Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Институт естественных и точных наук
Факультет математики, механики и компьютерных технологий
Кафедра прикладной математики и программирования
Направление подготовки Прикладная математика

РАБОТА ПРОВЕРЕНА
Рецензент,

_____
«____»_____2018г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой, д.ф.-м.н., доцент
_____А.А. Замышляева
«____»_____2018 г.

Разработка системы искусственного интеллекта для игры "Покер"

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ–01.03.04.2018.78.ПЗ ВКР

Руководитель работы, к.т.н., доцент
_____/Е.Ю. Алексеева
«____»_____2018 г.
Автор работы
Студент группы ЕТ-413
_____/ А.С. Саратовцев
«____»_____2018 г.
Нормоконтролер, к.э.н., доцент
_____/Д.А. Дрозин
«____»_____2018 г.

Челябинск 2018

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Покер (англ. *poker*) — карточная игра, цель которой — выиграть ставки, собрав как можно более высокую покерную комбинацию, используя 5 карт, или вынудив всех соперников прекратить участвовать в игре. Игра идёт с полностью или частично закрытыми картами. Конкретные правила могут варьироваться в зависимости от разновидности покера. Обобщающими элементами всех разновидностей покера являются комбинации и наличие торговли в процессе игры. [6,7]

Покер – это игра недостаточной информации, но не все игры недостаточной информации созданы одинаковыми. От игры к игре имеются различные степени доступности информации. [5]

Самой популярной игрой является безлимитный Техасский Холдем. В рамках данной работы обсуждается только этот вид покера. Ввиду того, что игрок не знает карты своих противников, покер является игрой с неполной информацией, как и многие другие карточные игры, в отличие от, например, шахмат, в которых оба игрока видят положение всех фигур на доске и могут точно сказать какой ход верный, а какой нет.

Покер не является рулеткой или блэк-джеком, где все вероятности известны заранее и от уровня игры ничего не зависит – удача и только. В покере игра ведется между людьми.

Выигрывает тот, кто играет лучше. Можно сказать, что покер – это игра ошибок. Чем больше допущено ошибок, тем меньше выигрыш или даже больше проигрыш.

Человеку свойственно ошибаться, входить в азарт, не контролировать эмоции. Лучший игрок в покер – это тот игрок, который постоянно контролирует ситуацию, анализирует происходящее и делает выводы. Возникает идея создать программу – идеального игрока, никогда не ошибающегося, не знающего, что такое эмоции и азарт, принимающего только верные решения, а значит постоянно выигрывающего.

История развития ИИ для игры в покер насчитывает более 30 лет, но самые выдающиеся достижения произошли буквально в последние 3 года.

Президент Федерации спортивного покера России Дмитрий Лесной в 2009 году на своём сайте высказал мнение по поводу искусственного интеллекта и игры покер: “Уже есть и шахматный искусственный интеллект, который обыгрывает Каспарова и Крамника и этим практически закрывает шахматы. Ведь что это за спорт, в котором чемпионы мира соревнуются между собой, но железяку все равно догнать не могут? Немного унизительно, правда? Так вот, для покера еще не создана такая программа, которая победила бы человека, при том, что на разработку тратятся очень не маленькие ресурсы. Это говорит о сложности мастерства игры в покер”. [10]

Первые программы и алгоритмы игры в покер появились еще в 80-х годах, например, система Orac от Mike Caro, написанная им в 1984 и продемонстрированная на турнире Stratosphere. В 1991 году в Университете

Альберты (Канада) была создана первая в мире исследовательская группа, посвященная развитию ИИ для покера. В 1997 году эта группа продемонстрировала свою систему Loki, первую успешную и значимую реализацию ИИ для покера. Loki играл на уровне немного хуже среднего человеческого игрока, но эта была значимая веха для всего исследовательского направления. В 2000-х годах произошел сдвиг парадигмы написания ИИ для покерных ботов. Исследователи отошли от подходов к покеру, вдохновленных успехами Deep Blue в шахматах (успешно одолевшему Гарри Каспарова в 1996 году), к полноценной методологии и постановке задач моделирования сразу для покера.

В 2015 году Университет Альберты представил свою систему Cepheus, которая буквально «решила» один из видов покера — лимитный Heads-up покер (упрощенная разновидность, порядка 1018 игровых ситуаций). Это значимая веха в развитии AI, так как это единственная на данный момент игра с неполной информацией, имеющая полное оптимальное решение. Достичь этого удалось, поставив Cepheus играть сам с собой в течение двух месяцев (похожим образом обучался и AlphaGo, победивший чемпиона мира по игре Го).

В 2017 году произошло сразу два важных события в мире покерных ботов. Университет Альберты представил алгоритм DeepStack для игры в безлимитный Heads-Up покер. Основанный на глубоких нейронных сетях алгоритм успешно одолел множество человеческих соперников, включая профессиональных игроков и аналогично AlphaGo смог «обучиться» имитировать человеческую интуицию, продолжительно играя множество партий сам с собой.

Самое значимое событие 2017 года в мире покерных ботов, а возможно и ИИ в целом. Система Libratus от Университета Карнеги-Меллон с уверенностью одолела профессиональных игроков в покер — команду, состоящую из лучших мировых игроков в безлимитный Heads-Up покер. По их оценке, алгоритм был настолько хорош, что казалось словно он мухлюет и видит карты соперников. Матчи шли в реальном времени в течение 20-дневного турнира, а действия алгоритма считались на Питтсбургском суперкомпьютере. [6]

Главной целью данной работы является разработка системы искусственного интеллекта для игры покер. Необходимо исследовать математический аппарат покера, существующие стратегии игры в покер, выбрать подходящую для алгоритмизации стратегию, доработать и запрограммировать ее.

1. ПРАВИЛА ПОКЕРА

В холдеме баттоном (от англ. «button» - «кнопка») отмечается номинальный дилер раздачи. Перед началом раздачи игрок, следующий по часовой стрелке после баттона, ставит малый блайнд – первую обязательную ставку. Следующий игрок ставит большой блайнд, который обычно в 2 раза больше малого блайнда.

После этого каждый игрок получает по две закрытых карты. Игроки делают свои ходы в порядке очереди по часовой стрелке, начиная с игрока в позиции «под прицелом» (англ. under the gun). Это позиция игрока, сидящего по часовой стрелке от большого блайнда.

Действия игрока. В холдеме, как и в большинстве других разновидностей покера, игрок может сделать «фолд» - сбросить карты, «бет» сделать ставку, «колл» - принять ставку или «рейз» - повысить ставку. Доступность тех или иных действий зависит от действий предыдущих игроков. У любого игрока всегда есть возможность сделать фолд, то есть сбросить свои карты и отказаться от борьбы за банк. Если до Вас никто не сделал бет (ставку), то Вы можете сделать либо чек (отказаться от ставки, но не сбросить карты), либо ставку. Если один из игроков сделал ставку, то последующие игроки могут сделать фолд, колл или рейз. «Сделать колл» означает добавить в банк количество фишек, необходимое для уравнивания ставки предыдущего игрока. «Сделать рейз» означает добавить в банк количество фишек, превышающее ставку предыдущего игрока.

Игра в холдем состоит из четырёх кругов торговли:

1. **Префлоп.** После того, как игроки получили свои закрытые карты, каждый из них может продолжить играть, сделав колл или рейз большого блайнда. Первым ходит игрок слева от большого блайнда. Каждый раунд торгов продолжается пока все активные игроки (которые не сделали фолд) не сделают равные ставки в банк.
2. **Флоп.** Теперь на борд сдаются три карты лицом вверх. Они называются «флопом». В холдеме три карты флопа являются общими картами, доступными всем игрокам, продолжающим игру в этой раздаче. Торги на флопе начинаются с первого активного игрока по часовой стрелке от баттона. Варианты действий практически такие же, как и на пре-флопе. Однако, если ни один из предыдущих игроков не сделал бет, игрок может сделать чек, передав ход следующему игроку по часовой стрелке.
3. **Тёрн.** После завершения торгов на флопе лицом вверх сдается «терн». Торн – это четвертая общая карта в холдеме Происходит еще один раунд торгов, начиная с активного игрока, находящегося по часовой стрелке от баттона.
4. **Ривер.** После завершения торгов на терне лицом вверх сдается «ривер». Ривер – это пятая общая карта в холдеме. Раунд торгов снова начинается с активного игрока, находящего по часовой стрелке от баттона, при этом применяются такие же правила торгов, что на флопе и терне. [8]

Вскрытие карт. Если после завершения последнего раунда торгов в раздаче осталось несколько активных игроков, то первым карты открывает игрок,

который последним сделал бет или рейз. Если же в последнем раунде не было ставок, то первым открывает карты активный игрок, сидящий по часовой стрелке от баттона. Банк забирает игрок с лучшей комбинацией из пяти карт. Если несколько игроков собрали равносильные комбинации, то банк делится между ними поровну. В холдеме ни одна из мастей не имеет преимущества над другими. После передачи банка в руки игрока, победившего в раздаче, начинается следующая раздача. Баттон перемещается к следующему по часовой стрелке игроку, ставятся блайнды и сдаются новые закрытые карты.

Отметим, что есть кеш игры, SNG (sit and go) турниры (6 – 10 человек), и турниры с большим количеством участников (от 45 и до нескольких тысяч). Они отличаются структурой. В кеш игре блайнды остаются неизменными на протяжении всей игры. В турнирах блайнды увеличиваются через определенное время, например, каждые 6 минут.

Из-за этого факта стратегия игры в кеш будет сильно отличаться от стратегии игры в SNG. В рамках диплома рассматривается кеш игра.

2. РЕШЕНИЯ СИСТЕМ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА ДЛЯ ИГРЫ «ПОКЕР»

2.1. Типы систем искусственного интеллекта для игры покер

Существующие разработки систем искусственного интеллекта можно разделить на два типа:

1. Исследовательские. Их целью является ответ на вопрос: может ли компьютерная программа обыграть человека? Основной упор делается на разработку искусственного интеллекта покерного бота. Здесь работа ведется уже много лет группами из разных университетов мира.
2. Прикладные. Под покерной системой искусственного интеллекта понимается комплекс программного обеспечения, которое самостоятельно способно играть на существующих покерных сайтах. Задача считать данные из покер клиента, проанализировать их, принять решение о необходимом действии и «нажать» нужные кнопки.

2.1.1 Исследовательские

В начале обзора уместно сказать, что все системы искусственного интеллекта такого типа играют исключительно heads-up холдем, т.е. один на один.

A1. Monash BPP

The Monash Bayesian Poker project – основан Кевином Корбом в 1993 году. С тех пор его разработкой занимались в основном студенты. Умеет играть в heads-up limit Holdem на уровне начинающего игрока. Модуль принятия решений основан на байесовской нейронной сети. Накапливает статистику игры противника (сыгранные раздачи), моделирует его поведение.

A2. Polaris

Программа, играет только лимитный холдем один на один (который в силу ограниченных ставок значительно больше зависит от математики и подходит для алгоритмизации, чем безлимитный холдем). Разработчики – группа из университета Альберты (University of Alberta Computer Poker Research Group). Их цель – создание компьютерной программы, играющей лучше, чем человек.

Группа основана в 1995 году и по сей день активно работает. В 2007 г. данная программа играла дубликатный матч против двух известных покерных профи: Фила Лаака (Phil "The Unabomber" Laak) и Али Эслами (Ali Eslami). Матч состоял из четырех сессий по 500 рук в каждой. Причем дубликатная система предусматривала раздачу одинаковых карт как компьютеру против человека, так и тех же карт человеку против компьютера, поэтому влияние случайности в игре было минимальным. В первой сессии Polaris выиграл, однако совсем немного, и

это было засчитано как статистическая ничья. Во второй сессии Polaris нанес людям ощутимое поражение. После анализа сыгранных в этих сессиях за первый день рук, игроки нашли некоторые особенности игры Polaris, и в оставшихся двух сессиях победили его, хотя результат и нельзя назвать разгромным. По их мнению программа играла очень сильно.

Основная модель действий:

- Накапливание информации для тренировки (история сыгранных рук)
- Создание модели поведения противника
- Поиск наилучших действий против поведения противника.

Искусственный интеллект использует огромный математический аппарат, в частности многое основывается на Равновесии Неша.

Равновесие Нэша (англ. *Nash equilibrium*) названо в честь Джона Форбса Нэша — в теории игр называется тип решений игры двух и более игроков, в котором ни один участник не может увеличить выигрыш, изменив своё решение в одностороннем порядке, когда другие участники не меняют решения. Такая совокупность стратегий выбранных участниками и их выигрыши называются равновесием Нэша. [9]

А3. Libratus

Libratus состоит из трех основных частей. Для первых кругов покера используется модуль, который рассматривает игру как абстракцию. Вместо того, чтобы учитывать все точки принятия решений, число которых достигает 10^{161}, он упрощает игру, при этом максимально учитывая стратегические аспекты оригинальной игры. Так, Libratus округляет ставки и не делает различий между похожими карточными комбинациями, например флэшем с королем во главе и флэшем, где старшая карта — дама. После создания абстракции, компьютер разрабатывает плановую стратегию поведения для первых кругов, а также очень приблизительную стратегию для следующих этапов.

Чтобы научиться делать это, Libratus играл против самого себя, используя измененную версию алгоритма Monte Carlo Counter-factual Regret Minimization (MCCFR). С его помощью для каждого действия вычислялось значение сожаления — то, насколько игрок сожалеет о том, что он не сделал определенный шаг в прошлом. Во время симуляции MCCFR выбирал «исследователя», который должен был анализировать все возможные действия и постоянно обновлять значение сожаления. При этом его противник играл согласно стратегии, которая выстаивается на основе уже имеющихся данных. В конце «исследователю» давалась награда за каждое действие, благодаря которой он понимал, какой ход был хорошим, а какой — нет. После каждой партии игроки менялись ролями. В классическом варианте компьютер обычно исследует все гипотетические действия, чтобы выяснить размер награды за них; здесь же он пропускал «неинтересные» ходы, которые имели низкое значение сожаления, что позволило быстрее усовершенствовать его работу.

Для следующих этапов игры использовался второй модуль Libratus. Он создавал детальную стратегию для конкретного этапа игры, руководствуясь при этом плановой стратегией, разработанной в начале. Каждый раз, когда противник совершал не предусмотренное системой ИИ действие, она разыгрывала «мини-игру», где учитывался ход соперника. Это позволяло корректировать стратегию в режиме реального времени.

Третья часть Libratus улучшала исходную стратегию алгоритма. Обычно для этого строится модель поведения противника, которая учитывает его возможные ошибки. Однако Браун и Сандхолм использовали данные о ставках. Днем компьютер следил, какие ставки чаще всего делают другие игроки, а ночью вычислял возможные варианты развития событий с учетом этих данных.

