
/// <param name="port">Порт</param>
/// <param name="ipServer">Ip-адрес в виде строки</param>
public BiddingClientWindow(WorldPart worldPart, int port, string ipServer)
{
    InitializeComponent();

    _worldPart = worldPart;
    _ipServer = ipServer;
    _style = (Style) FindResource("CardStyle");
    _visualCards = VisualCard.GenerateVisualDeck(_style);
    DenominationComboBox.ItemsSource = Enum.GetValues(typeof(Suite));
    LevelComboBox.ItemsSource = new List<int>() {1,2,3,4,5,6,7};
    this.Title = string.Format("Торговля. {0} {1}", worldPart, port);

    BackgroundWorker backgroundWorker = new BackgroundWorker {WorkerReportsProgress
= true};
    backgroundWorker.DoWork += backgroundWorker_DoWork;
    backgroundWorker.ProgressChanged += backgroundWorker_ProgressChanged;
    backgroundWorker.RunWorkerAsync(port);
}

/// <summary>
/// Обработка входящих данных
/// </summary>
private void backgroundWorker_ProgressChanged(object sender,
ProgressChangedEventArgs e)
{
    _game = (Game) e.UserState;

    // Добавляем в лог
    LogTextBox.AppendText(_game.LogMessage + "\n");

    // Действия в зависимости от состояния игры
    switch (_game.GameState)
    {
        // Сама игра
        case GameState.Game:
            // Закрываем сокет
            Network.CloseSocket(_gamer.Port);
            // Открываем окно клиента
            ClientWindow clientWindow = new ClientWindow(_gamer.WorldPart,
_gamer.Port, _ipServer, _game);
            clientWindow.Show();
            this.Hide();
            break;

        // Торговля
        case GameState.Bidding:
            _gamer = _game.Gamers.First(x => x.WorldPart == _worldPart);
            // Генерируем колоду карт (если ещё не сгенерирована)
            if (CardCanvas.Children.Count == 0)
            {

```

```

        double leftPosition = 30.0;
        foreach (Card card in _gamer.Cards)
        {
            VisualCard visualCard = _visualCards.First(x => x.Suite ==
card.Suite && x.Rank == card.Rank);
            visualCard.UpdateTexture();
            CardCanvas.Children.Add(visualCard.VisualRectangle);
            Canvas.SetLeft(visualCard.VisualRectangle, leftPosition);
            Canvas.SetTop(visualCard.VisualRectangle, 20);
            leftPosition += visualCard.VisualRectangle.Width/3;
        }
    }
    break;

    default:
        throw new ArgumentException();
    }
}

/// <summary>
/// Запуск нового потока для ожидания входящих данных
/// </summary>
void backgroundWorker_DoWork(object sender, DoWorkEventArgs e)
{
    Network.Listen<Game>((int)e.Argument, (BackgroundWorker)sender);
}

/// <summary>
/// Нажатие на одну из кнопок торговли
/// </summary>
private void BiddingButton_Click(object sender, RoutedEventArgs e)
{
    if (_gamer != null)
    {
        Button button = (Button)sender;
        BiddingCall biddingCall = null;
        switch (button.Content.ToString())
        {
            case "Отправить":
                biddingCall = new BiddingCall((int)LevelComboBox.SelectedItem,
(Suite)DenominationComboBox.SelectedItem);
                _game.LogMessage = string.Format("{0}: Заявка: {1}{2}", _worldPart,
LevelComboBox.SelectedItem, DenominationComboBox.SelectedItem);
                break;
            case "Пас":
                biddingCall = new BiddingCall(Call.Пас);
                _game.LogMessage = string.Format("{0}: Пас", _worldPart);
                break;
        }
        _gamer.BiddingCall = biddingCall;
        _game.BiddingList.Add(biddingCall);
        Network.Send(Game.Port, _ipServer, _game);
    }
}
}

using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Windows.Controls;

namespace DecisionTree.BridgeDecisionTrees

```

```

{
    /// <summary>
    /// Дерево принятия решений для бриджа
    /// </summary>
    public class BridgeDecisionTree: TreeView
    {
        /// <summary>
        /// Наименование столбца с результатами
        /// </summary>
        public static string ResultColumnName = "Result";

