Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и кибербезопасности
Высшая школа программной инженерии

К У Р С О В А Я Р А Б О Т А

Проектирование и программирование модуля игрока подкидного дурака

по дисциплине «Дополнительные главы программирования»

Выполнил

студент гр. 5130903/40001 М.М. Алексеев

Руководитель

доцент, к.т.н. А.В. Щукин

«___» _____ 2026 г.

Санкт-Петербург

2026

ОГЛАВЛЕНИЕ

ГЛАВА 1. ВВЕДЕНИЕ

Данная курсовая работа посвящена разработке модуля игрока для карточной игры «Подкидной дурак» на языке программирования C#. Актуальность темы обусловлена необходимостью исследования методов построения игровых алгоритмов в условиях неполной информации и разработки стратегий принятия решений, обеспечивающих преимущество перед соперником.

«Подкидной дурак» – это карточная игра, в классическом варианте использующая колоду из 36 карт (от шестерки до туза). В начале партии каждому игроку раздается по шесть карт, после чего определяется козырная масть – масть последней открытой карты в колоде. Цель игры заключается в том, чтобы первым избавиться от всех своих карт. Игрок, оставшийся с картами в конце партии, считается проигравшим.

Игровой процесс строится на чередовании фаз атаки и защиты. Атакующий игрок выкладывает одну или несколько карт одного достоинства. Защищающийся обязан побить каждую карту либо картой той же масти более высокого достоинства, либо козырной картой. В случае успешной защиты право хода переходит к оборонявшемуся. Если же игрок не может или не желает отбиться, он забирает все карты со стола, после чего ход остается за атакующим. Особенностью варианта «подкидной» является возможность подбрасывания дополнительных карт того же достоинства, что и уже находящиеся на столе, при соблюдении ограничений по количеству карт на столе у защищающегося игрока.

В рамках данной работы рассматривается модификация игры для двух игроков. Возможность первого хода распределяется равномерно между соперниками, что обеспечивает корректность и объективность оценки разработанного алгоритма. Существенным ограничением является требование честной игры: алгоритм не обладает доступом к скрытой информации о картах соперника и оперирует только теми данными, которые доступны человеку в

реальной партии – собственными картами, сыгранными картами и картами, открытыми в случае неудачной защиты противника. При этом алгоритм может использовать вычислительные возможности компьютера для хранения и анализа большого объема информации, а также для выполнения сложных расчетов.

В качестве исходного материала для разработки были предоставлены 4 файла с исходным кодом классов и структур MPlayer1, MPlayer2, MTable, SCard, SCardPair и прочих вспомогательным структур, классов и перечислений. Класс MTable содержит в себе точку входа, которая запускает 10000 игр с игроками MPlayer1 и MPlayer2, проверяет корректность проведенных ходов, ведет логи игры и подводит статистику по проведенным играм.

Целью работы является разработка алгоритмов атаки, подкидывания и обороны, обеспечивающих вероятность победы более 0,5 при серии игр против соперника заданного уровня.

Для достижения поставленной цели необходимо решить следующие задачи:

- спроектировать концепцию общего стиля игры бота и разработать стратегии атаки, подкидывания и обороны;
- реализовать разработанные алгоритмы на языке программирования C# на основе предоставленного шаблона MPlayer1;
- разработать и реализовать упрощенный алгоритм бота для проведения сравнительного анализа и проверки выигрышности предложенной стратегии;
- провести серию тестовых партий, выполнить анализ полученных результатов и сформулировать выводы по итогам работы.

Таким образом, работа направлена на исследование принципов построения интеллектуального игрового агента в условиях неполной

информации и оценку эффективности разработанных стратегий в рамках классической карточной игры.

ГЛАВА 2. ПРОЕКТИРОВАНИЕ

2.1. Концепции стратегии

Проектирование модуля игрока основано на трех ключевых концепциях: учёт сыгранных карт, деление игры на фазы и правило «ходить только по одной карте».

Первая концепция заключается в запоминании всех карт, вышедших из игры. Бот фиксирует сыгранные карты, а также карты, раскрытые соперником в случае неудачной защиты. На основании этой информации уменьшается множество потенциальных карт, которые могут находиться на руке противника. По мере развития партии это множество сужается. В финальной стадии игры, когда число возможных карт соперника становится не больше шести, появляется возможность точно определить его руку. Это позволяет выполнять гарантированную атаку картой, которую соперник не сможет побить.

Вторая концепция – деление игры на фазы, основываясь на количества карт в сбросе. В зависимости от текущей фазы изменяется степень агрессивности и ценность козырных карт. В начальной части партии козыри преимущественно сохраняются для использования в поздних раундах, когда они обладают наибольшей стратегической ценностью.

Третья концепция – правило «ходить только по одной карте». Даже при наличии нескольких карт одного ранга бот начинает атаку одной картой. Это вынуждает соперника раскрывать дополнительные карты при защите либо при взятии, что увеличивает объем доступной информации. Если бы атака выполнялась сразу несколькими картами одного достоинства, противник мог бы забрать их без раскрытия структуры своей руки, что снижает информативность игрового взаимодействия.

Перечисленные концепции формируют основу дальнейших стратегий атаки и обеспечивают сочетание информационного преимущества и контролируемой агрессии.

2.2. Стратегия начальной атаки

Логика первого хода в раунде реализуется как последовательная проверка трех стратегий, упорядоченных по приоритету:

1. Атака гарантированно неотбиваемой картой (при точном знании руки соперника);
2. Атака через формирование набора карт одного ранга (пары, тройки, четверки);
3. Атака минимальной картой.

2.2.1. Атака гарантированно неотбиваемой картой

Если на основе анализа сыгранных карт количество потенциальных карт соперника становится меньше либо равно шести, выполняется перебор карт в руке бота. Для каждой карты проверяется возможность ее побития соперником. Если данную карту соперник точно не сможет отбить, и карта имеет ранг ниже, чем предыдущая карта, которую соперник не мог бы отбить, то бот отдает ей предпочтение.

Таким образом, бот старается найти две карты с наименьшим рангом, которые соперник не может отбить – козырная и некозырная. Предпочтение отдается некозырной, если такой нет, бот ходит козырной. Данная стратегия может быть применена только в конце игры, поэтому поведение бота становится более агрессивным, и он может сходить козырем. В случае, если обе карты не были найдены, бот переходим к следующей стратегии.