2.1.2 Прикладные

OpenHoldem (далее OH) – набор инструментов с открытым исходным кодом для построения покерного бота. Это фреймворк для считывания данных о текущей игре и программируемый логический движок для Техасского Холдема. Он может работать с любым покер румом, но требует тонкой настройки в каждом конкретном случае.

Для получения игровой ситуации (карты, стеки и т.д.) из покерного клиента OH использует распознавание образов. OpenHoldem предоставляет язык скриптов для написания стратегии игры, также можно писать на C++.

OpenHoldem – фреймворк. Это значит, что нужно предоставить ему несколько вещей, для правильной работы. Первое – «научить» как в конкретном казино отображаются карты, стеки, ставки и т.д. для получения информации о текущей игровой ситуации. Второе – описать логику игры, что делать и в какой ситуации.

Таким образом OpenHoldem дает большой набор инструментов для построения собственного покерного бота, но построить по-настоящему гибкую логику принятия решения на нем не получиться, т.к. описать логику, хотя бы немного приближенную к логике человеческого мозга, скриптами невозможно. Из-за этого было принято решение разработать для игры комплекс программного обеспечения с «нуля».

2.2. Подходы к созданию стратегий

За более чем 30-летнюю историю развития покерных ботов было создано несколько семейств подходов в разработке стратегий в покер.

2.2.1 Классические подходы

Одним из наиболее простых в реализации и наименее затратных по времени является экспертная система. По сути это набор фиксированных IF-THEN правил, который относит игровую ситуацию к одному из заранее определенных классов. В зависимости от силы собранной комбинации система предлагает принять одно из множества доступных решений.

Также эту задачу можно решать сугубо математическим методом и в каждый момент времени рассчитывать оптимальное решение с точки зрения равновесия по Нэшу. Однако, решение будет оптимальным в том случае, если решения остальных участников тоже оптимальны. Поиск такого решения является ресурсозатратным, поэтому на практике его можно использовать только наряду с большим количеством ограничений в правилах. Например, в лимитном техасском холдеме для двух агентов или при возникновении определенных игровых ситуаций.

2.2.2 Подходы на основе машинного обучения

B1. Дерево поиска Монте-Карло

Более эффективной является эксплуатационная стратегия, которая разделяет оппонентов на кластеры и против каждого кластера реализуется контр-стратегия. Большинство хороших игроков в покер используют именно такой подход. Но, в отличие от человека, у компьютера есть преимущество в том, что он может перебрать огромное кол-во исходов игры и при правильном прогнозировании поведения соперников принять максимально выгодное решение с точки зрения математического ожидания. Для прогнозирования поведения оппонентов в таком случае может хорошо помочь сбор статистики игр в прошлых матчах и реализация алгоритмов машинного обучения. К сожалению авторов алгоритмов, перебрать все возможные исходы событий в большинстве игровых ситуаций не получится даже у мощных компьютеров, поэтому нужно использовать алгоритмы оптимизации, такие как Monte Carlo Tree Search.

Метод Монте-Карло это алгоритм принятия решений, часто используемый в играх в качестве основы искусственного интеллекта. Сильное влияние он оказал на программы для игры в Го, хотя находит свое применение и в других играх, как настольных, так и обычных компьютерных (например Total War: Rome II). Так же, стоит отметить, что метод Монте-Карло используется в программе AlphaGo, победившей го-профессионала 9-го дана Ли Седоля в серии из 5 игр.

После публикации версии алгоритма Монте-Карло под названием Upper Confidence bound applied to Trees (UCT) в 2006-м году, программы для игры в Го сильно усилили свои позиции и достигли значительных успехов в игре против человека.

Основой алгоритма UCT является решение задачи многоруких бандитов. В частности используется алгоритм Upper-Confidence-Bound (UCB).

Задача звучит так: есть набор игровых автоматов, каждый со своей вероятностью выигрыша. Требуется определить автомат, который принесет больший выигрыш. [19]

Алгоритм UCB:

1. Инициализация. Сыграть на каждом автомате один раз.

2. На каждой следующей итерации выбрать автомат с максимальным значением величины:

—— ——, (2.1)

где - средний выигрыш *i*-го автомата;

n - количество игр во всех автоматах;

n_i - количество игр сыгранных в *i*-м автомате.

На практике для поиска в дереве ходов настольных игр часто используется модификация формулы UCB. Например, такая:

— —, (2.2)

где – количество побед *i*-го узла;

– количество посещений *i*-го узла;

n – количество посещений всех соседних узлов;

c – константа, используемая для установки нужного баланса между шириной и глубиной поиска. Чем она больше, тем более глубокий будет поиск.

Алгоритм UCT (Upper Confidence bound applied to Trees) использует подход UCB для поиска по дереву. Работа алгоритма содержит 4 фазы:

1. **Выбор**. Каждая позиция рассматривается как задача многорукого бандита. Узлы на каждом этапе выбираются согласно алгоритму UCB. Эта фаза действует до тех пор, пока не будет найден узел в котором еще не все дочерние узлы имеют статистику побед. На рисунке первое значение в узле это количество побед, второе общее количество игр в этом узле.
2. **Расширение**. Когда алгоритм UCB больше не может быть применим, добавляется новый дочерний узел.
3. **Симуляция**. Из созданного на предыдущем этапе узла запускается игра со случайными или, в случае использования эвристик, не совсем случайными ходами. Игра идет до конца партии. Здесь важна только информация о победителе, оценка позиции не имеет значения.
4. **Обратное распространение**. На этом этапе информация о сыгранной партии распространяется вверх по дереву, обновляя информацию в каждом из ранее пройденных узлов. Каждый из этих узлов увеличивает показатель количества игр, а узлы, совпадающие с победителем, увеличивают также и количество побед. В конце алгоритма выбирается узел, посещенный наибольшее количество раз. [17]

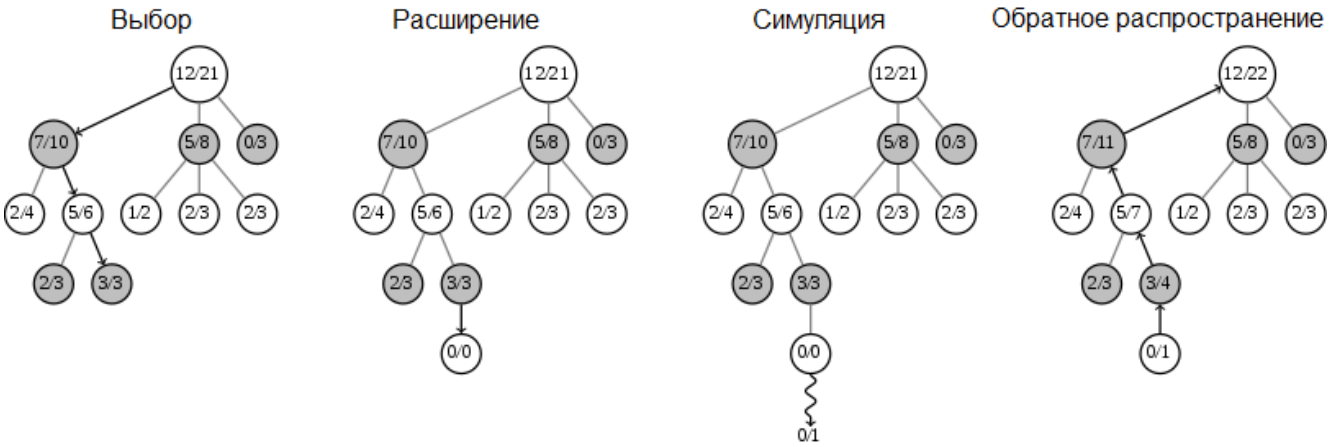

Рисунок 2.1 – Шаги поиска по дереву Монте-Карло

UCT не всегда выбирает только самый лучший ход, но так же периодически исследует и менее успешные узлы. Второй параметр формулы медленно растет для тех узлов, которые посещаются не так часто. И в итоге на каком-то этапе алгоритм выберет именно такой ход в качестве предпочтительного. Если ход оправдал ожидания, в следующий раз он будет выбран с большей вероятностью.

По сравнению с другими алгоритмами для поиска оптимальных ходов, UCT обладает следующими преимуществами:

1. UCT может быть безболезненно остановлен на любом этапе работы. И на момент остановки он просчитает равномерно все варианты ходов из корневого узла (рисунок 2.2).

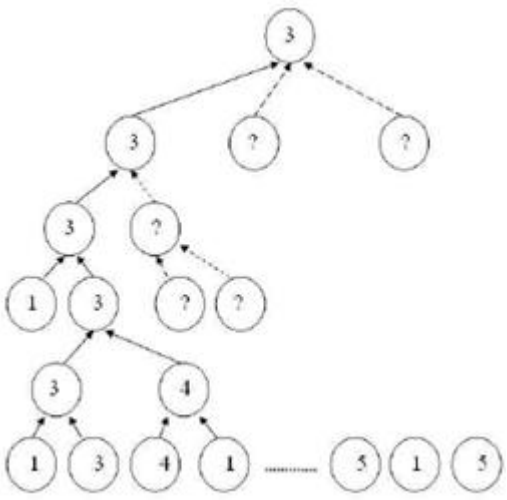

Рисунок 2.2 – Пример остановки алгоритма

2. Дерево растет асимметрично, исследуя предпочтительные ходы глубже остальных. Таким образом, достигается большая эффективность в сравнении с другими алгоритмами в играх со значительным количеством вариантов для перебора.

Т.к. ходы, выбранные случайно в большинстве своем бессмысленны и не имеют какого-то общего направления, то для улучшения работы алгоритма

используются различные эвристические методы, основанные на информации о конкретной игре. Один из таких методов это применение шаблонов (рисунок 2.3).

Рисунок 2.3 – Пример шаблона для разрезания камней в Го

Шаблоны могут использоваться как на этапе выбора хода в дереве, так и на этапе симуляции. Зачастую такие шаблоны ищутся где-то в окрестностях последнего сделанного хода. Это делается, потому, что текущий ход не редко является ответом на предыдущий и скорее всего, будет сделан где-то поблизости.

В большинстве случаев шаблоны пишутся вручную, но есть так же и примеры автоматической генерации.

Алгоритм UCT может быть одновременно запущен в нескольких потоках. Вот некоторые способы параллелизации вычислений:

- Распараллеливание узлов, одновременная симуляция игр из одного и того же узла.
- Распараллеливание корня, построение независимых деревьев.
- Распараллеливание дерева, совместное построение одного дерева несколькими потоками.

В целом алгоритм с учетом дополнительных модификаций показывает неплохие результаты игры. В пользу этого утверждения говорит, то, что все современные Го-программы используют именно его. [19]

В2. Реализация стратегий на основе нейросетей

Наконец, можно подойти к созданию стратегии еще более абстрактно и реализовать нейросеть, на входе у которой будут параметры игровой ситуации, а на выходе — множество возможных решений. К минусам этого подхода стоит отнести то, что в этом случае потребуется большой набор данных для обучения. Этот минус можно нивелировать, запустив нейросеть играть саму с собой на подобие подхода AlphaGo, но нужно быть готовым к более чем одним суткам обучения и моделирования.

DeepStack - универсальный алгоритм для большого класса последовательных несовершенных информационных игр. Состояние игры в покер можно разделить на частную информацию игроков, руки с двумя картами, лицевой стороной вниз, и общее состояние, состоящее из карт, лежащих на столе и последовательности действий ставок, сделанных игроками. Возможные последовательности открытых состояний в игре образуют дерево с каждым общим состоянием, имеющим связанное с ним поддерево (рисунок 2.4).

Часть открытого дерева представлена на рисунке 4. Узлы представляют собой состояния, тогда как ребра представляют собой действия:

- красный и бирюзовый цвета – показывают действия игрока;
- зеленые – открытые карты.

Игра заканчивается на терминальных узлах, показанных как чип со связанным значением. Для конечных узлов, где игрок не сбросил карты, игрок, чьи личные карты образуют более сильную покерную руку, получает значение состояния.

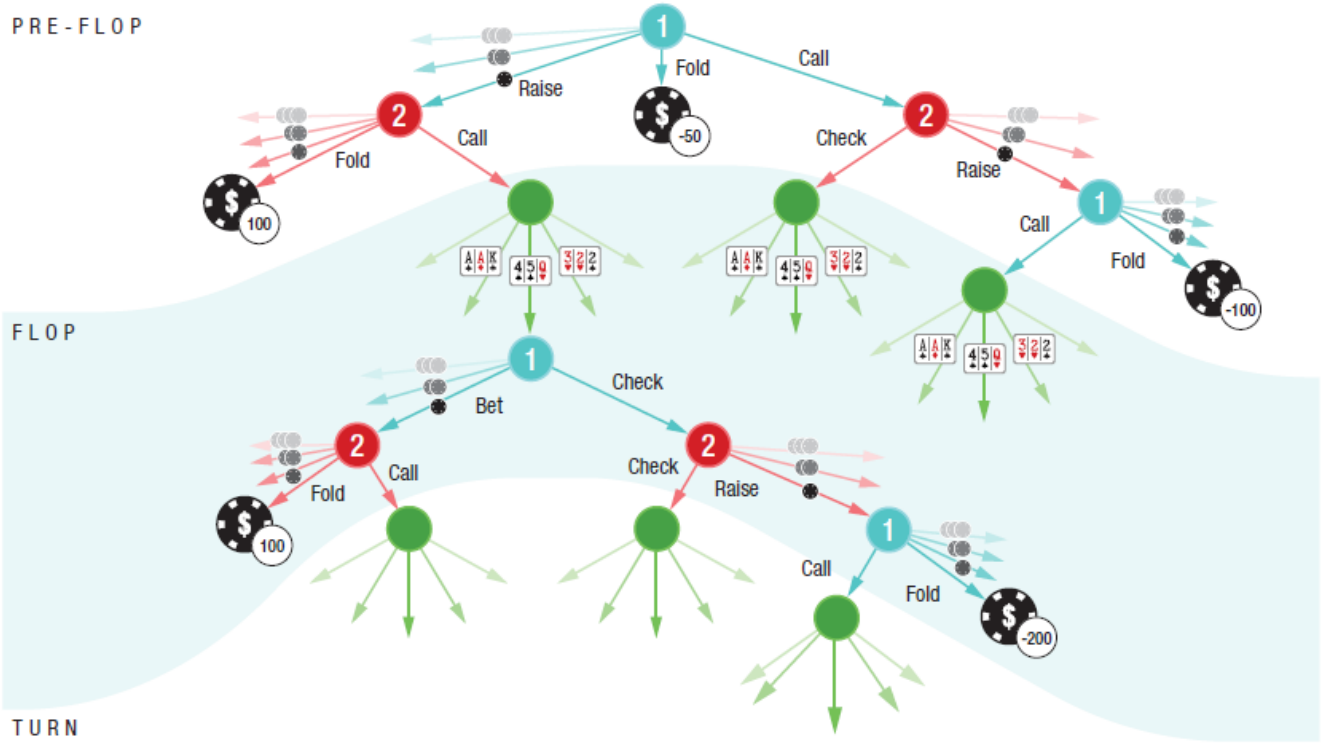

Рисунок 2.4 – Часть открытого дерева использованного в алгоритме DeepStack

На вход нейросети передаются данные о размере ставок, раскрытых картах и информация об игроках, после чего их данные преобразуются в представление «кластеров карточных рук». Эта информация подается на вход 7-слойной нейронной сети, после чего ее результат дополнительно пост-обрабатывается для удовлетворения теоретико-игровым критериям нулевой суммы. [21]

3. ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ ДЛЯ ИГРЫ «ПОКЕР»

3.1. Что должен уметь искусственный интеллект

Искусственный интеллект – должен считывать информацию из покер клиента, анализировать ее, принимать решение, и выполнять его. Логично разделить программу на три части:

Ввод данных в этой части происходит построение модели покерного стола. Вводным устройством является само программное обеспечение покер-клиента, в том числе все окна, лог файлы и история партий, а также внутренний режим, поддерживаемый исполняемым файлом. Цель этапа ввода - построение точной модели стола в каждый момент времени - ваши закрытые карты, имена и размеры стеков ваших оппонентов, текущие ставки, и так далее.