        /// <summary>
        /// Исходные данные
        /// </summary>
        List<BasicData.BasicDataItem> _basicData { get; set; }

        /// <summary>
        /// Тип дерева
        /// </summary>
        BridgeDecisionTreeType _treeType { get; set; }

        /// <summary>
        /// Конструктор
        /// </summary>
        /// <param name="basicData">Исходные данные</param>
        /// <param name="treeType">Тип дерева</param>
        /// <param name="isThresholds">Флаг использования пороговых значений для
чисел</param>
        public BridgeDecisionTree(List<BasicData.BasicDataItem> basicData,
BridgeDecisionTreeType treeType, bool isThresholds)
        {
            // Использовать ли пороги
            Type typeNumbers = isThresholds ? typeof(int) : typeof(string);

            // Конвертируем данные в DataTable
            DataTable table = new DataTable();
            // table.Columns.Add(new DataColumn("Contract", typeof(string)));
            table.Columns.Add(new DataColumn("Force", typeNumbers));
            // table.Columns.Add(new DataColumn("_Force", typeNumbers));
            table.Columns.Add(new DataColumn("LS", typeNumbers));
            table.Columns.Add(new DataColumn("LH", typeNumbers));
            table.Columns.Add(new DataColumn("LC", typeNumbers));
            table.Columns.Add(new DataColumn("LD", typeNumbers));
            //table.Columns.Add(new DataColumn("_LS", typeNumbers));
            //table.Columns.Add(new DataColumn("_LH", typeNumbers));
            //table.Columns.Add(new DataColumn("_LC", typeNumbers));
            //table.Columns.Add(new DataColumn("_LD", typeNumbers));
            table.Columns.Add(new DataColumn(ResultColumnName, typeof(string)));

            foreach (BasicData.BasicDataItem data in basicData)
            {
                DataRow row = table.NewRow();
                // row["Contract"] = data.DataFromExcel.Contract;
                row["Force"] = data.DealerCardInfo.Force;
                // row["_Force"] = data.WhistCardInfo.Force;
                row["LS"] = data.DealerCardInfo.SpadeLength;
                row["LH"] = data.DealerCardInfo.HeartLength;
                row["LC"] = data.DealerCardInfo.ClubLength;
                row["LD"] = data.DealerCardInfo.DiamondLength;
                //row["_LS"] = data.WhistCardInfo.SpadeLength;
                //row["_LH"] = data.WhistCardInfo.HeartLength;
                //row["_LC"] = data.WhistCardInfo.ClubLength;
                //row["_LD"] = data.WhistCardInfo.DiamondLength;
                row[ResultColumnName] = data.DataFromExcel.ResultTricks;
            }
        }
    }
}

```

```

        table.Rows.Add(row);
    }

    _basicData = basicData;
    _treeType = treeType;

    // Корневой атрибут
    BridgeDecisionTreeItem rootTreeItem = new BridgeDecisionTreeItem()
    {
        AttributeName = "ДПР",
        Value = "Корневой узел",
    };
    this.Items.Add(rootTreeItem);

    // Формируем дерево
    // Первая итерация
    List<PatritionCriteryInfo> patritionCriteryInfoList = GetGains(table);
    string attr = patritionCriteryInfoList.First(x => x.InformationGain ==
patritionCriteryInfoList.Max(y => y.InformationGain)).AttrName;
    List<object> distinctAttrValues = table.GetDistinctValues(attr);
    distinctAttrValues.Sort();
    foreach (object distinctAttrValue in distinctAttrValues)
    {
        BridgeDecisionTreeItem treeItem = GetNewTreeItem(attr,
distinctAttrValue.ToString(), rootTreeItem);
        // Рекурсивное построение дерева
        GetNextID3Item(table, attr, distinctAttrValue, treeItem);
    }
}