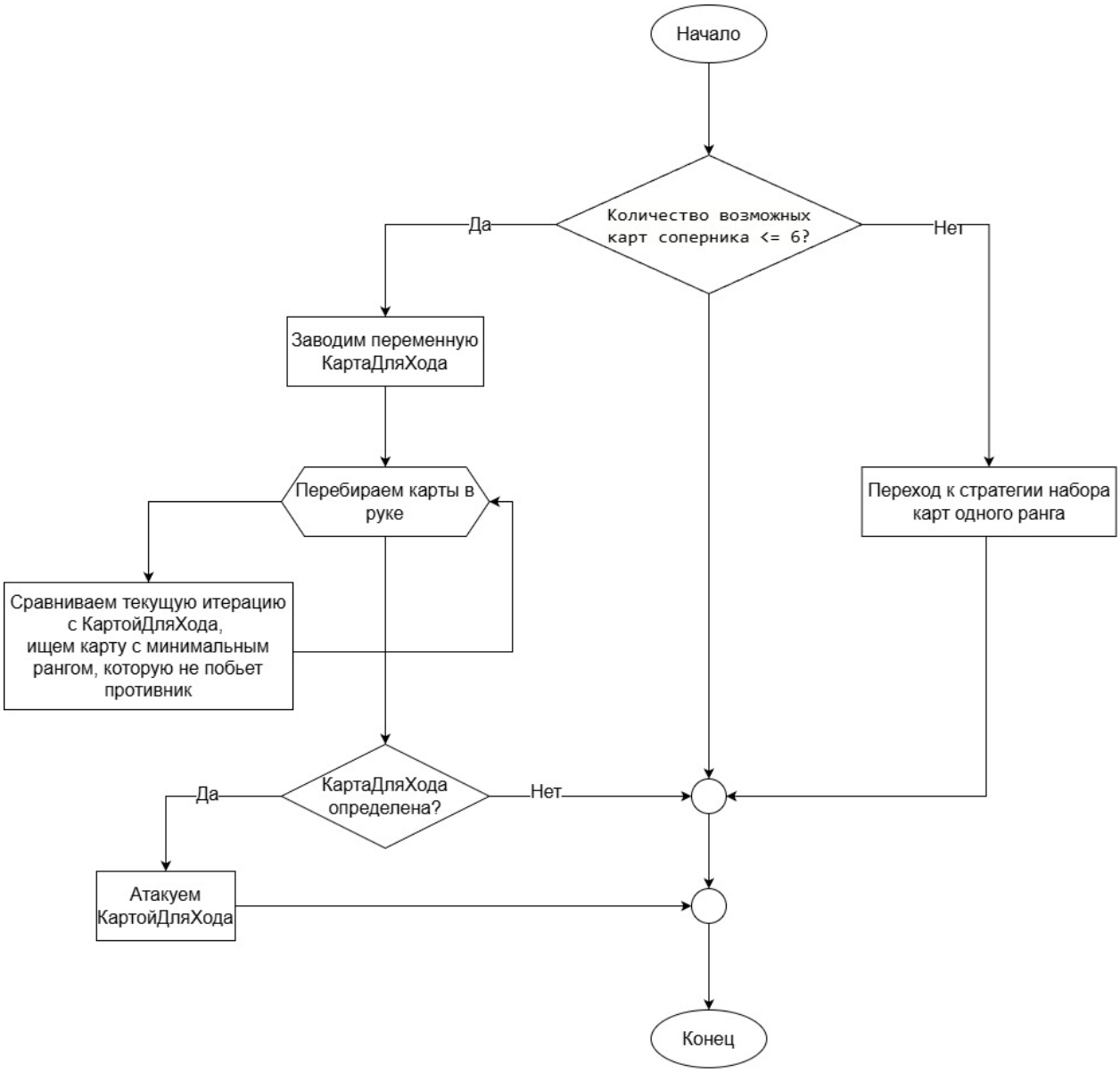

Рисунок 1. Диаграмма стратегии Атака гарантированно неотбиваемой картой

2.2.2. Стратегия набора карт одного ранга

Если гарантированная атака невозможна, выполняется поиск наборов карт одного ранга (от двух до четырех). При этом в начальной фазе игры (пока в сброс не ушла половина карт) козырные карты исключаются из формирования набора, поскольку их целесообразно сохранить.

Из всех возможных наборов карт одного ранга выбирается набор с минимальным рангом. После выбора бот атакует только одной картой из найденного набора. Остальные карты запоминаются для подкидывания в текущем раунде.

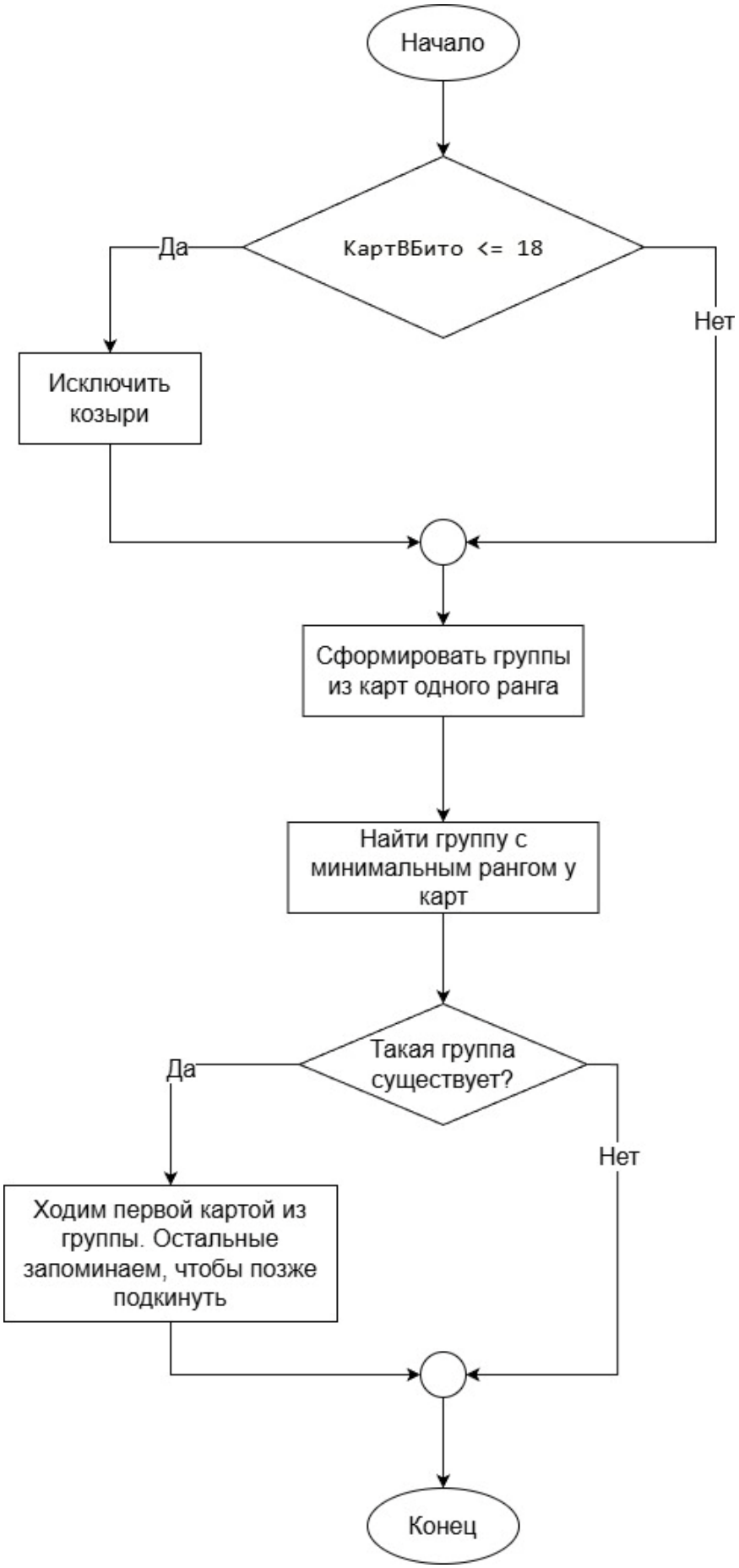

Рисунок 2. Стратегия набора карт одного ранга

2.2.3. Атака минимальной картой

Если ни одна из предыдущих стратегий не применима, выполняется выбор минимальной карты из текущей руки бота независимо от масти. Бот ищет две карты минимального ранга – козырную и некозырную. Предпочтение отдается некозырной карте, за неимением таковой, бот ходит козырной минимального ранга.

2.3. Подкидывание карт

После выполнения начальной атаки и защиты соперника наступает фаза подкидывания карт. Подкидывание возможно только при выполнении следующих условий:

- у бота имеются карты на руке;
- количество карт на столе не превышает допустимого лимита (не более шести и не более числа карт у соперника);
- подкидываемая карта имеет ранг, уже присутствующий на столе.

Стратегии подкидывания упорядочены по убыванию приоритета и логически продолжают стратегии первого хода.