Анализ – протекает независимо от двух других этапов. Цель: принять модель стола, составленную на этапе ввода и определить необходимое действие: сбросить карты, сделать ставку, поднять ее или принять.

Вывод данных – После того, как на этапе обработки принято решение, вступает в действие этап вывода. Это выражается в нажатии правильных кнопок на экране, или совершении необходимых действий в игре.

3.2. Ввод данных

Цель этапа – точная модель покерного стола в каждый момент времени.

Вводное устройство – покерный клиент.

Результат этапа – модель стола в виде структуры данных, понятной программе.

Существует множество способов считать данные из покерного клиента: прямое чтение нужных параметров из памяти, инъекция кода в адресное пространство клиента, разбор чата дилера, распознавание образов.

Для считывания данных из покерного клиента был выбран способ распознавания образов из окна покерного клиента. В качестве окна покерного клиента было взято окно покерного клиента «PokerStars» (рисунок 3.1).

Окно покерного клиента можно поделить на несколько условных областей, в каждой из которых отображается часть информации о состоянии игры в реальном времени:

1) Личные карты;
2) Общие карты (на столе);
3) Позиции игроков;
4) Действия игроков;
5) Размеры ставок;
6) Размер банка;
7) Статистика игроков.

Рисунок 3.1 – Окно покерного клиента с выводом статистики

Каждая область отображения информации имеет свои постоянные координаты расположения в окне покерного клиента. Отображение информации представляется в виде одного или множества графических объектов. Чтобы иметь информацию о текущем состоянии игры, необходимо распознать графические элементы из области окна, которые эту информацию в себе несут.

Распознавание графического элемента ведётся на основе вычисления коэффициента корреляции путём сопоставления распознаваемой области с эталонным изображением. Графическое изображение исходной картинки переводится в матричный вид. Элементы матрицы – нормированные значения цвета пиксела в градациях серого.

Формула вычисления коэффициента корреляции имеет следующий вид:

$$\frac{\qquad\qquad\qquad\qquad\qquad\qquad}{\qquad\qquad\qquad\qquad\qquad\qquad} \tag{3}$$

где суммирование ведется по всем парам координат, общим для f и w, – среднее значение элементов маски w (вычисляемое только один раз), а f_{xy} – среднее значение элементов изображения f по области, совпадающей с текущим положением w. Маску w часто называют эталоном, а вычисление корреляции – сопоставление с эталоном. Коэффициент корреляции изменяется в диапазоне от -1 до 1 и не зависит от изменения масштаба значений f и w. Максимальное значение достигается, когда нормализованный (вычитанием среднего значения) эталон и соответствующая область в f (также нормализованная) имеют наибольшее сходство в смысле соотношения.

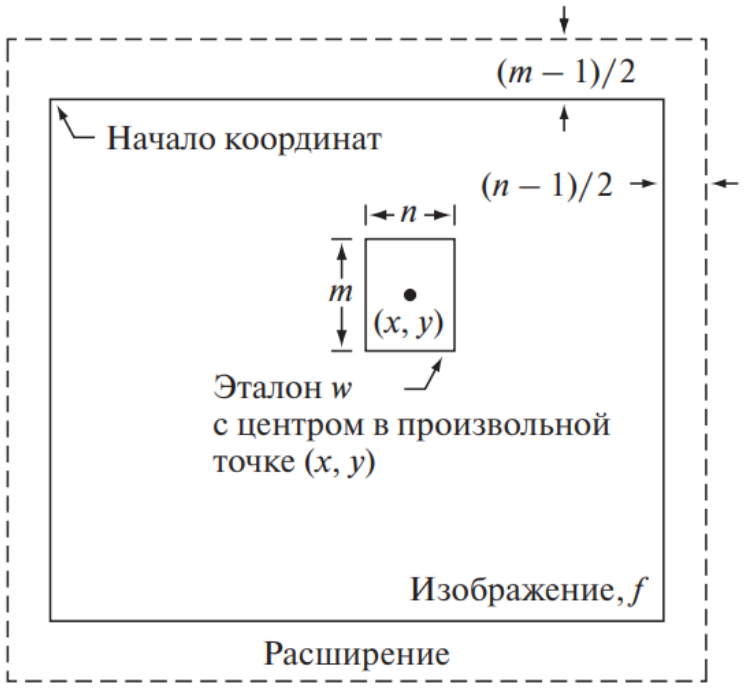

Рисунок 3.2 – Схема корреляционного сопоставления с эталоном

На рисунке 3.2 показан эталон размерами $m \times n$ с центром в произвольной точке (x, y). В этой точке с помощью формулы (3) вычисляется значение коэффициента корреляции, после чего центр эталона перемещается в соседнюю точку и повторяется та же процедура. Полное множество коэффициентов корреляции находится перемещением центром эталона (т.е. перебором значений x и y) так, чтобы центр w пробегал все пиксели f. [3]

Графический элемент можно считать распознанным, если коэффициент корреляции сопоставления эталонного объекта с исходным больше или равен какой либо константе, которая выбирается эмпирически. Если эталонов несколько, то результатом распознавания считается эталон, который показал максимальный коэффициент корреляции (рисунок 3.3).

Рисунок 3.3 – Распознавание карт

3.3. Анализ данных

Вход – модель покерного стола.

Выход – действие: сбросить карты, сделать ставку, принять ставку.

В этом разделе описывается логика принятия решений и ее теоретическое обоснование.

2.3.1 Математика покера

Покерный искусственный интеллект использует математику [11] для принятия решений, а именно математическое ожидание.

Математическое ожидание – мера среднего значения случайной величины в теории вероятностей. Если мат ожидание при игре в орлянку, очевидно равно нулю, то в покере оно зависит от множества факторов. [2] Представим ситуацию:

На ривере, у вас на руках натсовый флеш (см. Приложение 1). Против вас два игрока. Первый делает ставку, ваше слово. Если вы уверены, что у первого сет, а у второго, например, две пары, а также что второй готов уравнять ставку первого, но не ваш рейз, то, очевидно, что для увеличения мат ожидания нужно принять ставку первого, не повышая ее. Если же вы в чем то не уверены: может быть у

первого уже фулл хаус и вы биты, может быть, второй не примет вашу ставку ни при каких обстоятельствах, тогда вычисление мат ожидания сильно затрудняется. Вы определяете, с какой вероятностью у первого игрока на руках фулл хаус, с какой вероятностью второй игрок примет вашу ставку, из этого делаете вывод, при каком вашем действии будет самое большое мат ожидание. Это требует определённых умений по чтению рук и мыслей ваших противников.[5]

Шансы банка – это отношение величины ставки, необходимой, либо для вскрытия карт противника, либо для получения следующей карты (терна или ривера) к размеру банка. [4]

Математическое ожидание и шансы банка.

Пример: У нас на руках Ah2h (Туз черви и двойка черви). Борд: Th 6h Kc 3c (десять черви, шесть черви, король крести, 3 крести). А у противника вы предполагаете AK, т.е. топ пара и топ кикер (см Приложение глоссарий). В банке 100$. Ваш оппонент ставит 60$. Итого в банке 160$.

На данный момент мы биты. Наша рука может усилиться до флеша и тогда мы выиграем. Посчитаем вероятность собрать флеш. В колоде 52 карты. Две у нас на руках плюс четыре на борде, осталось 46 карт не открытых. Нам нужна любая карта червей. Всего червей в колоде 13 минус две на руках и минус две на борде, остается 9. Из 46 карт нам помогают усилиться 9. 9 к 37 = 1 к 4.11 на выигрыш. В банке 160$. Нужно доставить 60$. Шансы банка = 60 к 160 = 1 к 26.6. Получаем, что шансы банка намного меньше, чем шансы на выигрыш, а значит, на дистанции мы будем сильно проигрывать. Правильное решение – сбросить. Если бы противник ставил меньше 20$, было бы выгодно его принимать. [5]

2.3.2 Стратегия коротких стеков

Покерный искусственный интеллект использует стратегию коротких стеков [13]. Основной смысл в том, что все деньги отправляются в банк либо на префлопе, либо на флопе, в самых крайних случаях на терне. Минимизируется количество сложных решений, следовательно, уменьшается количество совершенных ошибок. Используются только самые сильные руки в покере, поэтому вероятность на выигрыш в большинстве случаев будет большей, чем у противника, а значит, на дистанции мы будем в плюсе. Основная часть решений за покерным столом при игре по стратегии коротких стеков будет приниматься на префлопе. Игра на этой улице описывается чартом стартовых рук.

Позиция – ваше положение за столом относительно малого и большого блайнда. Имеет большое значение – в разных позициях вы будете разыгрывать разные по силе руки. [12]

Позиции за столом на 10 человек. Если игроков меньше убираем позиции, начиная с ранней позиции.

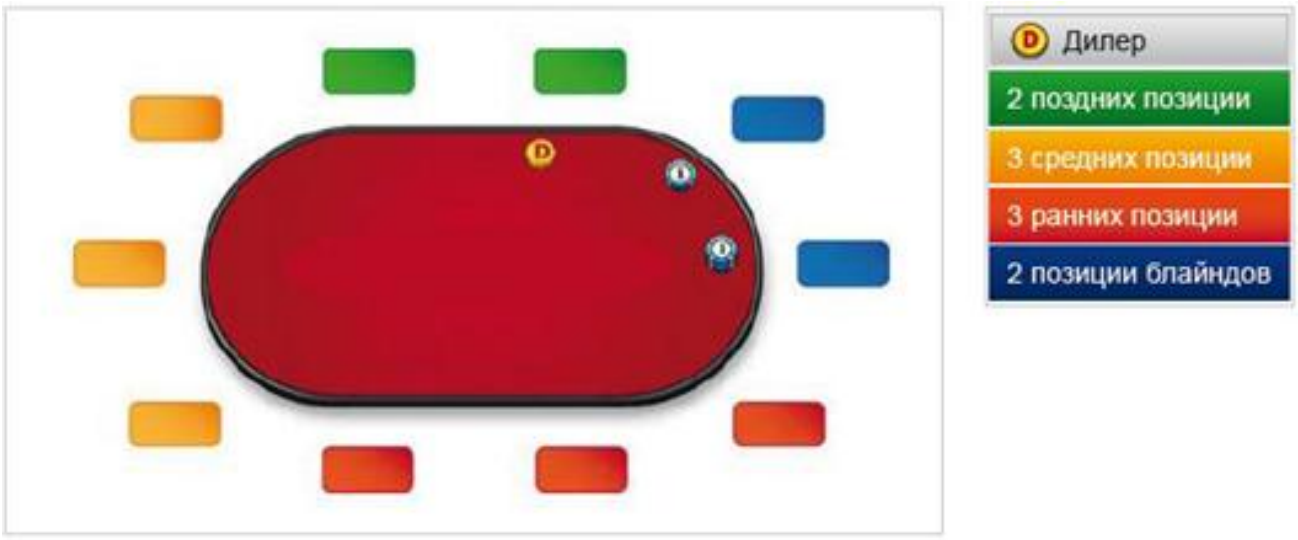

Рисунок 3.4 – Позиции за покерным столом [13]

Таблица 3.1

Чарт стартовых рук [13].

Позиция	Стартовые руки при различных действиях до нас				
	Если не было повышений, делаем рейз	Одно повышение, делаем рейз	2 повышения и больше идем all-in	Одно повышение после нашего рейза идем all-in	Повышение и n*колл отнесем к 2-му случаю
Ранняя	JJ-AA, AK	QQ-AA, AK – all-in AA – 3bet – 25% случаев, 75% – all-in;	KK-AA	JJ-AA, AK	JJ-KK, AK – all-in AA – 3bet + n*bet – 25% случаев, 75% – all in
Средняя	99-AA, AK, AQ	JJ-QQ, AK – all-in KK – 3bet – 35% случаев, 65% – all-in; AA – 3bet – 45% случаев, 55% – all-in;	KK-AA	JJ-AA, AK	JJ-QQ, AK – all-in; KK – 3bet + n*bet – 35% случаев, 65% – all-in; AA – 3bet + n*bet – 45% случаев, 55% – all-in;

Продолжение таблицы 3.1

Позиция	Стартовые руки при различных действиях до нас				
	Если не было повышений, делаем рейз	Одно повышение, делаем рейз	2 повышения и больше идем all-in	Одно повышение после нашего рейза идем all-in	Повышение и n*колл отнесем к 2-му случаю
Поздняя + блайнды	77-AA, AT-AK, KQ, KJ;	AK – all in; JJ – 3bet – 10% случаев, 90% - all-in; QQ – 3bet – 15% случаев, 85% - all-in; KK – 3bet – 35% случаев, 65% - all-in; AA – 3bet – 45% случаев, 55% - all-in;	KK-AA	JJ-AA, AK	AK – all in; JJ – 3bet – 10% случаев, 90% – all in; QQ – 3bet – 15% случаев, 85% – all in; KK – 3bet – 35% случаев, 65% – all in; AA – 3bet – 45% случаев, 55% – all in;
Когда все игроки до баттона или катофа (позиция баттон - 1) сбросили и игрок повышает – стил-рейз. Стил рейз с 88-AA, AJ-AK. В ответ на рестил оппонента идем all-in с TT – AA, AQ, AK					
Размер рейза – случайно - 3bb, 3.5bb, 4 bb. + bb на каждого лимпера (игрок вошедший в игру без повышения).					