/// <summary>
/// Получить следующий узел
/// </summary>
/// <param name="sourceTable">Исходные данные</param>
/// <param name="attrSource">Исходный атрибут</param>
/// <param name="valueSouce">Исходное значение атрибута</param>
/// <param name="treeItemSource">Узел дерева</param>
/// <param name="isLeft">Значение меньше либо равно (для чисел)</param>
private void GetNextID3Item(DataTable sourceTable, string attrSource, object
valueSouce, BridgeDecisionTreeItem treeItemSource, bool isLeft = true)
{
    // Для отладки
    //List<DataColumn> debugColumns = new List<DataColumn>();
    //foreach (DataColumn debugColumn in sourceTable.Columns)
    //    debugColumns.Add(debugColumn);
    //DataRow[] debugRows = sourceTable.Select();

    bool isDescreet = sourceTable.Columns[attrSource].DataType == typeof (string);

    // Создание подмножеств
    DataTable table = sourceTable.Copy();
    if (isDescreet)
    {
        for (int x = table.Rows.Count - 1; x >= 0; x--)
            if (table.Rows[x][attrSource].ToString() !=
Convert.ToString(valueSouce))
                table.Rows.RemoveAt(x);
    }
    else
    {
        if (isLeft)
        {
            for (int x = table.Rows.Count - 1; x >= 0; x--)

```

```

        if (Convert.ToInt32(table.Rows[x][attrSource]) >
Convert.ToInt32(valueSouce))
            table.Rows.RemoveAt(x);
        }
        else
        {
            for (int x = table.Rows.Count - 1; x >= 0; x--)
                if (Convert.ToInt32(table.Rows[x][attrSource]) <=
Convert.ToInt32(valueSouce))
                    table.Rows.RemoveAt(x);
        }
    }
    table.Columns.Remove(attrSource);

    // Получение индексов Джини
    List<PatritionCriteryInfo> patritionCriteryInfoList = GetGains(table);

    // Выбор результирующего атрибута и формирование нового узла
    if (patritionCriteryInfoList.Count > 0)
    {
        // Общий алгоритм
        PatritionCriteryInfo patritionCriteryInfo = patritionCriteryInfoList.First(x
=> x.InformationGain == patritionCriteryInfoList.Max(y => y.InformationGain));

        // Если прирост информации становится < 0.1, то перестаём строить дерево
        if (Math.Abs(patritionCriteryInfo.InformationGain) < 0.1)
        {
            BridgeDecisionTreeItem treeItem = GetNewTreeItem("Gini < 0.1",
patritionCriteryInfo.InformationGain.ToString(), treeItemSource);
            return;
        }

        List<object> distinctValues =
table.GetDistinctValues(patritionCriteryInfo.AttrName);
        distinctValues.Sort();
        if (patritionCriteryInfo.Threshold == null || distinctValues.Count == 1)
        {
            // Для дискретных значений и порогов, у которых число различных значений
= 1
            foreach (object value in distinctValues)
            {
                BridgeDecisionTreeItem treeItem =
GetNewTreeItem(patritionCriteryInfo.AttrName, value.ToString(), treeItemSource);
                GetNextID3Item(table, patritionCriteryInfo.AttrName, value,
treeItem);
            }
        }
        else
        {
            // Для порогов
            BridgeDecisionTreeItem treeItem =
GetNewTreeItem(patritionCriteryInfo.AttrName, string.Format("<= {0}",
patritionCriteryInfo.Threshold), treeItemSource);
            GetNextID3Item(table, patritionCriteryInfo.AttrName,
patritionCriteryInfo.Threshold, treeItem, true);
            treeItem = GetNewTreeItem(patritionCriteryInfo.AttrName,
string.Format("> {0}", patritionCriteryInfo.Threshold), treeItemSource);
            GetNextID3Item(table, patritionCriteryInfo.AttrName,
patritionCriteryInfo.Threshold, treeItem, false);
        }
    }
    else
    {

```