2.3.1. Продолжение ранее сформированного набора карт одного ранга

Если при первом ходе был сформирован набор карт одного ранга (пара, тройка или четверка), бот продолжает использовать его в фазе подкидывания. Карты из набора подкидываются по одной. Такой подход сохраняет правило «ходить по одной карте» и позволяет постепенно раскрывать структуру руки соперника.

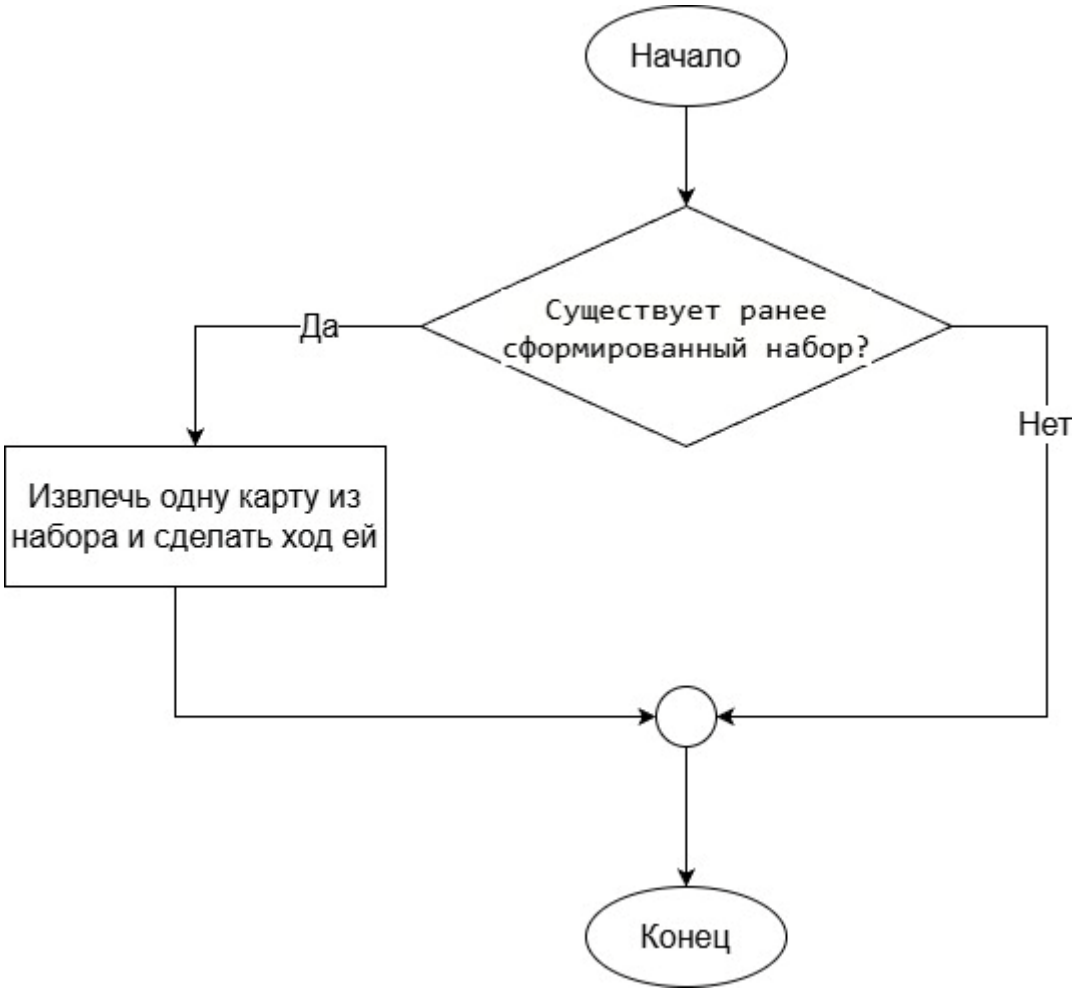

Рисунок 3. Продолжение ранее сформированного набора карт одного ранга

2.3.2. Формирование нового набора карт

Если ранее сформированный набор отсутствует, бот анализирует текущую ситуацию на столе. Если среди карт на руке имеются такие ранги, которые уже присутствуют на столе, и из них можно сформировать набор, запускается стратегия набора карт.

В первой половине игры (пока в сбросе количество карт меньше или равно 18) козырные карты сохраняются и не участвуют в формировании набора. Из допустимых наборов выбирается набор минимального ранга, после чего подкидывается только одна карта, а остальные сохраняются для последующего подкидывания.

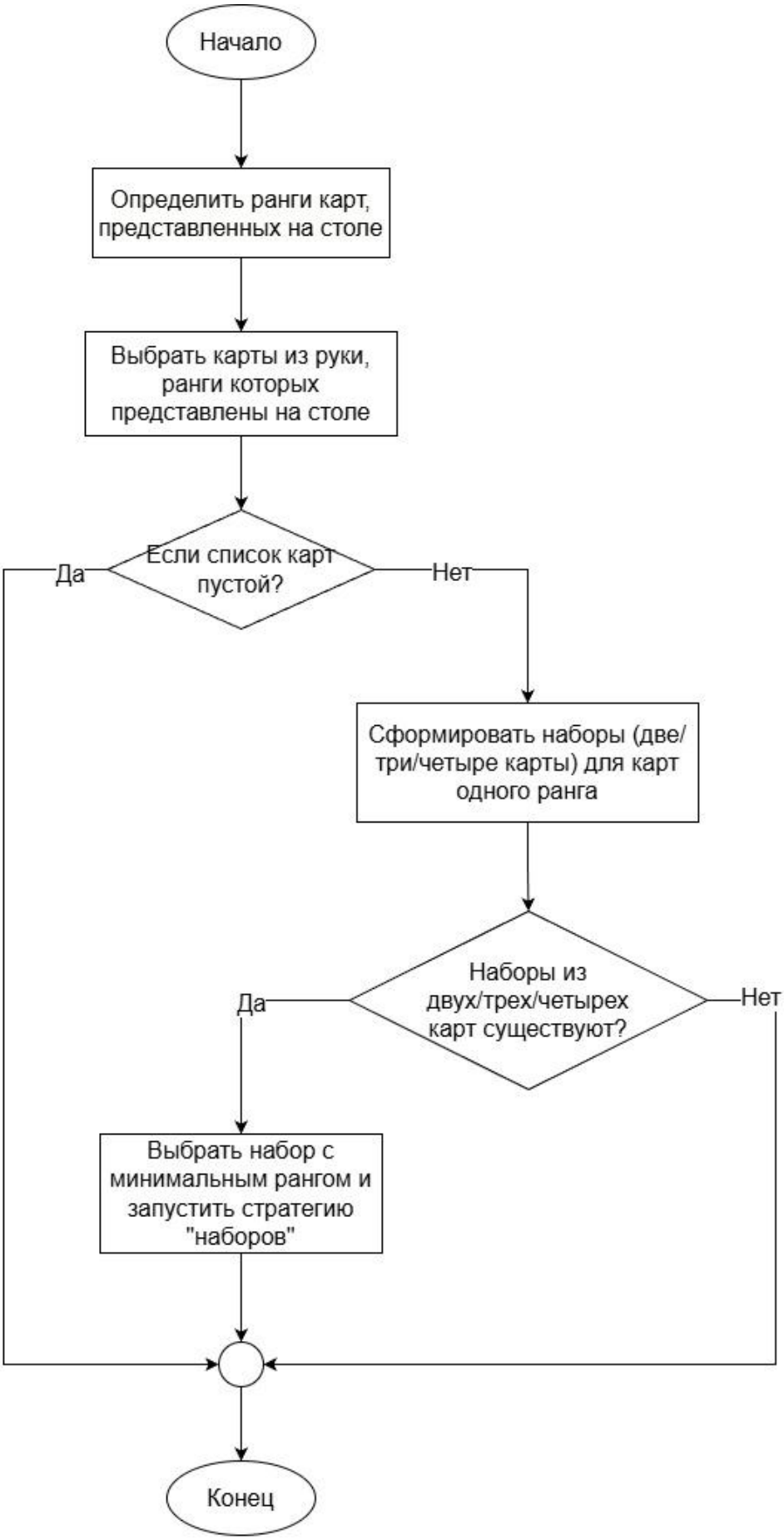

Рисунок 4. Формирование нового набора карт

2.3.3. Стратегия минимальной подходящей карты

Если сформировать набор невозможно, бот ищет одну подходящую карту для подкидывания. Подходящей считается карта, ранг которой уже присутствует на столе.