Постфлоп:

На рисунке 3.5 представлены комбинации, которые могут получиться на флопе.

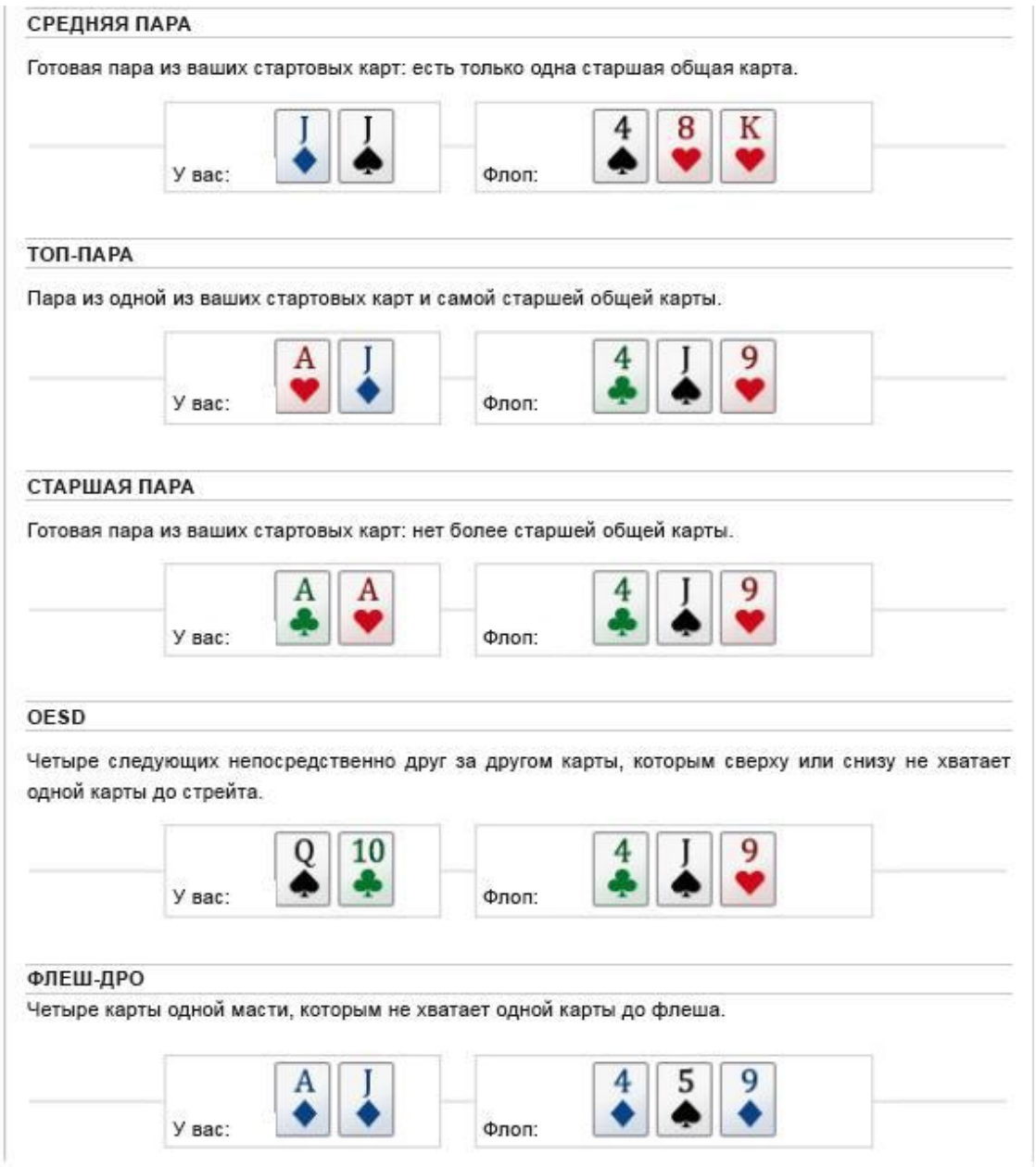

Рисунок 3.5 – Возможные комбинации [13]

Со средней парой (карманной), топ парой, старшей парой + флеш-дро и OESD и любой картой сильнее (стрит, флеш, сет, фуллхаус, каре, стрит-флеш, роял-флеш). Делаем следующее:

- Если никто еще не делал ставки, то ставим 2/3 банка.
- Если кто-то поставил, сразу идем all-in;
- Если сделали ставку, ее повысили сразу all-in;
- Если просто колировали ставку, на терне all-in;

Если ни во что не попали:

- Если банк к началу круга торговли вдвое превышает стек, то идем all-in;
- Если только один оппонент, то делаем продолженную ставку 2/3 стека
- Сбрасываем или чек, если противников > 1.

3.2.3 Классификация оппонентов

Классификация оппонентов на основе статистических данных. Это очень важная информация, определяющая тип соперника, то, как он реагирует на те или иные действия в разных ситуациях.

Основные параметры статистики [13]

1. Количество вложенных денег в банк (VP$IP)
2. Процент префлоп рейзов (PFR)
3. Фактор агрессии (AF)
4. Частота вскрытия карт (WSD)
5. Процент рук сброшенных на стилинг (FSBBBTS)
6. Количество попыток украсть блайнды (ATSB)

Первые две характеризуют игру на префлопе, третья на всех улицах.

VPIP - voluntary put $ in the pot (%) - добровольно поставленные в банк деньги – процент рук, когда игрок делал ставку на префлопе. Здесь все call & raise, но нет check на big blind и fold на small blind. Это важный показатель, делящий игроков на лузовых и тайтовых. Обычно у хороших игроков VPIP не превышает 25%. По этому показателю сразу видно с чем человек играет. Чем меньше этот показатель, тем с лучшими картами игрок входит в игру.

PFR - preflop raise (%) – префлоп рейз - процент рук, когда игрок делал рейз или ре-рейз до флопа. 2-й по важности параметр, делящий игроков на пассивных и агрессивных. Адекватное значение в районе 7-13%. Рейз от игрока с низким PFR значит что у него что-то сильное, типа старшей пары или AK. Высокий PFR в сочетании с высоким VPIP указывает на сверх агрессивного игрока.

AF - aggression factor - фактор агрессии - значение агрессивности игрока, определяется как (% bet + % raise) / % call. Фактор агрессии можно считать для каждого раунда игры, но обычно используют усредненный. Это, третий по важности параметр, позволяющий оценить агрессивность игрока, начиная с флопа. Средний фактор агрессии на флопе и терне 2,8 (диапазон 2-3,5), на ривере чуть ниже - 2,1 (1,6-2,5). Хороший игрок достаточно агрессивен, медленная игра на полном столе особо не рекомендуется, нормальные карты надо защищать, плохие - сбрасывать.

WSD - went to showdown (%) - дошел до вскрытия - процент сдач, в которых игрок смотрел флоп и дошел до вскрытия. Еще одна важная характеристика, позволяет оценить лузовость/тайтовость противника после префлопа. Средняя величина - 32%, диапазон 27-38%. По величине WSD лучше всего можно определить "сопротивляемость" блефу. Игрок с высоким WSD (так называемый "колинг стейшн") будет отвечать со своей младшей парой или даже голым тузом до упора, блеф становится бессмысленным. Игрок с низким значением боится проиграть, на опасном флопе они могут сбросить любую карту, кроме натса, против них блеф наиболее эффективен.

FSBBBTS - Fold SB/BB to steal (%) – процент рук, сброшенных на малом/большом блайнде против попытки "украсть блайнды" (такой попыткой считается любой рейз с двух последний позиций, если до этого в игру не вошел

ни один игрок). Средняя величина для малого блайнда - 83%, для большого - 63%, диапазоны, соответственно 68%-91% и 21-79%. Желательно знать данную величину для противников, у которых вы собираетесь воровать блайнды, чем выше - тем смелее нужно рейзить.

ATSB - Attempts to steal blinds (%) - процент попыток украсть блайнды к предоставленным возможностям. Предыдущие два параметра показывали на защиту блайндов, этот - на их атаку. Среднее значение 28%, в диапазоне от 19% до 33%.

Мы перечислили основные статистики. Они будут учитываться при определении правильного действия.

Введём понятие диапазона рук – это совокупность рук, которые может разыгрывать игрок. Например, все будут играть с двумя тузами, но 23 все будут скидывать.

Применение статистики для классификации оппонентов и определения их диапазона рук.

Игроки в покер могут играть по одной из многих возможных стратегий. Они могут играть тайтово на префлопе, но очень агрессивно на флопе, лузово на префлопе и агрессивно на флопе, тайтово на префлопе, пассивно на флопе. Классифицируем оппонентов, основываясь на основных статистиках.

1. Камень (VPIP = 9, AF = 7) - очень тайтовые и агрессивные. Такой игрок играет по очень тайтовой стратегии, ждет карманную пару на префлопе, возможно AK. После флопа пытается извлечь как можно больше денег из сета или оверпары, иначе делает фолд, не собрав ни то ни другое. Такие игроки открывают торговлю только рейзом. Диапазон рук таких противников описан в приложении Б. У них можно часто красть банки и блайнды.

2. Ниты (VPIP = 13, AF = 9) – Разыгрывают несколько больший диапазон рук на префлопе (см приложение Б). У них также как и у камней можно воровать блайнды и банки, но с большей осторожностью.

3. Нитовый, Нормальный, Лузовый ТАГ (ТАГ – тайтово агрессивный игрок VPIP = 17-24, AF = 13-20). Это хорошие игроки, игра которых может быть непредсказуемой, но 20 они все равно совершают большое количество ошибок. Диапазоны их рук смотрите в приложении Б.

4. Нормальный ЛАГ (VPIP = 29, AF = 24) – самые сложные противники. Их основное различие от ТАГов – это более частая кража блайндов и банков. Диапазоны смотрите в приложении Б.

5. Турист (VPIP = 22, AF = 7). Такие игроки имеют значения основных статистик, которые могут значительно колебаться, делают множество ошибок, могут идти до конца со средней парой, не понимают силу позиции, проводят полу блефы ни на чем не основываясь. Их диапазон стартовых рук может быть очень широк, против них нужно разыгрывать только сильные руки и на дистанции мы будем в плюсе. [22]

Остальные игроки играют в покер для развлечения и приносят большие прибыли профессионалам. На основе этой классификации стратегия коротких стеков может быть доработана для извлечения большей прибыли.

Пример:

Если предоставлена возможность для воровства блайднов (Мы в позиции баттона и все до нас сбросили), то мы смотрим, какие игроки на блайндах, если они ниты, то мы можем смело воровать блайнды практически с любой рукой. Если они ЛАГи, то нужно посмотреть их FSBBBTS статистику и сделать вывод. Диапазон рук, с которыми мы будем воровать блайнды резко сокращается.

3.2.4 Стратегия коротких стеков с учетом классификации оппонентов

Для извлечения большей прибыли из игры мы должны корректировать свою игру относительно игры соперников. Окончательная стратегия коротких стеков против различных типов оппонентов представлена в виде чарта стартовых рук.

Таблица 3.2

Модифицированная стратегия коротких стеков [22].

Позиция	Стартовые руки при различных действиях до нас				
	Если не было повышений, делаем рейз	Одно повышение, делаем рейз	2 повышения и больше идем all-in	Одно повышение после нашего рейза идем all-in	Повышение и n*колл отнесем к 2-му случаю
Ранняя	JJ-AA, AK.	Против нитов и камней: QQ-AA, AK – all-in; Против всех остальных: JJ-KK, AK – all-in; AA – 3bet – 25% случаев, 75% – all-in	KK-AA.	Против нитов и камней: QQ-AA, AK – all-in; Против всех остальных: JJ-AA, AK.	JJ-KK, AK – all-in; AA – 3bet + n*bet – 25% случаев, 75% – all in;
Средняя	99-AA, AK, AQ.	Против нитов и камней: QQ-AA, AK – all-in; Против всех остальных: JJ-QQ, AK – all-in;	KK-AA.	Против нитов и камней: QQ-AA, AK – all-in; Против всех остальных: JJ-AA, AK.	JJ-QQ, AK – all-in; KK – 3bet + n*bet – 35% случаев, 65% – all-in;

Продолжение таблицы 3.2

Позиция	Стартовые руки при различных действиях до нас				
	Если не было повышений, делаем рейз	Одно повышение, делаем рейз	2 повышения и больше идем all-in	Одно повышение после нашего рейза идем all-in	Повышение и n*колл отнесем к 2-му случаю
Средняя	99-AA, AK, AQ	KK – 3bet – 35% случаев, 65% – all-in; AA – 3bet – 45% случаев, 55% – all-in;		Против нитов и камней: QQ-AA, AK – all-in Против всех остальных: JJ-AA, AK	AA – 3bet + n*bet – 45% случаев, 55% – all-in;
Поздняя + блайнды	Против нитов и камней: 55-AA, AT-AK, KQ, KJ, QJ; Против ТАГов: 99-AA, AJ-AK, KQ; Против ЛАГов: 99-AA, AQ, AK; Против всех остальных: TT-AA, AK, AQ;	Против нитов и камней: QQ-AA, AK – all in; Против всех остальных: AK – all in; JJ – 3bet – 10% случаев, 90% - all-in; QQ – 3bet – 15% случаев, 85% - all-in; KK – 3bet – 35% случаев, 65% - all-in; AA – 3bet – 45% случаев, 55% - all-in;	KK-AA	Против нитов и камней: QQ-AA, AK – all in Против всех остальных: JJ-AA, AK	AK – all in; JJ – 3bet + n*bet – 10% случаев, 90% – all in; QQ – 3bet + n*bet – 15% случаев, 85% – all in; KK – 3bet + n*bet – 35% случаев, 65% – all in; AA – 3bet + n*bet – 45% случаев, 55% – all in;
Стил рейз (то же что и в поздней позиции). Рестил – ситуация, когда игрок повышает в ответ на наш стил рейз.					

Окончание таблицы 3.2

Идем all-in: Против нитов и камней: KK,AA Против ТАГов: JJ-AA,AK Против ЛАГов: TT-AA,AK,AQ,AJ Против всех остальных: JJ-AA,AK
Размер рейза – случайно - 3bb, 3.5bb, 4 bb. + bb на каждого лимпера.