```

        BridgeDecisionTreeItem treeItem = treeItemSource;
        if (table.Columns.Count == 1)
        {
            // Если в таблице остались одни результаты
            List<object> rowValues = table.GetDistinctValues(ResultColumnName);
            foreach (string value in rowValues)
                GetNewTreeItem(table.Columns[0].ColumnName, value, treeItem);
        }
        else
        {
            // Если по каким-то причинам индекс Джини не может посчитаться
            for (int numRows = 0; numRows < table.Rows.Count; numRows++)
                for (int x = 0; x < table.Columns.Count; x++)
                    treeItem = GetNewTreeItem(table.Columns[x].ColumnName,
table.Rows[numRows][x].ToString(), treeItem);
        }
    }

    /// <summary>
    /// Генерация и добавление нового узла дерева в коллекцию
    /// </summary>
    /// <param name="attributeName">Имя атрибута</param>
    /// <param name="value">Значение</param>
    /// <param name="parentTreeItem">Родительский узел</param>
    /// <returns>Новый узел</returns>
    private BridgeDecisionTreeItem GetNewTreeItem(string attributeName, string value,
BridgeDecisionTreeItem parentTreeItem)
    {
        BridgeDecisionTreeItem contractTreeItem = new BridgeDecisionTreeItem()
        {
            AttributeName = attributeName,
            Value = value,
        };
        parentTreeItem.Items.Add(contractTreeItem);
        return contractTreeItem;
    }

    /// <summary>
    /// Получить информацию о критериях разбиения
    /// </summary>
    /// <param name="table">Таблица</param>
    /// <returns>Информация о критериях разбиения</returns>
    private List<PatritionCriterlyInfo> GetGains(DataTable table)
    {
        List<PatritionCriterlyInfo> patritionCriterlyInfoList = new
List<PatritionCriterlyInfo>();

        // Вычисление энтропии
        List<string> results =
table.GetDistinctValues(ResultColumnName).Cast<string>().ToList();
        double entropy = PatritionCriterlyInfo.GetEntropy(results, table.Rows.Count);

        // Получение информации о критериях разбиения
        foreach (DataColumn column in table.Columns)
        {
            // Результаты не участвуют в разбиении
            if (column.ColumnName == ResultColumnName)
                continue;

            if (column.DataType == typeof(string))
            {
                // Для атрибутов строкового типа

```



```

        PatritionCriteryInfo patritionCriteryInfo =
        PatritionCriteryInfo.GetInfoForDiscrete(table, column.ColumnName, entropy, _treeType);
        patritionCriteryInfoList.Add(patritionCriteryInfo);
    }
    else
    {
        List<PatritionCriteryInfo> patritionCriteryInfoes =
        PatritionCriteryInfo.GetInfoForThresholds(table, column.ColumnName, entropy, _treeType);
        patritionCriteryInfoList.AddRange(patritionCriteryInfoes);
    }
}

return patritionCriteryInfoList;
}

/// <summary>
/// Получение списка узлов
/// </summary>
/// <returns>Список узлов</returns>
public List<BridgeDecisionTreeItem> GetItemsAsList()
{
    List<BridgeDecisionTreeItem> listNodes = new List<BridgeDecisionTreeItem>();

    // Вызов рекурсивной функции обхода дерева
    foreach (BridgeDecisionTreeItem buferTreeNode in this.Items)
        TraversalNodes(buferTreeNode, listNodes);

    return listNodes;
}

/// <summary>
/// Рекурсивный обход дерева для получения узлов
/// </summary>
/// <param name="treeNode">Дерево</param>
/// <param name="listNodes">Список узлов</param>
private static void TraversalNodes(BridgeDecisionTreeItem treeNode,
List<BridgeDecisionTreeItem> listNodes)
{
    // Добавление имени дерева в список имён
    listNodes.Add(treeNode);