В первой половине партии бот избегает использования козырей. Если рука соперника фактически известна, бот стремится найти карту минимального ранга, которую соперник не сможет отбить. При отсутствии такой карты либо при недостатке информации о точной руке соперника выбирается минимальная допустимая карта независимо от возможности ее побития.

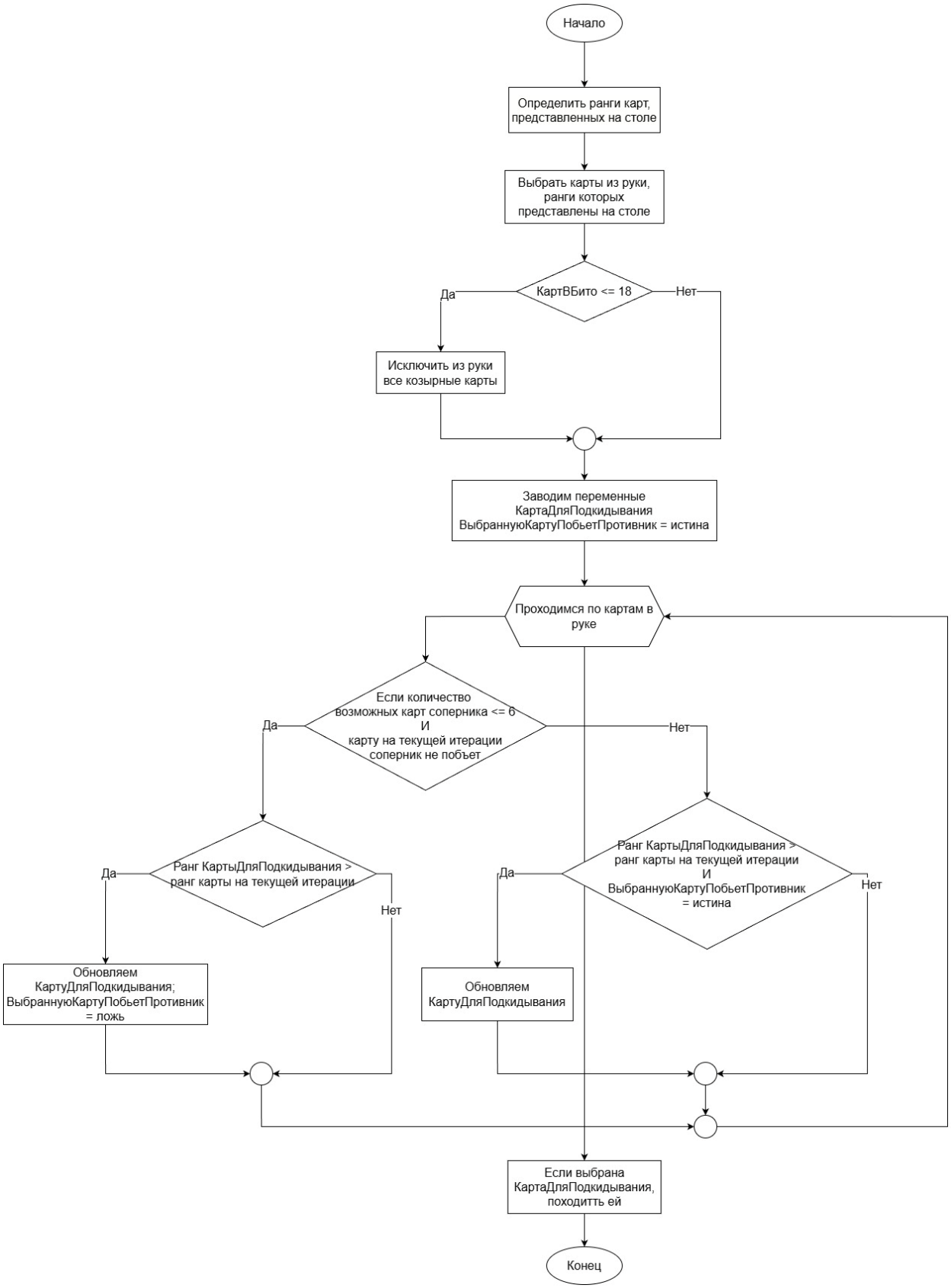

Рисунок 5. Стратегия подкидывания минимальной подходящей карты

2.4. Защита бота

Фаза защиты реализует наиболее консервативную часть стратегии бота. В отличие от атакующих действий, где используется система приоритетов, при защите применяется единый принцип – выбор карты наименьшего ранга, способной побить карту соперника.

Особенностью алгоритма является фазовая адаптация поведения. В первой трети игры, когда количество карт в сбросе не превышает 12, бот придерживает козырные карты. Если для защиты требуется использовать козырь, а альтернативной некозырной карты нет, бот предпочитает принять карты соперника, сохранив козырь для более поздних этапов партии. Это связано с тем, что в середине и конце игры бот сможет использовать данные карты для более агрессивного нападения, применяя стратегию наборов.

Если же партия вышла за пределы первой трети, либо существует возможность побить карту соперника некозырной картой, бот выбирает карту минимального ранга из всех допустимых.

2.5. Конец раунда

В конце каждого раунда модуль первого игрока пересчитывает количество карт на руке соперника следующим образом. Пусть:

N_{opp} – число карт на руке соперника в начале завершения раунда;

A – число атакующих карт на столе (нижние карты;

D – число карт, оставшихся в колоде до добора в конце раунда.

Если бот был атакующим игроком, то соперник либо потратил карты на защиту, либо взял карты со стола.

При успешной защите:

$$N'_{opp} = N_{opp} - A$$

При неуспешной защите:

$$N'_{opp} = N_{opp} + A$$

Если атаковал соперник, то соперник в любом случае избавился от A атакующих карт, тогда:

$$N'_{opp} = N_{opp} - A$$

Рассмотрим ситуацию, когда игрокам необходимо добрать карты. Пусть:

N_{me} – количество карт у бота перед добором;

$need_{me}$ – количество карт, которое необходимо добрать боту;

$need_{opp}$ – количество карт, которое необходимо добрать сопернику;

$draw_{opp}$ – итоговое количество карт на руке соперника к началу нового раунда. Необходимо расчитать.

Тогда потребность в доборе рассчитывается как:

$$need_{me} = \max(0, 6 - N_{me})$$

$$need_{opp} = \max\left(0, 6 - N'_{opp}\right)$$

Добор возможен только если $D > 0$. При этом порядок добора зависит от того, кто атаковал в текущем раунде. Если атаку проводил бот, то он добирает первым, а соперник – вторым. Бот получает $\min(need_{me}, D)$ карт, тогда в колоде останется $\max(0, D - \min(need_{me}, D))$. Итого соперник получает:

$$draw_{opp} = \min\left(need_{opp}, \max(0, D - \min(need_{me}, D))\right)$$

Если атаковал соперник, то соперник добирает первым:

$$draw_{opp} = \min\left(need_{opp}, D\right)$$

ГЛАВА 3. ПРОГРАММИРОВАНИЕ

3.1. Общие сведения разработанной программы

Код был написан на языке C# на основе прототипа MPlayer1. Он содержит public методы, которые предоставляет внешний API для модуля MTable. Разберем поля и методы класа MPlayer1.