Постфлоп игра остается такой же, за исключением того, что против нитов и камней скидываем на их переповышение:

Со средней парой (карманной), топ парой, старшей парой + флеш-дро и OESD. Делаем следующее:

- Если никто еще не делал ставки, ставим 2/3 банка.
- Если кто то поставил сразу идем алл-ин.
- Если сделали ставку, ее повысили сразу алл-ин
- Если просто колировали ставку, на терне алл-ин.

Если ни во что не попали:

- Если банк к началу круга торговли вдвое превышает стек то идем алл-ин.
- Если только один оппонент, то делаем продолженную ставку 2/3 стека
- Сбрасываем или чек, если противников > 1.

3.2.5 Определение комбинации и шансов улучшить комбинацию

Для определения комбинации мы будем сопоставлять ей число. Первая цифра этого числа будет обозначать номер комбинации – чем больше цифра, тем сильнее комбинация. В случае равенства типа комбинаций, сравниваются карты, из которых эта комбинация получена. В каждой комбинации есть «важность» карты (например в фулхаусе 3-йка важнее чем пара, а в каре четверка важнее кикера). Карты сравниваются в порядке важности: если самые «важные» одинаковы, сравнивают, менее важные. Если очередные сравниваемые карты различны по достоинству, то комбинация со старшей картой считается сильнее. Если различных по достоинству карт нет, то комбинации считаются 23 равными. Если перевести достоинство карты в числа от 1 до 13, то соответственно 2 и 3 цифры числа комбинации будут соответствовать достоинству самой важной карты, 4 и 5 второй по важности и т.д. Таким образом, мы получим, что чем больше число, тем сильнее комбинация. Например, следующие комбинации будут иметь номера:

Таблица 3.3

Примеры комбинаций и соответствующие им числа.

Комбинация	Номер
Стрит Флэш	91000000000 (для стритов важна только 1 карта)
Карэ	80907000000 (сначала карта каре, затем кикер)
Фулл-Хаус	71007000000 (сначала тройка, затем пара)
Флэш	61009060402 (все карты в порядке убывания)

Окончание таблицы 3.3

Комбинация	Номер
Стрит	50600000000 (стрит – только 1 важная карта)
Три карты	40914070000 (сначала тройка, затем страший кикер и младший кикер)
Две пары	31002080000 (старшая пара, младшая пара и кикер)

Однако такое представление неэффективно использует пространство чисел: из 2-х разрядов (100 значений) использует только 13 значений. Результат можно улучшить, используя 16-ричную систему: на 1 карту использовать 1-н 16-ричный разряд, комбинации (уменьшить на 1) и карты (уменьшить на 2) при этом нумеровать с 0. Тогда значения из примера будут иметь вид: 0x880000, 0x775000, 0x685000, 0x587420, 0x440000, 0x37C500, 0x280600, 0x137640, 0x086420. Эти числа уже помещаются в стандартное четырехбайтовое число и их можно спокойно использовать. Если же этого будет не достаточно, то можно будет просто пронумеровать все комбинации, а определять их номера с помощью хеш-таблиц или деревьев и отображений (map), т.е. произвести сжатие.

Определить из 7-ми карт 5-тикаточную комбинацию можно так же, как мы это определяем в жизни глазами:

Прежде всего, определим максимальное количество карт одной масти, если их 5 или больше то мы имеем минимум flush, и тогда следует запомнить первых 5 старших карт этой масти.

Теперь посчитаем, каких номиналов сколько карт у нас есть, и разобьем на соответствующие группы по количеству: 4-ре одинаковых в 1-ю группу, тройки одинаковых во 2-ю, пары в 3-ю, одиночные в последнюю.

Отсортируем их в порядке убывания в группах. Помимо этого получим последовательность различных карт упорядоченных по убыванию.

Если мы имеем в этой последовательности 5 или более карт и разница между 1-й и 5-й равна 4, то мы имеем straight, старшинство определяем 1-й картой.

Тоже самое если в ней 6 карт и разница между 2-ой и 6-ой равна 4, старшинство по 2-й, а также если их там 7 и разница между 3-й и 7-ой (по 3-й).

Если первая карта в последней группе “Ace”, а 4-я с конца “5”, то мы имеем straight с кикером “Ace”.

Определение шансов на улучшение комбинации

Если закрытых карт больше нет, то определить комбинацию не составляет труда. Если закрыты 1 или 2 карты, то мы предполагаем, что вероятность выпадения каждой карты, кроме открытых одинакова, и для каждой из них (или для каждой пары) определяем комбинацию. Несмотря на количество различных вариантов карт, комбинаций будет ограниченное количество, и мы можем разбить карты на группы по комбинациям, которые они дадут. Таким образом, мы можем оценить вероятность выпадения соответствующей комбинации. В случае когда у нас известны только 2-е карты на руках ситуация немного другая. Результирующих комбинаций может получиться очень много, поэтому чаще оценивают среднюю вероятность выигрыша, которая заранее просчитана. [1]

4. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

Программа реализована на языке C# в среде разработки Visual Studio 2015.

4.1. Алгоритм программы

Алгоритм работы программы показан на рисунке 4.1. Программа выполняет цикл, пока не будет нажата кнопка «Стоп». Тело цикла позволяет непрерывно получать информацию о состоянии игры (рисунок 4.2) с окна покерного клиента, анализировать данные (рисунок 4.3) и выполнять действия.

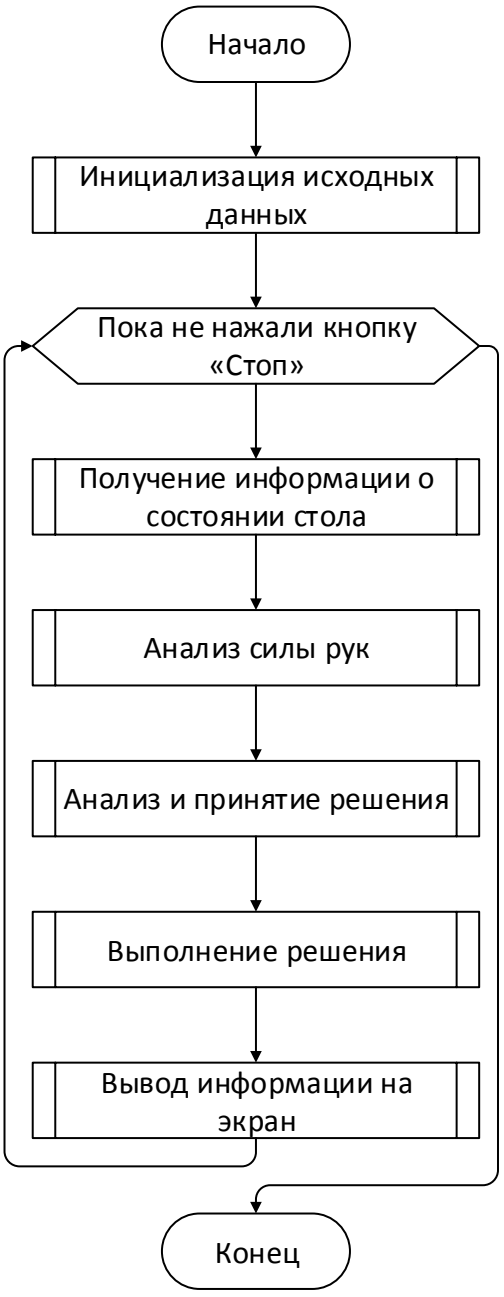

Рисунок 4.1 – Алгоритм работы программы

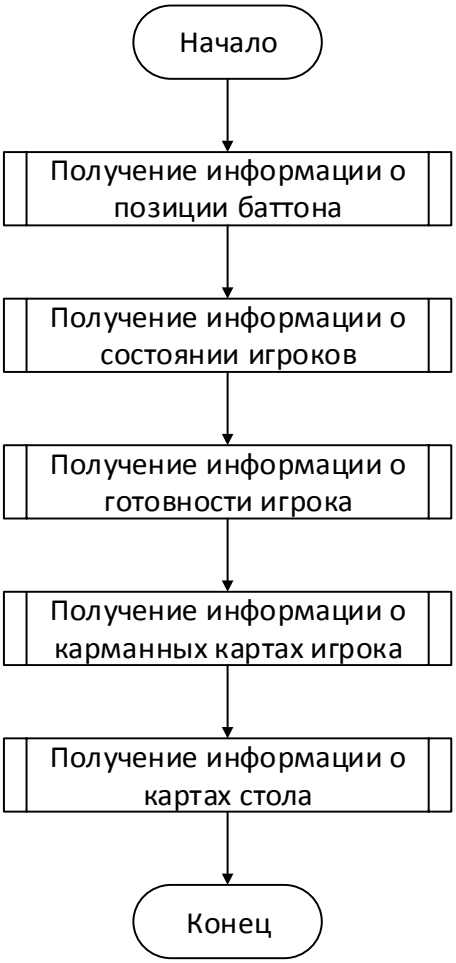

Рисунок 4.2 – Алгоритм блока получения информации о состоянии стола

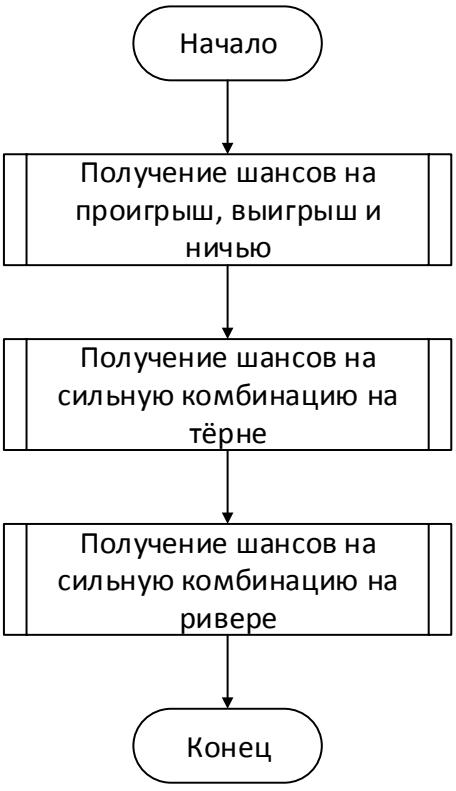

Рисунок 4.3 – Алгоритм блока анализа

4.2. Руководство пользователя

Чтобы запустить покерный искусственный интеллект откройте файл Pr_poker.exe, после чего запустится главное окно программы (рисунок 4.2).

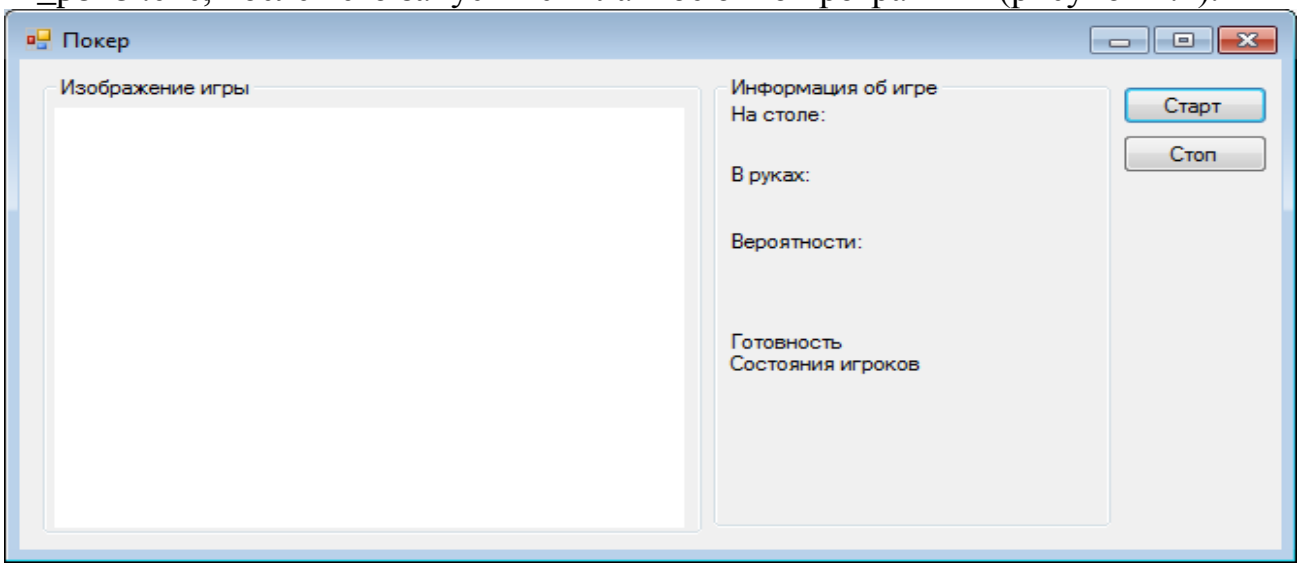

Рисунок 4.4 – Главное окно программы

Далее требуется запустить покерный клиент «PokerStars», выбрать стол для игры в покер на 6 человек, уменьшить окно покерного клиента до минимального размера и установить его в левый верхний угол экрана. Для вывода статистики об игроках требуется запустить дополнительно программу «Holdem Manager 2».

Требуется нажать кнопку «Старт» для запуска работы искусственного интеллекта. Кнопка «Стоп» останавливает работу искусственного интеллекта. В области «Информация об игре» выводится информация о состоянии игры и результаты анализа данных. Программа выполняет действия, полученные на основе результатов анализа данных и принятия решения (рисунок 4.5).

Рисунок 4.4 – Пример работы программы

4.3. Результаты работы программы

Проверка работы программы и получение результатов производилась для следующих стратегий:

- Стратегии коротких стеков (рисунок 4.3).
- Стратегии коротких стеков с учетом классификации оппонентов (рисунок 4.4).

Получение результатов в ходе всей проверки производилось в играх со следующими свойствами:

- Количество игроков: 6;
- Блайнды: $0.1/$0.2;
- Величина стартового стека: $1 или 50bb.

Условие остановки работы программы является просадка в 250bb или $5.

Для стратегии коротких стеков (см. рисунок 4.3) было сыграно 3667 рук, максимальный выигрыш составлял 61bb или $1.22. Максимальный проигрыш составлял -228bb или -$4.56. Итоговый выигрыш -198bb или -$3.96. Средний выигрыш на 100 рук составляет -5.45bb/100 или $0.11/100.

Рисунок 4.5 – График прибыли стратегии коротких стеков

Для стратегии коротких стеков с учетом классификации оппонентов (см. рисунок 4.4) было сыграно 34081 руки, максимальный выигрыш составлял 509bb или $10.18. Максимальный проигрыш составлял -11bb или -$0.22. Итоговый выигрыш 506bb или $10.12. Средний выигрыш на 100 рук составляет 1.48bb/100 или $0.03/100.