    // Вызов рекурсивной функции обхода дерева
    foreach (BridgeDecisionTreeItem buferTreeNode in treeNode.Items)
        TraversalNodes(buferTreeNode, listNodes);
}
}

}

using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Security.Cryptography;

namespace DecisionTree.BridgeDecisionTrees
{
    /// <summary>
    /// Информация о критерии разбиения
    /// </summary>
    public class PatritionCriteryInfo
    {
        /// <summary>

```

```

    /// Наименование атрибута
    /// </summary>
    public string AttrName { get; set; }

    /// <summary>
    /// Прирост информации
    /// </summary>
    public double InformationGain { get; set; }

    /// <summary>
    /// Порог (для числовых атрибутов)
    /// </summary>
    public int? Threshold { get; set; }

    /// <summary>
    /// Получить информацию о критерии разбиения для дискретных значений
    /// </summary>
    /// <param name="table">Таблица с результатами</param>
    /// <param name="attr">Атрибут</param>
    /// <param name="treeType">Тип дерева</param>
    /// <param name="entropyResult">Энтропия результатов</param>
    public static PatritionCriteriyInfo GetInfoForDiscrete(DataTable table, string attr,
double entropyResult, BridgeDecisionTreeType treeType)
    {
        // Получение индекса Джини
        List<object> distinctValues = table.GetDistinctValues(attr);
        double gain = 0.0;
        foreach (object value in distinctValues)
        {
            List<string> results = table.GetResultsForAttr(attr, value);
            double entropy = GetEntropy(results);
            gain += Convert.ToDouble(results.Count)/table.Rows.Count*entropy;
        }

        PatritionCriteriyInfo patritionCriteriyInfo = new PatritionCriteriyInfo() {AttrName
= attr,};
        switch (treeType)
        {
            case BridgeDecisionTreeType.ID3:
                patritionCriteriyInfo.InformationGain = entropyResult - gain;
                break;
            case BridgeDecisionTreeType.C45:
                double splitInfo = 0.0;
                foreach (object value in distinctValues)
                {
                    List<string> results = table.GetResultsForAttr(attr, value);
                    double resultPart = Convert.ToDouble(results.Count) /
table.Rows.Count;
                    splitInfo += - resultPart * Math.Log(resultPart, 2);
                }
                if (splitInfo > 0)
                    patritionCriteriyInfo.InformationGain = - gain / splitInfo;
                break;
            default:
                throw new ArgumentException();
        }

        return patritionCriteriyInfo;
    }

    /// <summary>
    /// Получить информацию о критерии разбиения для значений, которые надо разбить на
пороги
    /// </summary>

```

```

    /// <param name="table">Таблица с результатами</param>
    /// <param name="attr">Атрибут</param>
    /// <param name="entropyResult">Энтропия результатов</param>
    /// <param name="treeType">Тип дерева</param>
    public static List<PatritionCriteryInfo> GetInfoForThresholds(DataTable table,
string attr, double entropyResult, BridgeDecisionTreeType treeType)
    {
        List<PatritionCriteryInfo> patritionCriteryInfoList = new
List<PatritionCriteryInfo>();
        List<object> distinctValues = table.GetDistinctValues(attr);
        distinctValues.Sort();

        // Одно значение не имеет смысла разбивать на пороги
        if (distinctValues.Count == 1)
        {
            PatritionCriteryInfo patritionCriteryInfo = GetInfoForDiscrete(table, attr,
entropyResult, treeType);
            patritionCriteryInfoList.Add(patritionCriteryInfo);
            return patritionCriteryInfoList;
        }

        // Разбиваем на пороги
        distinctValues.Remove(distinctValues.Last());
        foreach (int distinct in distinctValues)
        {
            // Преобразуем данные в два типа: значение, меньшее или равное атрибуту, и
значение, большее атрибута
            DataTable tableForInt = table.Copy();
            for (int n = 0; n < tableForInt.Rows.Count; n++)
                if (Convert.ToInt32(tableForInt.Rows[n][attr]) <= distinct)
                    tableForInt.Rows[n][attr] = distinct;
                else
                    tableForInt.Rows[n][attr] = distinct + 1;