Название поля	Тип данных	Описание
hand	List<SCard>	Список карт на руке бота.
opponentHand	List<SCard>	Список потенциальных карт, которые могут быть у соперника. Изначально содержит все 36 карт. Как только бот получает карту в свою руку, или какая-то карта выходит из игры, эта карта удаляется из opponentHand. Когда количество карт в списке становится меньше или равно 6, бот может точно узнать карты соперника.

roundAttacker	enum RoundAttacker { ME, OPPONENT }	Показывает атакующего текущего раунда. Необходимо для логики подсчета количества карт соперника.
trumpSuit	Suits	Козырная масть.
batchMemoCards	Queue<KeyValuePair<int, SCard>>	Структура данных, которая отражает карты, которые бот запоминает, при активной стратегии наборов карт.
countBittedCards	int	Количество карт в бито.
deckRemaining	int	Сколько карт осталось в колоде.
opponentCurrentCountCards	int	Количество карт оппонента.

Таблица 1. Поля класса MPlayer1

Прототип метода	Описание
private void InitOpponentHand()	Инициализация потенциальных карт в руке оппонента. Изначально оппонент может иметь любые карты.
private List<int> GetTableRanks(List<SCardPair> table)	Возвращает список уникальных рангов на столе.

private bool OpponentCanBeat(SCard myCard)	Проверяет, может ли оппонент отбить карту. Вызывается в финальных раундах, когда известны точные карты противника.
private SCard? PlayBatchStrategy(List<SCard>? hand = null)	Запускает стратегию наборов карт с одинаковыми рангами. Возвращает карту, с которой надо походить, если такая есть, значит набор был сформирован успешно, и переменная batchMemoCards хранит карты для подкидывания.
public string GetName()	Возвращает имя бота.
public int GetCount()	Возвращает количество карт на руке бота.
public void AddToHand(SCard card)	Хук-функция, которая отрабатывает каждый раз при добавлении новой карты в руку бота.
public List<SCard> LayCards()	Хук-функция, которая отрабатывает при первой атаке бота. Возвращает в список карт, которые бот играет на стол. Внутри функции расписаны 3 стратегии первого хода, которые бот старается применить, начиная с более приоритетной и заканчивая менее.
public bool AddCards(List<SCardPair> table, bool opponentDefensed)	Хук-функция, которая отрабатывает при возможности бота подбросить карты. Также реализует в себе стратегии, описанные в главе проектирования. Возвращает true, если бот подкинул карты, false, если не стал подкидывать.
public bool Defend(List<SCardPair> table)	Хук-функция, которая отрабатывает, когда боту необходимо покрыть карты

	противника. Возвращает true, если бот отбился успешно, false, если бот берет карты в руку по собственному желанию или необходимости.
public void OnEndRound(List<SCardPair> table, bool isDefenseSuccessful)	Хук-функция, которая отрабатывает при завершении каждого раунда, перед добором карт на руки.
public void SetTrump(SCard newTrump)	Хук-функция, вызывается один раз, когда боту передается козы

Таблица 2. Методы класса MPlayer1

3.2. Стратегия атаки

Атака в модуле игрока строится как последовательная система приоритетов: бот пытается выполнить ход, который либо гарантирует выигрыш текущего размена, либо обеспечивает выгодный размен ресурсов и получение дополнительной информации о руке соперника. В большинстве алгоритмов для оптимального хранения «наилучшей» карты используется индекс данной карты в руке бота. Также хранения индекса позволяет удалить карту из руки бота за время $O(1)$.

Далее подробно рассматриваются две наиболее значимые стратегии: «атака гарантированно неотбиваемой картой» и «стратегия набора карт одного ранга».

3.2.1. Атака гарантированно неотбиваемой картой

Бот перебирает все свои карты и отбирает такие, которые соперник точно не сможет побить. Среди найденных кандидатов бот отдельно отслеживает минимальную по рангу некозырную карту и минимальную по рангу козырную карту, которые являются неотбиваемыми. Приоритет отдается некозырной карте, поскольку сохранение козырей повышает устойчивость стратегии в следующих раундах. Если некозырного варианта

нет, выбирается козырный. При отсутствии обоих вариантов стратегия не применяется и управление передается следующей атакующей стратегии.

```
int notTrumpCardIdx = -1; // индекс в таблице и сама карта для
быстрого удаления
int trumpCardIdx = -1;
int selectedCardIdx = -1; // итоговая карта, которой будем ходить

if (opponentHand.Count <= 6) // стратегия 1. если мы точно знаем карты
соперника
{
    for (int i = 0; i < hand.Count; i++)
    {
        SCard card = hand[i];

        // если соперник не может отбиться от некозырной карты
        if (
            card.Suit != trumpSuit &&
            !OpponentCanBeat(card) &&
            (notTrumpCardIdx == -1 || card.Rank <
hand[notTrumpCardIdx].Rank)
        )
        {
            notTrumpCardIdx = i;
        }
        // если соперник не может отбиться от козырной карты
        else if (
            (trumpCardIdx == -1 ||
            card.Rank < hand[trumpCardIdx].Rank) &&
            !OpponentCanBeat(card)
        )
        {
            trumpCardIdx = i;
        }
    }

    selectedCardIdx = notTrumpCardIdx != -1 ? notTrumpCardIdx :
trumpCardIdx; // лучше походить некозырной

    if (selectedCardIdx != -1)
    {
        SCard selectedCard = hand[selectedCardIdx];
        hand.RemoveAt(selectedCardIdx);
        return new List<SCard> { selectedCard };
    }
}
```

Листинг 1. Атака гарантированно неотбиваемой картой

3.2.2. Стратегия набора карт одного ранга

Данная стратегия ориентирована на создание «пакета» из двух – четырех карт одного ранга и последующее давление на соперника за счет возможности подкидывания карт этого же ранга. На ранней стадии партии козырные карты

исключаются из формирования набора, поскольку их выгоднее сохранить для обороны и поздних разменов.

Алгоритм работает следующим образом: выбирается подмножество карт (с учетом правила сохранения козырей на ранней стадии), затем карты группируются по рангу и сортируются по возрастанию ранга. Из полученных групп выбирается первая подходящая группа, содержащая более одной карты и не превышающая по размеру количество карт на руке соперника (чтобы набор был реализуем с точки зрения лимита подкидывания). После выбора набора бот делает ход одной картой из группы, а оставшиеся карты сохраняет как «заготовку» для подкидывания в течение текущего раунда.

```
private SCard? PlayBatchStrategy(List<SCard>? hand = null)
{
    if (hand == null) hand = this.hand;
    if (hand.Count == 0) return null;

    List<SCard> filteredHand = countBittedCards <= 18 ? // в первой
половине игры придерживаем козыри
                    hand.Where(card => card.Suit !=
trumpSuit).ToList() :
                    hand;
    // ищем пары/тройки/четверки одного ранга
    List<SCard> cardPairGroup = filteredHand
                    .GroupBy(card => card.Rank) // группируем по рангу
                    .OrderBy(cardGroup => cardGroup.Key) // сортируем
по рангу по возрастанию
                    .FirstOrDefault(cardGroup => cardGroup.Count() > 1
&& cardGroup.Count() <= opponentCurrentCountCards) // первая группа,
где количество карт больше одной
                    ?.ToList() ?? new List<SCard>();

    if (cardPairGroup.Count > 1)
    {
        SCard first = cardPairGroup[0];

        cardPairGroup.RemoveAt(0); // ходим по одной
        this.hand.Remove(first);
        batchMemoCards.Clear();

        // запоминаем карты для подкидывания
        for (int i = 0; i < cardPairGroup.Count; i++)
            batchMemoCards.Append(new KeyValuePair<int, SCard>(i,
cardPairGroup[i]));

        return first;
    }

    return null;
}
```

Листинг 2. Стратегия набора карт одного ранга

3.3. Подкидывание карт

Алгоритм подкидывания является пошаговым и на каждом вызове добавляет не более одной карты, что соответствует общей концепции «ходить только по одной».