Рисунок 4.6 – График прибыли стратегии коротких стеков с учетом классификации оппонентов

ЗАКЛЮЧЕНИЕ

В общем виде задачу искусственного интеллекта игры в покер в интернете можно разбить на следующие проблемы:

1. Извлечение данных из покерного клиента для получения модели покерного стола;
2. Анализ модели и выбор соответствующего действия;
3. Выполнение действия.

В ходе проектирования были изучены различные подходы к извлечению данных из программ сторонних производителей. Один из подходов детально изучен и реализован.

В ходе изучения второй проблемы, были изучена теория покера и его математический аппарат. Проведен анализ базовых стратегий игры. В ходе анализа было установлено, что стратегия коротких стеков легка для алгоритмизации, при этом не является ущербной и даже имеет преимущества перед остальными стратегиями. Базовая стратегия коротких стеков была доработана с учетом реальных динамических условий игры и реализована в покерном искусственном интеллекте.

Реализовано моделирование поведения противника на основе статистик, которые считаются по истории сыгранных рук. Реализовано автоматическое нажатие кнопок, соответствующих принятому решению.

Приведены результаты сравнения предложенных алгоритмов для искусственного интеллекта. По сравнению с существующими алгоритмами было достигнуто увеличение среднего выигрыша. Получены экспериментальные оценки среднего выигрыша в игре покер в сети интернет.

Таким образом, реализована программа – система искусственного интеллекта, автоматизирующая игру в покер в интернете.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Автоматизированная система принятия решений для участника многоагентных взаимодействий в условиях неполноты информации [Электронный ресурс]. – Режим доступа : http://rud.exdat.com/docs/index-833756.html#4210231, свободный. – Загл. с экрана.

2 Вадим, Ю. М. Секреты покера. Учимся выигрывать с Вадимом Маркушевским / Ю. М. Вадим. – : Питер, 2011. – 170 с.

3 Гонсалес, Р. Цифровая обработка изображений / Р. Гонсалес, Р. Вудс. – М. : Техносфера, 2012. – 1104 с.

4 Дэн, Х. Харрингтон о Холдеме. Том 1 / Х. Дэн. – : Сафари, 2009. – 234 с.

5 Дэн, Х. Харрингтон о Холдеме. Том 2 / Х. Дэн, Б. Роберти. – : Сафари, 2009. – 399 с.

6 ИИ для покера: как научить алгоритмы блефовать / Блог компании Сбербанк / Хабр [Электронный ресурс]. – Режим доступа : https://habr.com/company/sberbank/blog/337264/, свободный. – Загл. с экрана.

7 Покер — Википедия [Электронный ресурс]. – Режим доступа : https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BA%D0%B5%D1%80, свободный. – Загл. с экрана.

8 Покер - это... Что такое Покер? [Электронный ресурс]. – Режим доступа : https://dic.academic.ru/dic.nsf/ruwiki/47427, свободный. – Загл. с экрана.

9 Правила техасского холдема [Электронный ресурс]. – Режим доступа : https://www.pokerstars-kk.com/ru/poker/games/texas-holdem/?no_redirect=1, свободный. – Загл. с экрана.

10 Равновесие Нэша — Википедия [Электронный ресурс]. – Режим доступа : https://ru.wikipedia.org/wiki/Равновесие_Нэша, свободный. – Загл. с экрана.

11 Сайт Дмитрия Лесного [Электронный ресурс]. – Режим доступа : http://www.dmitrylesnoy.com/?mod=public&tag=18, свободный. – Загл. с экрана.

12 Словарь покера | Покер блог [Электронный ресурс]. – Режим доступа : http://pokerazart.ru/slovar-pokera, свободный. – Загл. с экрана.

13 Словарь покерных терминов [Электронный ресурс]. – Режим доступа : http://www.pokerpalace.ru/glossary.htm/, свободный. – Загл. с экрана.

14 Стратегия коротких стеков [Электронный ресурс] / PokerStrategy.com – электронные текстовые и графические данные – Режим доступа: http://resources.pokerstrategy.com/Strategy/ru/pdf/ps_nl_basic_handout_V4_ru.pdf. – Загл. с экрана.

15 Danny, Bragonier. Statistical Analysis of Texas Holdem Poker / Bragonier. Danny. // California State Polytechnic University. – 2010. – 1. – С. 1-58.

16 David Sklansky “The Theory Of Poker” - Two Plus Two Pub 1994 – 276 p.

17 Ed Miller, Sunny Mehta, Matt Flynn “Small Stakes No Limit Hold’em” – Two Plus Two Pub 2009 – 235 p.

18 Guy Van den Broeck. Monte-Carlo Tree Search in Poker using Expected Reward Distributions / VandenBroeck. Guy, Driessens. Kurt, Ramon. Jan. // Lecture Notes in Computer Science. – 2009. – 528. – С. 3-4.

19 How I Built a Working Poker Bot [Электронный ресурс] / CodingTheWheel.com – электронные текстовые и графические данные – Режим доступа: http://www.codingthewheel.com/archives/how-i-built-a-working-poker-bot - свободный.

20 Johannes, Heinrich. Self-Play Monte-Carlo Tree Search in Computer Poker / Heinrich. Johannes, Silver. David. // Lecture Notes in Computer Science. – 2014. – 1141. – С. 162.

21 Jonathan, Rubin. On the Construction, Maintenance and Analysis of Case-Based Strategies in Computer Poker / Rubin. Jonathan. – Auckland : The University of Auckland, 2013. – 183 с.

22 Matej, Moravčík. DeepStack: Expert-Level Artificial Intelligence in Heads-Up No-Limit Poker / Moravčík. . Matej. и др. // Artificial Intelligence. – 2017. – 3. – С. 3-6.

23 PokerTracker Statistical Reference Guide [Электронный ресурс] / PokerTracker.com – электронные текстовые данные – Режим доступа: http://www.pokertracker.com/products/PT3/docs/PokerTracker3_Statistical_Reference_Guide.pdf - свободный

ПРИЛОЖЕНИЕ 1. Список терминов

В тексте отчета используются покерные термины, ниже приведены часто использующиеся.

Action – действие – активные ставки (бет-рейз-перерейз) в игре.

All-in – ва-банк, алл-ин – ставка всех имеющихся денег в банк.

Bet – ставка, делать ставку, бет – действие по добавлению фишек в банк при игре, а также собственно ставка.

Big blind – большой блайнд,– в холдеме два первых игрока перед сдачей делают обязательные ставки «вслепую». Второй игрок, а также сделанная им ставка в размере двух малых ставок и называются большим блайндом.

Blinds – блайнды – два игрока слева от дилера, а также сделанные ими до раздачи обязательные ставки.

Bluff – блеф – ставка, когда ваша рука наверняка слабее руки оппонента.

Board – стол, букв. доска – общие карты, выкладываемые на стол.

Button – кнопка, баттон – небольшая кнопка, указывающая, какой игрок считается дилером. Также название этого игрока, говорят, что он сидит на кнопке/баттоне.

Call – ответ, отвечать, уравнивать, кол – если игрок перед вами сделал ставку, то для продолжения игры вы должны уравнять его ставку.

Cash game – не турнирная игра, букв. игра на деньги – обычная игра на один стол, когда игрок в любой момент может прийти или уйти.

Check – пас, чек – если до игрока не было сделано ставок, то он может продолжить игру, не делая ставку.

Dealer – дилер – сдающий карты. В домашней игре дилером обычно является каждый игрок по очереди, в казино есть профессиональный дилер, а в онлайн казино дилером является компьютер, тем не менее, игроки по кругу считаются дилерами.

Draw – тянуть, дро – комбинация, на данный момент слабая, но имеющая шансы на улучшение. Чаще всего говорят о флэш-дро (у игрока четыре карты к флэшу) и стрит-дро (четыре карты к стриту).

Early position – ранняя позиция – три игрока слева от большого блайнда.

Flop – флоп – три карты, выкладываемые в открытую на столе, а также следующий за этим раунд торговли.

Flush – флеш – одна из сильных покерных комбинаций, пять карт одной масти.

Flush draw – флеш-дро, букв. тянуть флеш – четыре карты одной масти, дающие шанс собрать флэш.

Fold – сброситься, фолд – сбросить карты в ответ на ставку противника.

Heads up – один на один, букв. с поднятыми головами – игра двух игроков, либо когда остальные сбросились, либо в игре один на один.

Hole cards – карманные карты – в холдеме две карты, раздаваемые игроку в закрытую.

In position – в позиции – быть в позиции, значит находится за противником, действовать вслед за ним.

Kicker – кикер, букв. ударяющий – при сравнении одинаковых пар, двух пар, сетов и каре выигрывает игрок у которого старше не парная карта, эта карта и называется кикером.

Late position – поздняя позиция – баттон и игрок перед ним.

Limp – лимп, букв. хромать – входить в игру на префлопе колом, без рейза.

Loose – лузовый, букв. широкий, свободный – игрок, играющий слишком много рук, а также игра, где играет много лузовых игроков.

Loose aggressive – лузово-агрессивный – характеристика игрока, который играет много рук, причем часто ставит и поднимает ставки.

Loose passive – лузово-пассивный - характеристика игрока, который играет много рук, причем обычно идет до ривера, отвечая на ставки противников.

Middle position – средняя позиция – три игрока между ранней и поздней позицией (шестой-восьмой игроки считая от дилера).

No limit – безлимит – величина ставки, которую игрок может сделать в любом из раундов торговли, ничем не ограничена сверху, игрок в любой момент может пойти ва-банк.

Offsuit – разномастные – две карты разных мастей.

Open – открывать – делать первую ставку в данном раунде торговли.

Overpair – оверпара, букв. пара старше – карманная пара, старше наибольшей карты на столе.

Pair – пара – низкая покерная комбинация, две карты одного ранга.

Pocket pair – карманная пара – пара, полученная игроком на префлопе.

Position – позиция – положение игрока за столом относительно дилера. От дилера по часовой стрелки различают малый и большой блайнды, ранние, средние и поздние позиции.

Pot – банк – куча фишек, составлявших ставки игроков, находится в середине стола.

Quads – каре

Raise – поднимать, рейз – увеличивать ставку, уже сделанную другим игроком.

Reraise – перейз, букв. повторный рейз – рейз после рейза оппонента, также называется 3- бет.

River – ривер, букв. река – пятая карта на столе, а также следующей за ней последний раунд торговли.

Rock – камень, булыжник – обозначение очень тайтового игрока.

Royal flush – роял флэш – наивысшая комбинация в покере, пять карт от десятки до туза одной масти.

Set – тройка, сет – три карты одного ранга, составленные из карманной пары и карты на столе.

Short stacked – малый стек – игра с очень маленьким стеком, которого не хватит чтобы делать ставки до ривера, игрок вынужден будет раньше поставить ва-банк.

Showdown – вскрытие, показ карт – по окончании торговли на ривере, уравнявшие карты игроки показывают карты и определяют победителя. В онлайн покере обязательно показывается только победившая рука, проигравшие руки можно не показывать.

Small blind – малый блайнд, букв. малый слепой – в холдеме два первых игрока перед сдачей делают обязательные ставки «вслепую». Первый игрок, а также сделанная им ставка и называются малым блайндом.

Stack – стек – 1. все фишки, имеющиеся у игрока для игры, 2. стопка фишек.

Steal the blinds – воровать, украсть блайнды – рейз из последней позиции, когда не было ставок до этого, чтобы заставить сбросится блайнды и взять банк еще на префлопе (или повторить ставку и взять банк на флопе).

Straight – стрит – пять последовательных карт.

Straight draw – стрит-дро, букв. тянуть стрит – четыре карты к стриту, дающие шанс собрать стрит.

Straight flush – стрит-флеш – вторая по силе покерная комбинация, пять последовательных карт одной масти.

Street – улица - раунд торговли, торн и ривер иногда называют третьей и четвертой улицей.

Texas hold'em – техасский холдем – самая популярная покерная игра.

Three bet – три ставки, 3-бет – то же что и перейз.

Tight – тайтовый, букв. тесный – игрок, играющий только самые лучшие карты на префлопе, очень небольшой процент сдач.

Tight aggressive – тайтово-агрессивный – игрок, играющий лучшие карты, причем агрессивно – ставит и рейзит.

Tight passive – тайтово-пассивный – игрок, играющий лучшие карты, но пассивно – редко поднимает, часто отвечает.

Top kicker – старший кикер – самый высокий кикер при вскрытии на равной комбинации.

Trips – тройка, трипс – три карты одного ранга, причем пара на столе соединяется с одной карманной картой игрока.

Turn – торн, букв. поворот – четвертая карта на столе, а также следующей за ней предпоследний раунд торговли.

Two pair – две пары – низкая покерная комбинация, две карты одного ранга, плюс две карты другого ранга.

Under pair – нижняя пара – пара ниже младшей карты стола. [11,12]

ПРИЛОЖЕНИЕ 2. Диапазоны стартовых рук

На всех рисунках сиреневым показаны те руки, которые разыгрывает оппонент. Красным – одномастные руки, голубым – разномастные, зеленым – карманные пары.