            // Вычисляем данные для каждого из порогов
            PatritionCriteryInfo patritionCriteryInfo = GetInfoForDiscrete(table, attr,
entropyResult, treeType);
            patritionCriteryInfo.Threshold = distinct;
            patritionCriteryInfoList.Add(patritionCriteryInfo);
        }

        return patritionCriteryInfoList;
    }

    /// <summary>
    /// Вычисление энтропии
    /// </summary>
    /// <param name="results">Список результатов</param>
    /// <param name="rowCount">Число результатов (для вычисления энтропии для
результатов, а не для атрибутов)</param>
    /// <returns>Энтропия</returns>
    public static double GetEntropy(List<string> results, int rowCount = 0)
    {
        double entropy = 0.0;
        foreach (string result in results)
        {
            double countResult = results.Count(x => x == result);
            double resultPart = 0.0;
            if (rowCount == 0)
                resultPart = countResult / results.Count();
            else
                resultPart = countResult / rowCount;
            entropy += -resultPart * Math.Log(resultPart, 2);
        }
    }

```

```

        return entropy;
    }
}

using System.Collections.Generic;
using System.Data;
using System.Linq;
using DecisionTree.BridgeDecisionTrees;

namespace DecisionTree
{
    /// <summary>
    /// Методы расширения
    /// </summary>
    public static class ExtensionMethods
    {
        /// <summary>
        /// Получить уникальные значения из DataTable
        /// </summary>
        /// <param name="table">DataTable</param>
        /// <param name="columnName">Имя столбца</param>
        /// <returns>Список уникальных значений</returns>
        public static List<object> GetDistinctValues(this DataTable table, string
columnName)
        {
            List<object> distincts = table.AsEnumerable().Select(x =>
x[columnName]).Distinct().ToList();
            return distincts;
        }

        /// <summary>
        /// Получить результаты для атрибута из DataTable
        /// </summary>
        /// <param name="table">DataTable</param>
        /// <param name="attr">Атрибут</param>
        /// <param name="value">Значение атрибута</param>
        /// <returns>Список результатов</returns>
        public static List<string> GetResultsForAttr(this DataTable table, string attr,
object value)
        {
            List<string> result = new List<string>();
            for (int x = 0; x < table.Rows.Count; x++)
                if (table.Rows[x][attr].Equals(value))
                    result.Add((string) table.Rows[x][BridgeDecisionTree.ResultColumnName]);

            return result;
        }
    }
}

using System.Collections.Generic;
using System.Windows;
using DecisionTree.BridgeDecisionTrees;
using GraphSharp.Controls;
using QuickGraph;

namespace DecisionTree.Visualize
{
    /// <summary>
    /// Логика взаимодействия для VisualizeTreeWindow.xaml
    /// </summary>
    public partial class VisualizeTreeWindow : Window
    {

```

```

    /// <summary>
    /// Граф
    /// </summary>
    BidirectionalGraph<BridgeDecisionTreeItem, Edge<BridgeDecisionTreeItem>> _graph;

    /// <summary>
    /// Слой графа
    /// </summary>
    GraphLayout<BridgeDecisionTreeItem, Edge<BridgeDecisionTreeItem>,
    BidirectionalGraph<BridgeDecisionTreeItem, Edge<BridgeDecisionTreeItem>>> _graphLayout = new
    GraphLayout<BridgeDecisionTreeItem, Edge<BridgeDecisionTreeItem>,
    BidirectionalGraph<BridgeDecisionTreeItem, Edge<BridgeDecisionTreeItem>>>();

    /// <summary>
    /// Список вершин
    /// </summary>
    List<BridgeDecisionTreeItem> _vertexes = new List<BridgeDecisionTreeItem>();