Сначала проверяются базовые ограничения: подкидывание невозможно, если у бота отсутствуют карты на руке, либо если на стол уже выложено максимально допустимое число карт (ограничение задаётся минимальным значением между количеством карт соперника и лимитом в шесть карт). Далее действия выполняются по приоритетам.

Во-первых, если на предыдущем ходе был сформирован набор карт одного ранга (пара/тройка/четвёрка), то бот продолжает его разыгрывание: извлекается одна карта из сохранённого набора и подкидывается на стол. Это сохраняет правило «по одной карте» и позволяет реализовать заранее подготовленное давление.

Во-вторых, если сохранённого набора нет, бот пытается сформировать новый набор, но уже с учётом текущего стола. Для этого отбираются только те карты на руке, ранги которых представлены на столе (включая ранги карт в побитых парах). Если среди этих карт удаётся сформировать набор одного ранга, бот подкидывает одну карту из найденного набора, а оставшиеся карты рассматриваются как заготовка для последующих подкидываний.

В-третьих, если набор сформировать нельзя, применяется стратегия выбора минимальной подходящей карты. Рассмотрим данную стратегию более подробно:

- Бот перебирает все карты на руке и рассматривает только те, которые разрешено подкинуть: их ранг уже присутствует на столе. На ранней стадии игры козырные карты дополнительно исключаются из рассмотрения.
- Если рука соперника фактически известна (конец партии), бот стремится выбрать минимальную по рангу карту, которую соперник не сможет побить. При этом, если ранее была выбрана карта, которая

допускает успешную защиту соперника, то более предпочтительной становится карта меньшего ранга, ухудшающая положение соперника.

- Если рука соперника неизвестна, либо подходящей карты, которую соперник не побил бы, нет, бот выбирает просто минимальную по рангу карту среди допустимых для подкидывания. Это снижает стоимость подкидывания и позволяет сохранять сильные карты на будущие размены.

```
// ищем минимальную подходящую карту
int selectedCardIdx = -1;
bool selectedCanBeat = true; // могут ли побить выбранную карту

for (int i = 0; i < hand.Count; i++)
{
    if (hand[i].Suit == trumpSuit && countBittedCards <= 18) continue;
// в первой половине игры придерживаем козыри

    if (ranksOnTable.Contains(hand[i].Rank)) // если можем подкинуть
    {
        if (
            opponentHand.Count <= 6 && // если знаем точные карты
оппонента и можем отбиться
            OpponentCanBeat(hand[i]) &&
            // если карты нет, карту могут отбить или есть карта
меньшего ранга, которую нельзя отбить
            (selectedCardIdx == -1 || selectedCanBeat || hand[i].Rank
< hand[selectedCardIdx].Rank)
        )
        {
            selectedCardIdx = i;
            selectedCanBeat = false;
        }
        // если выбранную можно отбить или есть карта меньшего ранга
        else if (selectedCanBeat && (selectedCardIdx == -1 ||
hand[i].Rank < hand[selectedCardIdx].Rank))
        {
            selectedCardIdx = i;
            selectedCanBeat = true;
        }
    }
}
```

Листинг 3. Стратегия подкидывания наименьшей карты

3.4. Защита бота

Алгоритм защиты похож на алгоритм поиска карты наименьшего ранга. Приведем код данного алгоритма.

```
// защита от карт
// на вход подается набор карт на столе, часть из них могут быть уже
покрыты
public bool Defend(List<SCardPair> table)
{
    roundAttacker = RoundAttacker.OPPONENT;
    if (hand.Count == 0) return false;

    for (int i = 0; i < table.Count; i++)
    {
        if (table[i].Beaten) continue; // если карта уже побита

        // выбираем минимальную карту, чтобы побиться
        int selectedIdx = -1;

        for (int j = 0; j < hand.Count; j++)
        {
            if (!SCard.CanBeat(table[i].Down, hand[j], trumpSuit))
continue; // если не можем побить

            bool currentIsTrump = hand[j].Suit == trumpSuit;
            bool cardIsTrump = selectedIdx != -1 &&
hand[selectedIdx].Suit == trumpSuit;

            if (currentIsTrump && countBittedCards <= 12) continue; //
в первой трети игры придерживаем козыри

            // предпочитаем некозырные карты козырям
            if (
                selectedIdx == -1 ||
                // если карта не козырная, но "лучшая" - козырная
                (!currentIsTrump && cardIsTrump) ||
                // если выбираем среди некозырных, но эта карта ниже
рангом
                (!currentIsTrump && !cardIsTrump && hand[j].Rank <
hand[selectedIdx].Rank) ||
                // если выбираем среди козырных, но эта карта ниже
рангом
                (currentIsTrump && cardIsTrump && hand[j].Rank <
hand[selectedIdx].Rank))
            {
                selectedIdx = j;
            }
        }

        // если хоть одну карту не можем побить, забираем карты
соперника
        if (selectedIdx == -1) return false;

        SCardPair pair = table[i];
        pair.SetUp(hand[selectedIdx], trumpSuit); // бьемся
        table[i] = pair;

        hand.RemoveAt(selectedIdx);
    }

    return true;
}
```

Листинг 4. Защита бота

ГЛАВА 4. ТЕСТИРОВАНИЕ

4.1. Описание модуля MPlayer2

Для оценки качества разработанного алгоритма и проверки выигрышности стратегии модуля MPlayer1 был реализован соперник MPlayer2, представляющий собой облегчённую версию первого модуля. Назначение MPlayer2 – выступать в роли стабильного «бенчмарка», который играет корректно, но без интеллектуальных надстроек, за счёт чего результаты серии игр позволяют объективнее оценить вклад усложнённых эвристик MPlayer1.

По сравнению с MPlayer1 из MPlayer2 были исключены следующие элементы, влияющие на поведение игрока:

1. **Моделирование состояния оппонента.** В MPlayer1 ведётся учёт возможной руки соперника (множество потенциальных карт), а также выполняются проверки вида «может ли соперник отбиться» от конкретной карты. В MPlayer2 эти механизмы.
2. **Учёт параметров партии и динамики добора.** MPlayer1 отслеживает оставшиеся карты в колоде и текущее количество карт у оппонента, корректируя ограничения на подкидывание и интерпретацию ситуации в раунде. В MPlayer2 нет подсистемы, которая бы учитывала оставшуюся колоду, порядок добора и точное количество карт у противника (кроме грубой проверки по числу карт в игре).
3. **Расширенные атакующие эвристики.** Из MPlayer1 исключена стратегия формирования наборов карт одного ранга (пары/тройки/четвёрки) с запоминанием очереди для последовательного подкидывания. В MPlayer2 отсутствует планирование серий подкидываний.
4. **Более аккуратная защита и управление козырями.** MPlayer1 в защите учитывает стадию игры и старается экономить козыри на ранних

этапах. MPlayer2 защищается проще: выбирает минимально возможную карту для отбоя без дополнительных ограничений по сохранению козырей.

Иными словами, MPlayer2 реализует корректные действия по правилам, но использует минимальный набор эвристик, что делает его удобным контрольным противником для сравнительного тестирования.