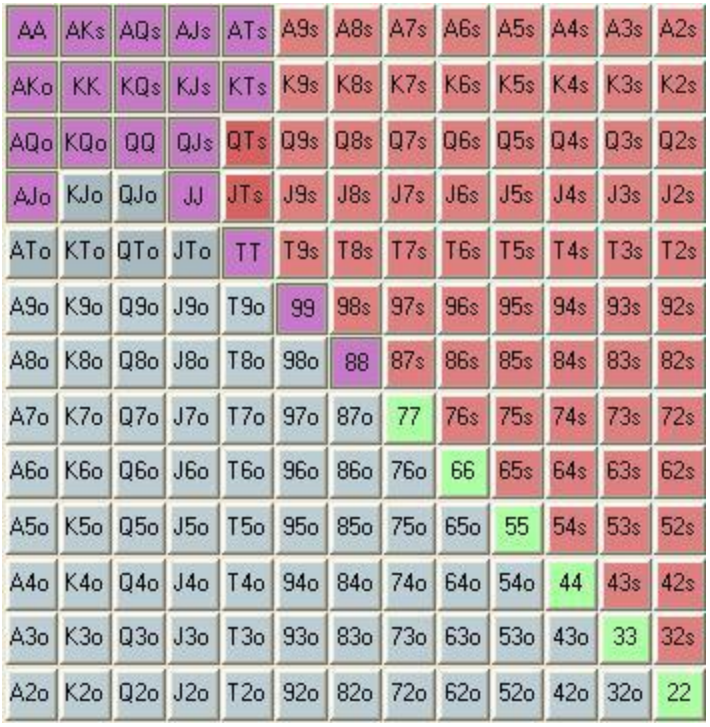

Рисунок 2.1 - Камень (VPIP = 9, AF = 7)

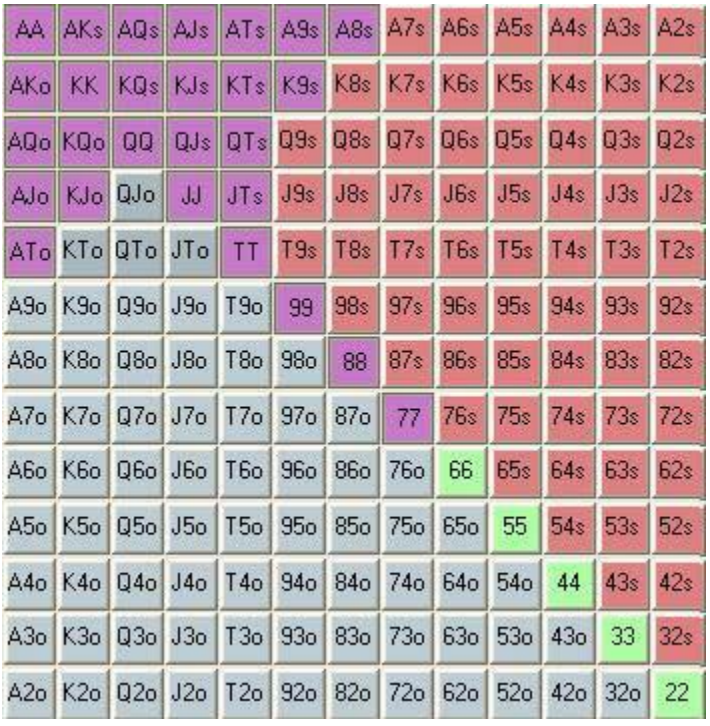

Рисунок 2.2 - Ниты 13 - (VPIP = 13, AF = 9)

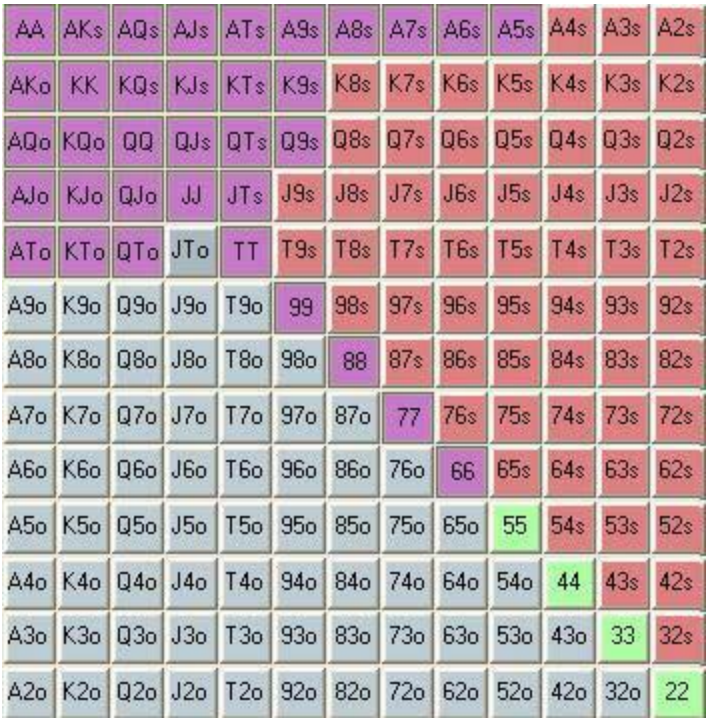

Рисунок 2.3 - Нормальный ТАГ(VPIP = 17 AF = 13)

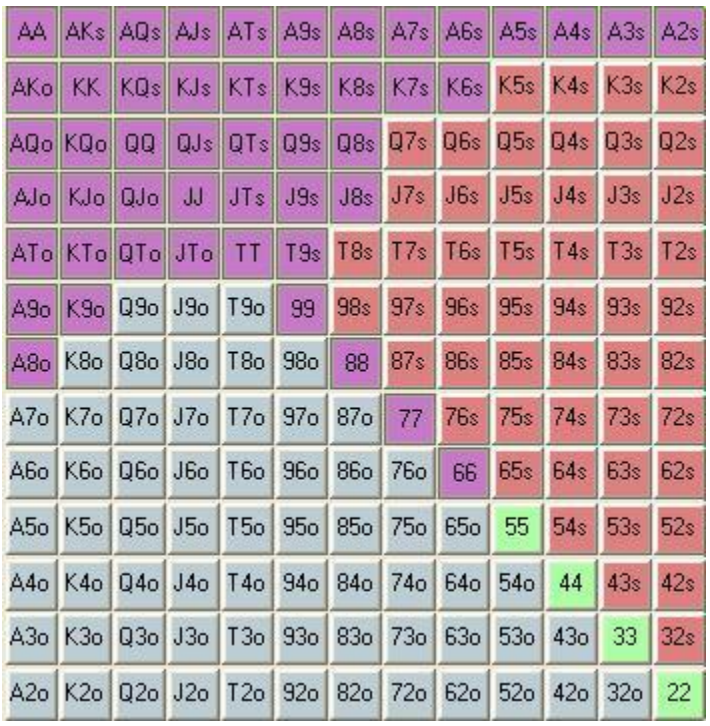

Рисунок 2.4 - Лузовый ТАГ (VPIP = 24 AF = 20)

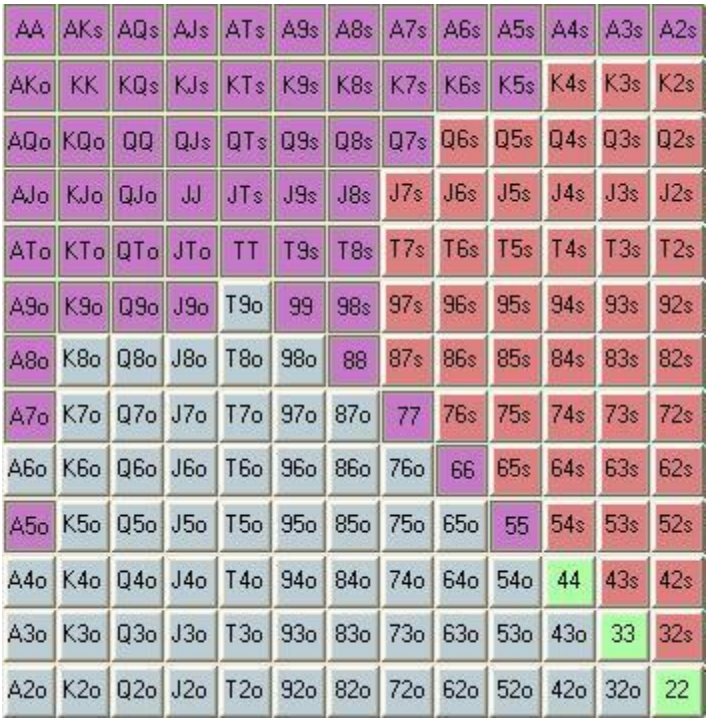

Рисунок 2.5 - Нормальный ЛАГ (VPIP = 29, AF = 24)

Диапазоны стратовых рук помогают скорректировать свою стратегию игры против того или иного типа противника. [15]

ПРИЛОЖЕНИЕ 3. Листинг программы

Файл Form1.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Drawing.Imaging;
using System.Diagnostics;

namespace Pr_poker
{
 public partial class Form1 : Form
 {
  private Bitmap BMP;
  private bool b_exit;
  public Form1()
  {
   InitializeComponent();
  }
  private void Form1_Load(object sender, EventArgs e)
  {

  }
  private void Out_Info(Info info)
  {
   string Odds_str = "";
   Odds_str += "\r\nВыигрыш: " + (info.Odds[0] * 100.0).ToString("0.00") + "%";
   Odds_str += "\r\nНичья: " + (info.Odds[2] * 100.0).ToString("0.00") + "%";
   Odds_str += "\r\nПроигрыш: " + (info.Odds[1] * 100.0).ToString("0.00") + "%";
   string states_str = "";
   if (info.StatePlayers != null)
   {
    int n = info.StatePlayers.Count() - 1;
    for (int i = 0; i < n; i++)
    {
     states_str += "\r\n " + (i + 1).ToString() + ") " + info.StatePlayers[i];
    }
   }
   label1.Text = "На руках: " + info.playerCards;
   label2.Text = "На столе: " + info.tableCards;
   label3.Text = Odds_str;
   label4.Text = "Готов: " + (info.ready ? "Да" : "Нет");
   label5.Text = states_str;
   pictureBox1.Image = BMP;
  }
  private void button1_Click(object sender, EventArgs e)
  {
   button1.Enabled = false;
   b_exit = true;
   button2.Enabled = true;
   BMP = new Bitmap(490, 365);
   Recognize rec = new Recognize(6);
```

```
   Poker_Class poker = new Poker_Class(6);
   while (b_exit)
   {
    Graphics GH = Graphics.FromImage(BMP as Image);
    GH.CopyFromScreen(0, 0, 0, 0, BMP.Size);

    Info info = rec.GetInfo(BMP);

    Info info2 = poker.AddInfo(info);
    var action = poker.PlayMax69(info2);

    sys.SetAction(action);

    Out_Info(info2);
    sys.pause(1);
   }
   BMP.Dispose();
   button1.Enabled = true;
  }
  private void button2_Click(object sender, EventArgs e)
  {
   button2.Enabled = false;
   b_exit = false;
  }
  private void Form1_FormClosed(object sender, FormClosedEventArgs e)
  {
   b_exit = false;
   Application.Exit();
  }
 }
```

Файл Poker_Class.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Drawing;
using HoldemHand;

namespace Pr_poker
{
 public class Info
 {
  public string playerCards;
  public string tableCards;
  public int position;
  public bool ready;
  public string[] StatePlayers;
  public double[] Odds;
  public double PotOdds3;
  public double PotOdds4;
  public Info(string playerCards, string tableCards, int position, bool ready, string[]
StatePlayers,
   double[] Odds, double PotOdds3, double PotOdds4)
  {
   this.playerCards = playerCards;
```

```
  this.tableCards = tableCards;
  this.position = position;
  this.ready = ready;
  this.StatePlayers = StatePlayers;
  this.Odds = Odds;
  this.PotOdds3 = PotOdds3;
  this.PotOdds4 = PotOdds4;
 }
}
public class Poker_Class
{
 public Recognize rec;
 string boardCards0 = "";
 string boardCards1 = "_";
 string playerCards0 = "";
 string playerCards1 = "_";
 public Info info;
 public Poker_Class(int ColPlayers)
 {
  rec = new Recognize(ColPlayers);
 }
 public static string IntToStrCard(int iNum)
 {
  int value = iNum % 13;
  if (value < 8)
  {
   return (value + 2).ToString();
  }
  else
  {
   switch (value)
   {
    case 8:
     return "T";
    case 9:
     return "J";
    case 10:
     return "Q";
    case 11:
     return "K";
    default:
     return "A";
   }
  }
 }
 public static double GetPotOdds3(string playerCards, string boardCards)
 {
  if (boardCards.Length != 9)
   return 0;
  ulong PlayerMask = Hand.ParseHand(playerCards);
  ulong BoardMask = Hand.ParseHand(boardCards);
  long count = 0;
  long PotWins = 0;
  double PotOdds = 0;
  var BoardList4 = Hand.Hands(BoardMask, PlayerMask, 4);
  foreach (ulong Board4 in BoardList4)
  {
   long PlayerWins = 0, OpponentWins = 0;
   long count1 = 0;
```

```
  var BoardList5 = Hand.Hands(Board4, PlayerMask, 5);
  foreach (ulong Board5 in BoardList5)
  {
   var OpponentList = Hand.Hands(0UL, Board5 | PlayerMask, 2);
   foreach (ulong OpponentMask in OpponentList)
   {
    uint PlayerHandValue = Hand.Evaluate(Board5 | PlayerMask, 7);
    uint OpponentHandValue = Hand.Evaluate(Board5 | OpponentMask, 7);
    if (PlayerHandValue > OpponentHandValue)
    {
     PlayerWins++;
    }
    else if (PlayerHandValue < OpponentHandValue)
    {
     OpponentWins++;
    }
    count1++;
   }
  }
  if ((double)PlayerWins / count1 > 0.9)
  {
   PotWins++;
  }
  count++;
 }
 PotOdds = (double)PotWins / count;
 return PotOdds;
}
private static double GetPotOdds4(string playerCards, string boardCards)
{
 if (boardCards.Length != 12)
  return 0;
 double PotOdds = 0;
 ulong PlayerMask = Hand.ParseHand(playerCards);
 ulong BoardMask = Hand.ParseHand(boardCards);
 long PlayerWins = 0, OpponentWins = 0;
 long count = 0;
 long PotWins = 0;
 var BoardList5 = Hand.Hands(BoardMask, PlayerMask, 5);
 foreach (ulong Board5 in BoardList5)
 {
  PlayerWins = 0;
  OpponentWins = 0;
  long count1 = 0;
  var OpponentList = Hand.Hands(0UL, Board5 | PlayerMask, 2);
  foreach (ulong OpponentMask in OpponentList)
  {
   uint PlayerHandValue = Hand.Evaluate(Board5 | PlayerMask, 7);
   uint OpponentHandValue = Hand.Evaluate(Board5 | OpponentMask, 7);
   if (PlayerHandValue > OpponentHandValue)
   {
    PlayerWins++;
   }
   else if (PlayerHandValue < OpponentHandValue)
   {
    OpponentWins++;
   }
   count1++;
  }
```

```
  if ((double)PlayerWins / count1 > 0.95)
  {
   PotWins++;
  }
  count++;
 }
 PotOdds = (double)PotWins / count;
 return PotOdds;
}
private static double[] GetOdds(string playerCards, string boardCards)
{
 double[] Odds = new double[3];
 ulong PlayerMask = Hand.ParseHand(playerCards);
 ulong BoardMask = Hand.ParseHand(boardCards);
 long PlayerWins = 0, OpponentWins = 0;
 long count = 0;
 var OpponentList = Hand.Hands(0UL, BoardMask | PlayerMask, 2);
 foreach (ulong OpponentMask in OpponentList)
 {
  var BoardList = Hand.RandomHands(BoardMask, PlayerMask | OpponentMask, 5, 1000);
  foreach (ulong Board in BoardList)
  {
   uint PlayerHandValue = Hand.Evaluate(Board | PlayerMask, 7);
   uint OpponentHandValue = Hand.Evaluate(Board | OpponentMask, 7);
   if (PlayerHandValue > OpponentHandValue)
   {
    PlayerWins++;
   }
   else if (PlayerHandValue < OpponentHandValue)
   {
    OpponentWins++;
   }
   count++;
  }
 }
 Odds[0] = (double)PlayerWins / count;
 Odds[1] = (double)OpponentWins / count;
 Odds[2] = 1.0 - Odds[0] - Odds[1];
 return Odds;
}
private bool IsLevel1(Hand.PocketHand169Enum PoketHand)
{
 return
  PoketHand == Hand.PocketHand169Enum.Pocket77 ||
  PoketHand == Hand.PocketHand169Enum.Pocket88 ||
  PoketHand <= Hand.PocketHand169Enum.PocketATo &&
  PoketHand >= Hand.PocketHand169Enum.PocketAJs;
}
private bool IsLevel2(Hand.PocketHand169Enum PoketHand)
{
 return
  PoketHand == Hand.PocketHand169Enum.Pocket99 ||
  PoketHand == Hand.PocketHand169Enum.PocketTT ||
  PoketHand == Hand.PocketHand169Enum.PocketAQo ||
  PoketHand == Hand.PocketHand169Enum.PocketAQs;
}
private bool IsLevel3(Hand.PocketHand169Enum PoketHand)
{
 return
```

```
  PoketHand <= Hand.PocketHand169Enum.PocketAA &&
  PoketHand >= Hand.PocketHand169Enum.PocketJJ ||
  PoketHand == Hand.PocketHand169Enum.PocketAKo ||
  PoketHand == Hand.PocketHand169Enum.PocketAKs;
}
private double GetDecide()
{

 return 0;
}
private double GetDecideByHand()
{
 var PoketHand = Hand.PocketHand169Type(Hand.ParseHand(info.playerCards));
 bool level3 = IsLevel3(PoketHand);
 bool level2 = IsLevel2(PoketHand) || level3;
 bool level1 = IsLevel1(PoketHand) || level2;
 var iPos = info.position;
 if (iPos == 1 && level3 || iPos == 2 && level2 || iPos == 3 && level1)
 {
  return 2.5;
 }
 else
 {
  return -1;
 }
}
public Info AddInfo(Info info)
{
 this.info = info;
 var Odds = GetOdds(info.playerCards, info.tableCards);
 var PotOdds3 = GetPotOdds3(info.playerCards, info.tableCards);
 var PotOdds4 = GetPotOdds4(info.playerCards, info.tableCards);
 Info info_temp = new Info(
  info.playerCards,
  info.tableCards,
  info.position,
  info.ready,
  info.StatePlayers,
  Odds,
  PotOdds3,
  PotOdds4);
 return info_temp;
}
public double PlayMax69(Info info)
{
 this.info = info;
 if (info.playerCards.Length == 6)
 {
  if (!info.ready)
  {
   return GetDecideByHand();
  }
  else
  {
   return GetDecide();
  }
 }
 else
 {
```

```
    return double.NaN;
   }
  }
 }
}
```