    /// <summary>
    /// Список рёбер
    /// </summary>
    List<Edge<BridgeDecisionTreeItem>> _edges = new
    List<Edge<BridgeDecisionTreeItem>>();

    /// <summary>
    /// Конструктор
    /// </summary>
    /// <param name="bridgeDecisionTree">Дерево</param>
    public VisualizeTreeWindow(BridgeDecisionTree bridgeDecisionTree)
    {
        InitializeComponent();

        List<BridgeDecisionTreeItem> bridgeDecisionTreeItemList =
        bridgeDecisionTree.GetItemsAsList();
        foreach (BridgeDecisionTreeItem bridgeDecisionTreeItem in
        bridgeDecisionTreeItemList)
        {
            _vertexes.Add(bridgeDecisionTreeItem);
            if (bridgeDecisionTreeItem.Parent is BridgeDecisionTreeItem)
                _edges.Add(new
                Edge<BridgeDecisionTreeItem>((BridgeDecisionTreeItem)bridgeDecisionTreeItem.Parent,
                bridgeDecisionTreeItem));
        }

        _graph = new BidirectionalGraph<BridgeDecisionTreeItem,
        Edge<BridgeDecisionTreeItem>>();
        _graphLayout = new GraphLayout<BridgeDecisionTreeItem,
        Edge<BridgeDecisionTreeItem>, BidirectionalGraph<BridgeDecisionTreeItem,
        Edge<BridgeDecisionTreeItem>>>();
        _graphLayout.LayoutAlgorithmType = "EfficientSugiyama";
        _graphLayout.OverlapRemovalAlgorithmType = "FSA";
        _graphLayout.HighlightAlgorithmType = "Simple";
        _graph.AddVertexRange(_vertexes);
        _graph.AddEdgeRange(_edges);
        _graphLayout.Graph = _graph;

        Zoom.Content = _graphLayout;
    }
}

using System;
using System.Collections.Generic;

```

```

using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using DecisionTree.BridgeDecisionTrees;
using DecisionTree.Visualize;
using System.Xml;
using System.Runtime.Serialization;
using System.Xml.Serialization;
using System.IO;

namespace DecisionTree
{
    /// <summary>
    /// Логика взаимодействия для MainWindow.xaml
    /// </summary>
    public partial class MainWindow
    {
        /// <summary>
        /// Исходные данные
        /// </summary>
        List<BasicData.BasicDataItem> _basicData { get; set; }

        /// <summary>
        /// Конструктор
        /// </summary>
        public MainWindow()
        {
            InitializeComponent();
            _basicData = BasicData.BasicDataItem.GetBasicData();
            DataGrid.ItemsSource = _basicData;
        }

        /// <summary>
        /// Нажатие на кнопку "Дерево ID3"
        /// </summary>
        private void ID3Button_OnClick(object sender, RoutedEventArgs e)
        {
            WaitWindow waitWindow = new WaitWindow();
            waitWindow.Show();
            try
            {
                BridgeDecisionTree bridgeDecisionTree = new BridgeDecisionTree(_basicData,
                BridgeDecisionTreeType.ID3, (bool)ThresholdCheckBox.IsChecked);
                VisualizeTreeWindow window = new VisualizeTreeWindow(bridgeDecisionTree);
                window.Show();
            }
            finally
            {
                waitWindow.Close();
            }
        }

        /// <summary>
        /// Нажатие на кнопку "Дерево C4.5"
        /// </summary>
        private void C45Button_OnClick(object sender, RoutedEventArgs e)
        {
            WaitWindow waitWindow = new WaitWindow();
            waitWindow.Show();
            try
            {

```

```

        BridgeDecisionTree bridgeDecisionTree = new BridgeDecisionTree(_basicData,
BridgeDecisionTreeType.C45, (bool)ThresholdCheckBox.IsChecked);
        VisualizeTreeWindow window = new VisualizeTreeWindow(bridgeDecisionTree);
        window.Show();
    }
    finally
    {
        waitWindow.Close();
    }
}
}
}
}

```