Поведение MPlayer2 можно описать следующими правилами:

- **Атака (первый ход в раунде).** Выбирается минимальная карта среди некозырных; если некозырных карт нет – минимальный козырь. Это обеспечивает простую тактику с минимальными вычислениями.
- **Защита.** Для каждой непобитой атакующей карты подбирается минимальная по рангу карта, которая способна побить её (с учётом козырной масти). Если хотя бы одну карту побить невозможно – защита считается проваленной.
- **Подкидывание.** Подкидывание выполняется только после успешной защиты соперника, при этом выбирается минимальная карта из руки, ранг которой уже присутствует на столе. Дополнительно применяется ограничение на максимальное число пар на столе (не более 6) и выполняется проверка, что игра не находится в состоянии окончания колоды (чтобы не провоцировать некорректные сценарии).
- **Учёт сыгранных карт.** После успешной защиты увеличивается счётчик карт, ушедших в «бито», на фактическое количество карт на столе (включая побитые).

4.2. Результаты тестового соревнования

Была выполнена серия из 10000 тестовых игр. Получены следующие итоги:

- Побед игрока 1 (MPlayer1): 6072
- Побед игрока 2 (MPlayer2): 3742
- Ничья: 186
- Ошибок выполнения: 0

№	Имена	Победы	Очки	Ошибки		Загрузить
0	CardFool vs CardFool	6072 vs 3742	26660 vs 37778	0	Больше	Сохранить

Рисунок 6. Результаты 10000 игр с MPlayer2

4.3. Выводы

Проведённое тестирование показывает, что разработанная стратегия MPlayer1 статистически превосходит упрощённого соперника MPlayer2: модуль чаще доводит партию до победы за счёт более содержательных правил выбора карт и более «осмысленного» поведения в атаке и подкидывании, а также за счёт управляемого использования ресурсов (козырей и оценок ситуации). Полученные результаты подтверждают, что усложнение эвристик по сравнению с базовым жадным алгоритмом даёт практический выигрыш в эффективности игры при сохранении корректности и устойчивости исполнения.

СПИСОК ИСТОЧНИКОВ

1. Metanit.com. C# и платформа .NET [Электронный ресурс]. – Режим доступа: https://metanit.com/sharp/ дата обращения: 18.02.2026).
2. Подкидной дурак // Википедия : свободная энциклопедия [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Подкидной_дурак (дата обращения: 18.02.2026).

ПРИЛОЖЕНИЕ 1

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;

namespace CardFool
{
    public class MPlayer1
    {
        private enum RoundAttacker { ME, OPPONENT }

        private List<SCard> hand = new List<SCard>(); // карты на руке
        private List<SCard> opponentHand = new List<SCard>(); //
предполагаемые карты на руке оппонента
        private RoundAttacker roundAttacker; // кто атакует в текущем
раунде
        private Suits trumpSuit; // козырная масть
        // запоминаем карты для стратегии двойки/тройки/четверки
        private Queue<KeyValuePair<int, SCard>> batchMemoCards = new
Queue<KeyValuePair<int, SCard>>();
        private int countBittedCards = 0; // количество карт в бито
        private int deckRemaining = 36 - 12;  // сколько карт осталось
в колоде (включая козырь)
        private int opponentCurrentCountCards = 6; // количество карт
оппонента


        // инициализация потенциальных карт в руке оппонента.
изначально оппонент может иметь любые карты
        private void InitOpponentHand()
        {
            for (int rank = 6; rank < 15; rank++)
            {
                opponentHand.AddRange(new List<SCard> {
                        new SCard(Suits.Clubs, rank),
                        new SCard(Suits.Diamonds, rank),
                        new SCard(Suits.Hearts, rank),
                        new SCard(Suits.Spades, rank)
                    });
            }
        }

        // получаем ранги карт на столе
        private List<int> GetTableRanks(List<SCardPair> table)
        {
            List<int> ranks = new List<int>(); // оптимальнее
использовать HashSet

            table.ForEach(pair =>
            {
                if (!ranks.Contains(pair.Down.Rank))
ranks.Add(pair.Down.Rank);
                if (pair.Beaten && !ranks.Contains(pair.Up.Rank))
ranks.Add(pair.Up.Rank);
            });

            return ranks;
        }

        // проверка "может ли оппонент отбиться?"
```

```
        private bool OpponentCanBeat(SCard myCard)
        {
            return opponentHand.Exists(card => SCard.CanBeat(myCard,
card, trumpSuit));
        }

        // играем стратегию двоек/троек/четверок
        private SCard? PlayBatchStrategy(List<SCard>? hand = null)
        {
            if (hand == null) hand = this.hand;
            if (hand.Count == 0) return null;

            List<SCard> filteredHand = countBittedCards <= 18 ? // в
первой половине игры придерживаем козыри
                            hand.Where(card => card.Suit !=
trumpSuit).ToList() :
                            hand;
            // ищем пары/тройки/четверки одного ранга
            List<SCard> cardPairGroup = filteredHand
                            .GroupBy(card => card.Rank) // группируем
по рангу
                            .OrderBy(cardGroup => cardGroup.Key) //
сортируем по рангу по возрастанию
                            .FirstOrDefault(cardGroup =>
cardGroup.Count() > 1 && cardGroup.Count() <=
opponentCurrentCountCards) // первая группа, где количество карт
больше одной
                            ?.ToList() ?? new List<SCard>();

            if (cardPairGroup.Count > 1)
            {
                SCard first = cardPairGroup[0];

                cardPairGroup.RemoveAt(0); // ходим по одной
                this.hand.Remove(first);
                batchMemoCards.Clear();

                // запоминаем карты для подкидывания
                for (int i = 0; i < cardPairGroup.Count; i++)
                    batchMemoCards.Append(new KeyValuePair<int,
SCard>(i, cardPairGroup[i]));

                return first;
            }

            return null;
        }

        // возвращает имя игрока
        public string GetName() => "Mikhail";

        // возвращает количество карт на руке
        public int GetCount() => hand.Count;

        // добавление карты в руку, во время добора из колоды, или
взятия карт
        public void AddToHand(SCard card)
        {
            hand.Add(card);
            opponentHand.Remove(card);
        }

        // начальная атака
        public List<SCard> LayCards()
        {
```

```
            roundAttacker = RoundAttacker.ME;

            if (hand.Count == 0) return new List<SCard>();

            int notTrumpCardIdx = -1; // индекс в таблице и сама карта
для быстрого удаления
            int trumpCardIdx = -1;
            int selectedCardIdx = -1; // итоговая карта, которой будем
ходить

            if (opponentHand.Count <= 6) // стратегия 1. если мы точно
знаем карты соперника
            {
                for (int i = 0; i < hand.Count; i++)
                {
                    SCard card = hand[i];

                    // если соперник не может отбиться от некозырной
карты
                    if (
                        card.Suit != trumpSuit &&
                        !OpponentCanBeat(card) &&
                        (notTrumpCardIdx == -1 || card.Rank <
hand[notTrumpCardIdx].Rank)
                    )
                    {
                        notTrumpCardIdx = i;
                    }
                    // если соперник не может отбиться от козырной
карты
                    else if (
                        (trumpCardIdx == -1 ||
                        card.Rank < hand[trumpCardIdx].Rank) &&
                        !OpponentCanBeat(card)
                    )
                    {
                        trumpCardIdx = i;
                    }
                }

                selectedCardIdx = notTrumpCardIdx != -1 ?
notTrumpCardIdx : trumpCardIdx; // лучше походить некозырной

                if (selectedCardIdx != -1)
                {
                    SCard selectedCard = hand[selectedCardIdx];
                    hand.RemoveAt(selectedCardIdx);
                    return new List<SCard> { selectedCard };
                }
            }

            // стратегия 2. играем стратегию пар/троек/четверок
            SCard? batchFirst = PlayBatchStrategy();
            if (batchFirst.HasValue)
                return new List<SCard> { batchFirst.Value };

            notTrumpCardIdx = -1;
            trumpCardIdx = -1;

            // 3. минимальная карта
            // выбираем минимальную некозырную карту и минимальную
козырь для хода
            for (int i = 0; i < hand.Count; i++)
            {
                SCard card = hand[i];
```