Файл Recognize.cs:

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

namespace Pr_poker
{
 public class Recognize
 {
  public Bitmap BMP = new Bitmap(490, 365);
  private int iColPlayers;
  public static int w2 = 15;
  public static int h2 = 21;
  public Bitmap[] bmp = new Bitmap[52];
  public List<et> ets = new List<et>();
  public et fold_et;
  public et ready_et;
  public et dollar_et;
  public et button_et;
  public List<et> symbols = new List<et>();
  public List<et> Pot_symbols = new List<et>();
  public static double porog_corr = 0.8;
  private class ish_card
  {
   public int M;
   public int N;
   public double W_mean;
   public double[,] W;
   public string suit;
   public ish_card(Bitmap bmp, int x, int y)
   {
    M = w2;
    N = h2;
    W = new double[M, N];
    double r = 0;
    double g = 0;
    double b = 0;
    for (int i = 0, xx = x; i < M; i++, xx++)
    {
     for (int j = 0, yy = y; j < N; j++, yy++)
     {
      W[i, j] = (bmp.GetPixel(xx, yy).R +
           bmp.GetPixel(xx, yy).G +
           bmp.GetPixel(xx, yy).B) / 3.0 / 255.0;
      r += bmp.GetPixel(xx + M, yy).R;
      g += bmp.GetPixel(xx + M, yy).G;
      b += bmp.GetPixel(xx + M, yy).B;
     }
    }
```

```
  W_mean = 0;
  for (int i = 0; i < M; i++)
  {
   for (int j = 0; j < N; j++)
   {
    W_mean += W[i, j];
   }
  }
  W_mean /= (M * N);
  r /= (M * N);
  g /= (M * N);
  b /= (M * N);
  double c = 1.2;
  if (r > g * c && r > b * c)
   suit = "h";
  else if (g > r * c && g > b * c)
   suit = "c";
  else if (b > r * c && b > g)
   suit = "d";
  else
   suit = "s";
 }
}
public class et
{
 public string ch;
 public int m;
 public int n;
 public double f_mean;
 public double[,] f;
 public et(Bitmap bmp, string ch)
 {
  this.ch = ch;
  m = bmp.Width;
  n = bmp.Height;
  f = new double[m, n];
  for (int i = 0; i < m; i++)
  {
   for (int j = 0; j < n; j++)
   {
    f[i, j] = (bmp.GetPixel(i, j).R +
          bmp.GetPixel(i, j).G +
          bmp.GetPixel(i, j).B) / 3.0 / 255.0;
   }
  }
  f_mean = 0;
  for (int i = 0; i < m; i++)
  {
   for (int j = 0; j < n; j++)
   {
    f_mean += f[i, j];
   }
  }
  f_mean /= (m * n);
 }
 public et(Bitmap bmp, string ch, int x, int y, int w, int h)
 {
  this.ch = ch;
  m = w;
```

```
  n = h;
  f = new double[m, n];
  for (int i = 0; i < m; i++)
  {
   for (int j = 0; j < n; j++)
   {
    f[i, j] = (bmp.GetPixel(x + i, y + j).R +
         bmp.GetPixel(x + i, y + j).G +
         bmp.GetPixel(x + i, y + j).B) / 3.0 / 255.0;
   }
  }
  f_mean = 0;
  for (int i = 0; i < m; i++)
  {
   for (int j = 0; j < n; j++)
   {
    f_mean += f[i, j];
   }
  }
  f_mean /= (m * n);
 }
 public double corr2(Bitmap bmp_temp, int x, int y)
 {
  et ishod = new et(bmp_temp, "", x, y, this.m, this.n);
  double SUM1 = 0;
  double SUM2 = 0;
  double SUM3 = 0;
  for (int i = 0; i < this.m; i++)
  {
   for (int j = 0; j < this.n; j++)
   {
    double temp_W = ishod.f[i, j];
    double AA = (temp_W - ishod.f_mean);
    double BB = (this.f[i, j] - this.f_mean);
    SUM1 += AA * BB;
    SUM2 += AA * AA;
    SUM3 += BB * BB;
   }
  }
  return SUM1 / Math.Sqrt(SUM2 * SUM3);
 }
}
public string RecognizeCards(int _x, int _y, int ot, int N, double koef_corr)
{
 ish_card[] ishod = new ish_card[N];
 for (int i = 0; i < N; i++)
 {
  ishod[i] = new ish_card(this.BMP, _x + i * (30 + ot), _y);
 }
 string[] str = new string[N];
 for (int d = 0; d < N; d++)
 {
  double corr_max_xy = -1.0;
  et rasp = ets[0];
  for (int k = 0; k < ets.Count; k++)
  {
   double SUM1 = 0;
   double SUM2 = 0;
   double SUM3 = 0;
```

```
   for (int i = 0; i < w2; i++)
   {
    for (int j = 0; j < h2; j++)
    {
     double temp_W;
     try
     {
      temp_W = ishod[d].W[i, j];
     }
     catch
     {
      temp_W = ishod[d].W_mean;
     }
     double AA = (temp_W - ishod[d].W_mean);
     double BB = (ets[k].f[i, j] - ets[k].f_mean);
     SUM1 += AA * BB;
     SUM2 += AA * AA;
     SUM3 += BB * BB;
    }
   }
   double temp_corr = SUM1 / Math.Sqrt(SUM2 * SUM3);
   if (temp_corr > corr_max_xy)
   {
    corr_max_xy = temp_corr;
    rasp = ets[k];
   }
  }
  if (corr_max_xy >= koef_corr)
  {
   str[d] = rasp.ch + ishod[d].suit;
  }
 }
 string text = "";
 foreach (string s in str)
 {
  if (s != null &&  s.Length == 2)
  {
   text += s + " ";
  }
 }
 return text;
}
public class player_coordinates
{
 public class coordinates
 {
  public int x { get; private set; }
  public int y { get; private set; }
  public coordinates(int x, int y)
  {
   this.x = x;
   this.y = y;
  }
 }
 public coordinates action;
 public coordinates bet;
 public coordinates button;
 public player_coordinates(int x, int y, int x1, int y1, int x2, int y2)
 {
```

```
  action = new coordinates(x, y);
  bet = new coordinates(x1, y1);
  button = new coordinates(x2, y2);
 }
}
public player_coordinates[] players;
private bool IsReady(double koef_corr)
{
 double temp_corr = ready_et.corr2(BMP, 220, 287);
 if (temp_corr >= koef_corr)
 {
  return true;
 }
 return false;
}
private void GetButtonPos()
{

}
private string GetPlayerState(int ind)
{
 if (fold_et.corr2(BMP, players[ind].action.x, players[ind].action.y) >= porog_corr)
 {
  return "fold";
 }
 else
 {
  string str_bet = "";
  int x = players[ind].bet.x;
  int y = players[ind].bet.y;
  int i = x - 40;
  int n = x + 40;
  for (; i < n; i++)
  {
   for (int j = 0; j < symbols.Count; j++)
   {
    if (symbols[j].corr2(BMP, i, y) > porog_corr)
    {
     i += symbols[j].m - 1;
     str_bet += symbols[j].ch;
     break;
    }
   }
  }
  if (str_bet == "")
  {
   str_bet =  "check";
  }
  return str_bet;
 }
}
private string[] GetStatePlayers()
{
 int n = players.Count() - 1;
 var steps = new string[n];
 for (int i = 0; i < n; i++)
 {
  steps[i] = GetPlayerState(i);
 }
```

```
 return steps;
}
private int GetPosition()
{
 int n = players.Count() - 1;
 for (int i = 0; i < n; i++)
 {
  if (button_et.corr2(BMP, players[i].button.x, players[i].button.y) >= porog_corr)
  {
   return i;
  }
 }
 return n;
}
public Info GetInfo(Bitmap BMP)
{
 //this.BMP.Dispose();
 this.BMP = BMP;
 int pos = GetPosition(); // позиция батона
 var sp = GetStatePlayers(); // состояния игроков
 bool r = IsReady(0.5); // готовность
 string pc = RecognizeCards(218, 220, 0, 2, 0.3); // карты игрока
 string tc = RecognizeCards(166, 137, 2, 5, 0.7); // карты стола
 Info info_temp = new Info(pc, tc, pos, r, sp, null, 0, 0);
 return info_temp;
}
public Recognize(int iColPlayers)
{
 this.iColPlayers = iColPlayers;
 players = new player_coordinates[iColPlayers];
 if (iColPlayers == 6)
 {
  players[0] = new player_coordinates(45, 202, 128, 190, 124, 205);
  players[1] = new player_coordinates(45,  99, 141, 115, 109, 125);
  players[2] = new player_coordinates(223, 59, 239, 95,  204, 94);
  players[3] = new player_coordinates(426, 99, 324, 115, 375, 128);
  players[4] = new player_coordinates(426,202, 336, 190, 355, 205);
 }
 else
 {

 }
 Bitmap bmp_et;
 for (int i = 0; i < 52; i++)
 {
  bmp_et = new Bitmap("1/" + i.ToString() + ".png");
  ets.Add(new et(bmp_et, Poker_Class.IntToStrCard(i)));
  bmp_et.Dispose();
 }
 bmp_et = new Bitmap("1/button.bmp");
 button_et = new et(bmp_et, "button");
 bmp_et.Dispose();
 bmp_et = new Bitmap("1/fold.png");
 fold_et = new et(bmp_et, "fold");
 bmp_et.Dispose();
 bmp_et = new Bitmap("1/ready.bmp");
 ready_et = new et(bmp_et, "ready");
 bmp_et.Dispose();
 bmp_et = new Bitmap("symbols/$.png");
```

```
    dollar_et = new et(bmp_et, "ready");
    bmp_et.Dispose();
    for (int i = 0; i < 9; i++)
    {
     bmp_et = new Bitmap("symbols/" + i.ToString() + ".png");
     symbols.Add(new et(bmp_et, i.ToString()));
     bmp_et.Dispose();
    }
    bmp_et = new Bitmap("symbols/,.png");
    symbols.Add(new et(bmp_et, ","));
    bmp_et.Dispose();
   }
  }
}
```

Файл sys.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Runtime;
using System.Runtime.InteropServices;
using System.Windows.Forms;
using System.Drawing;

namespace Pr_poker
{
 public static class sys
 {
  private const UInt32 MOUSEEVENTF_LEFTDOWN = 0x0002;
  private const UInt32 MOUSEEVENTF_LEFTUP = 0x0004;
  [DllImport("user32.dll")]
  public static extern void mouse_event(UInt32 dwFlags, UInt32 dx, UInt32 dy, UInt32
dwData, IntPtr dwExtraInfo);
  public static void SendClick(Point location)
  {
   sys.pause(10);
   Cursor.Position = location;
   sys.pause(10);
   mouse_event(MOUSEEVENTF_LEFTDOWN, 0, 0, 0, new IntPtr());
   sys.pause(10);
   mouse_event(MOUSEEVENTF_LEFTUP, 0, 0, 0, new IntPtr());
  }
  private static class buttons
  {
   public static Point Fold = new Point(279, 338);
   public static Point AllIn = new Point(454, 294);
   public static Point Raise = new Point(442, 339);
  }
  public static void SetAction(double action)
  {
   if (action < 0)
   {
    SendClick(buttons.Fold);
   }
   else if (double.IsPositiveInfinity(action))
   {
```

```
     SendClick(buttons.AllIn);
     SendClick(buttons.Raise);
    }
   }
   public static void pause(int value)
   {
    System.Diagnostics.Stopwatch sw = new System.Diagnostics.Stopwatch();
    sw.Start();
    while (sw.ElapsedMilliseconds < value)
     Application.DoEvents();
   }
  }
 }
```