```
                if (
                    card.Suit != trumpSuit &&
                    (notTrumpCardIdx == -1 || card.Rank <
hand[notTrumpCardIdx].Rank)
                )
                    notTrumpCardIdx = i;
                else if (trumpCardIdx == -1 || card.Rank <
hand[trumpCardIdx].Rank)
                    trumpCardIdx = i;
            }

            selectedCardIdx = notTrumpCardIdx != -1 ? notTrumpCardIdx
: trumpCardIdx; // лучше походить некозырной

            if (selectedCardIdx != -1)
            {
                SCard selectedCard = hand[selectedCardIdx];
                hand.RemoveAt(selectedCardIdx);
                return new List<SCard> { selectedCard };
            }

            return new List<SCard>();
        }

        // защита от карт
        // на вход подается набор карт на столе, часть из них могут
быть уже покрыты
        public bool Defend(List<SCardPair> table)
        {
            roundAttacker = RoundAttacker.OPPONENT;
            if (hand.Count == 0) return false;

            for (int i = 0; i < table.Count; i++)
            {
                if (table[i].Beaten) continue; // если карта уже
побита

                // выбираем минимальную карту, чтобы побиться
                int selectedIdx = -1;

                for (int j = 0; j < hand.Count; j++)
                {
                    if (!SCard.CanBeat(table[i].Down, hand[j],
trumpSuit)) continue; // если не можем побить

                    bool currentIsTrump = hand[j].Suit == trumpSuit;
                    bool cardIsTrump = selectedIdx != -1 &&
hand[selectedIdx].Suit == trumpSuit;

                    if (currentIsTrump && countBittedCards <= 12)
continue; // в первой трети игры придерживаем козыри

                    // предпочитаем некозырные карты козырям
                    if (
                        selectedIdx == -1 ||
                        // если карта не козырная, но "лучшая" -
козырная
                        (!currentIsTrump && cardIsTrump) ||
                        // если выбираем среди некозырных, но эта
карта ниже рангом
                        (!currentIsTrump && !cardIsTrump &&
hand[j].Rank < hand[selectedIdx].Rank) ||
                        // если выбираем среди козырных, но эта карта
ниже рангом
```

```
                        (currentIsTrump && cardIsTrump && hand[j].Rank
< hand[selectedIdx].Rank))
                    {
                        selectedIdx = j;
                    }
                }

                // если хоть одну карту не можем побить, забираем
карты соперника
                if (selectedIdx == -1) return false;

                SCardPair pair = table[i];
                pair.SetUp(hand[selectedIdx], trumpSuit); // бьемся
                table[i] = pair;

                hand.RemoveAt(selectedIdx);
            }

            return true;
        }

        // добавление карт
        // на вход подается набор карт на столе, а также отбился ли
оппонент
        public bool AddCards(List<SCardPair> table, bool
opponentDefensed)
        {
            if (hand.Count == 0) return false; // нет карт в руке
            if (table.Count >= Math.Min(opponentCurrentCountCards, 6))
// не можем подкидывать больше, чем есть у оппонента
                return false;

            if (batchMemoCards.Count > 0) // если активна стратегия
пар/двоек/троек
            {
                var cardPair = batchMemoCards.Dequeue(); // снова
ходим по одной, берем первую карту в очереди

                hand.RemoveAt(cardPair.Key);
                table.Add(new SCardPair(cardPair.Value));

                return true;
            }

            // формируем новый наборы карт. выбираем карты для
подкидывания
            List<SCard> batchHand = hand.Where(card =>
            {
                return table.Any(tableCard =>
                {
                    if (tableCard.Beaten)
                        return card.Rank == tableCard.Down.Rank ||
card.Rank == tableCard.Up.Rank;
                    return card.Rank == tableCard.Down.Rank;
                });
            }).ToList();
            SCard? batchFirst = PlayBatchStrategy(batchHand); //
пытаемся подкинуть пары/двоики/троики
            if (batchFirst.HasValue)
            {
                table.Add(new SCardPair(batchFirst.Value));
                return true;
            }

            List<int> ranksOnTable = GetTableRanks(table);
```

```
            // ищем минимальную подходящую карту
            int selectedCardIdx = -1;
            bool selectedCanBeat = true; // могут ли побить выбранную
карту

            for (int i = 0; i < hand.Count; i++)
            {
                if (hand[i].Suit == trumpSuit && countBittedCards <=
18) continue; // в первой половине игры придерживаем козыри

                if (ranksOnTable.Contains(hand[i].Rank)) // если можем
подкинуть
                {
                    if (
                        opponentHand.Count <= 6 && // если знаем
точные карты оппонента и можем отбиться
                        OpponentCanBeat(hand[i]) &&
                        // если карты нет, карту могут отбить или есть
карта меньшего ранга, которую нельзя отбить
                        (selectedCardIdx == -1 || selectedCanBeat ||
hand[i].Rank < hand[selectedCardIdx].Rank)
                    )
                    {
                        selectedCardIdx = i;
                        selectedCanBeat = false;
                    }
                    // если выбранную можно отбить или есть карта
меньшего ранга
                    else if (selectedCanBeat && (selectedCardIdx == -1
|| hand[i].Rank < hand[selectedCardIdx].Rank))
                    {
                        selectedCardIdx = i;
                        selectedCanBeat = true;
                    }
                }
            }

            if (selectedCardIdx == -1) return false;

            table.Add(new SCardPair(hand[selectedCardIdx]));
            hand.RemoveAt(selectedCardIdx);

            return true;
        }

        public void OnEndRound(List<SCardPair> table, bool
isDefenseSuccessful)
        {
            int attackCards = table.Count; // сколько атакующих карт
лежит снизу

            if (isDefenseSuccessful) countBittedCards += attackCards *
2; // если защита успешна, увеличиваем бито

            if (roundAttacker == RoundAttacker.ME)
            {
                // если соперник успешно защитился
                if (isDefenseSuccessful) opponentCurrentCountCards -=
attackCards;
                else opponentCurrentCountCards += attackCards; //
соперник взял
            }
            else opponentCurrentCountCards -= attackCards; // если
соперник атаковал
```

```
            // удаляем карты, которые вышли из игры, из возможной руки
соперника
            foreach (var pair in table)
            {
                opponentHand.Remove(pair.Down);
                if (pair.Beaten) opponentHand.Remove(pair.Up);
            }

            // сколько карт нужно добрать мне и сопернику
            int myNeed = Math.Max(0, 6 - hand.Count);
            int opponentNeed = Math.Max(0, 6 -
opponentCurrentCountCards);

            int opponentDraw = 0; // сколько доберет оппонент

            if (deckRemaining > 0)
            {
                if (roundAttacker == RoundAttacker.ME)
                {
                    // если атаковали первыми, то добираем первым
                    // добираем сколько необходимо или сколько
осталось
                    deckRemaining -= Math.Min(myNeed, deckRemaining);

                    opponentDraw = Math.Min(opponentNeed,
deckRemaining);
                    deckRemaining -= opponentDraw; // аналогично
добирает соперник
                }
                else
                {
                    opponentDraw = Math.Min(opponentNeed,
deckRemaining);
                    deckRemaining -= opponentDraw; // первый добирает
соперник
                    deckRemaining -= Math.Min(myNeed, deckRemaining);
// потом добираю я
                }
            }

            opponentCurrentCountCards += opponentDraw;

            // сбрасываем память для стратегии двоек/троек/четверок
            batchMemoCards.Clear();
        }

        // установка козыря, на вход подаётся козырь, вызывается перед
первой раздачей карт
        public void SetTrump(SCard newTrump)
        {
            trumpSuit = newTrump.Suit;
            InitOpponentHand();
            opponentHand.Remove(newTrump);
        }
    }
}
```