

Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «Пензенский государственный университет»

Факультет вычислительной техники

Кафедра «Вычислительная техника»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

по курсу «Программирование»

на тему: «Разработка программы сложной структуры методом нисходящего
программирования. Консольная игра «Двадцать одно»»

Выполнил: ст. гр. 25BBB1

Садыков Д.Р.

Проверил: к.т.н. доцент

Токарев А.Н.

071

20.05.26



Пенза, 2026

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет Вычислительной техники
Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ

_____ М.А. Митрохин

« ____ » _____ 20 __ г

ЗАДАНИЕ

на курсовое проектирование по курсу

Программирование

Студенту Сапожкову Даниилу Рустамовичу Группа 25ВВВ1

Тема проекта: *Разработка программы сложной структуры методом нисходящего программирования.*

Исходные данные (технические требования) на проектирование
Игра "Двадцать одно". Игрок и компьютер поочередно открывают карты. Задача игрока – набрать больше очков, чем у компьютера, но не более чем 21. Статистика побед и поражений хранится в файлах.

Обязательные требования к программе:

1. Многомодульность.
2. Использование сложных типов данных – структур, списков, файлов.
3. Режим работы видеосистемы – текстовый / графический.
4. Устройство ввода информации – клавиатура, мышь.
5. Пользовательский интерфейс должен быть построен на основе меню и панели инструментов.
6. Наличие заставки.
7. Операционная система MS Windows.
8. Язык программирования – Си и Ассемблер.
9. Среда разработки ПО – Microsoft Visual Studio.

Объем работы по курсу

1. Расчетная часть

Разработка программы.

2. Графическая часть

Схема данных, схема ресурсов системы, схема работы системы, иерархическая структура программы, схема взаимодействия программы.

3. Экспериментальная часть

Отладка программы.

Срок выполнения проекта по разделам

В соответствии с графиком.

Дата выдачи задания “ 4 ” 03.2026

Дата защиты проекта “ ”

Руководитель Аюкариб

Задание получил “ 4 ” 03. 2026г.

Студент Сагдиев.Д.Р.

Содержание

Введение	5
1. Постановка задачи	6
2. Выбор решения	7
3. Описание разработки программы	7
4. Отладка и тестирование	10
5. Описание программы	10
5.1 Описание модулей	10
5.2 Разработка функции на языке Assembler	13
5.3 Графическая часть (Схемы)	15
6. Руководство пользователя	18
6.1 Заставка и Меню	19
6.2 Игровое поле и процесс игры	20
6.3 Победа и поражение	21
Заключение	22
Список используемых материалов	23
Приложение А. Листинг программы	24
Файл main.c	24
Файл value.c	27
Файл game.c	29
Файл display.c	32
Файл deck.c	39
Файл stats.c	40

Введение

Цифровые развлечения прочно вошли в повседневную жизнь современного человека, охватывая не только сферу досуга, но и образование. Среди множества логических и карточных игр особое место занимает «Блэкджек» (в локальной адаптации — «Двадцать одно») — игра, требующая понимания теории вероятностей и математической интуиции.

Несмотря на минималистичный текстовый интерфейс, за ней скрывается строгая алгоритмическая основа, изучение которой даёт ценный практический опыт в разработке программного обеспечения на языках низкого и среднего уровня. Отличительной особенностью данного проекта является намеренный отказ от механик ставок и азартной составляющей. Это архитектурное решение позволило сфокусироваться исключительно на математической вероятности, подсчете очков и корректной работе конечного автомата состояний игры.

Актуальность данной работы определяется тем, что реализация подобного приложения охватывает широкий спектр фундаментальных задач программирования: работу с указателями и массивами, проектирование игровой логики, обработку пользовательского ввода, работу с файловой системой и построение псевдографического интерфейса в консоли. Всё это в совокупности формирует системный подход к разработке — от архитектуры отдельных модулей до взаимодействия компонентов программы.

В качестве основного языка разработки выбран Си, поскольку он обеспечивает прямой доступ к памяти и высокую скорость выполнения. Для реализации низкоуровневой оптимизации наиболее ресурсоемкой операции (динамического подсчета очков с учетом плавающего номинала туза) в работе применяется ассемблерная вставка.

1. Постановка задачи

Целью данной работы является разработка консольной версии карточной игры «Двадцать одно» с использованием языка программирования Си, а также ассемблерной вставки для низкоуровневой оптимизации ключевых математических операций.

В ходе выполнения работы необходимо реализовать следующий функционал:

1. Генерация и перемешивание колоды из 52 карт (с поддержкой нескольких колод).
2. Управление процессом добора карт (Hit) и остановки (Stand) при помощи клавиатуры.
3. Автоматический подсчет очков с учетом того, что Туз может давать 1 или 11 очков в зависимости от текущей суммы.
4. Алгоритм действий дилера (добор карт при счете менее 17).
5. Проверка условий завершения раунда — победа при переборе у дилера, большем количестве очков или блэкджеке, поражение при собственном переборе.
6. Сохранение результатов и статистики партий в текстовый файл с отображением таблицы рекордов.

В процессе реализации проекта должны быть использованы сложные типы данных — структуры, многомерные массивы, файлы, а также принципы многомодульного программирования. Ассемблерная вставка должна быть выполнена в стиле MASM.

Режим работы видеосистемы — текстовый, с использованием псевдографики и цветового оформления WinAPI.

2. Выбор решения

В процессе разработки данной игры была выбрана среда MS Visual Studio. Она предоставляет все необходимые инструменты для комфортной разработки консольных приложений на языке Си, а также поддерживает синтаксис ассемблерных вставок `__asm {}` для архитектуры x86.

Для создания графического интерфейса в рамках консольного окна применяются функции операционной системы Windows (WinAPI). Настройка кодировки UTF-8 (65001) и флагов виртуального терминала обеспечивает корректное отображение псевдографических символов мастей (♠, ♣, ♥, ♦) и рамок карт. Для выделения мастей используется функция `SetConsoleTextAttribute`, окрашивающая их в соответствующие цвета.

Стоит отметить реализацию финального экрана (благодарности за игру). В отличие от основного игрового цикла, где применяется точечное позиционирование курсора, экран выхода выводится с использованием упрощенного последовательного вывода через стандартный `printf`. Это избавляет от артефактов при завершении работы программы и возврате управления системной консоли.

Проект структурирован с использованием модулей, что позволяет логически разделить код на функциональные блоки: игровая механика, работа с колодой, интерфейс, подсчет очков и статистика. Результаты сохраняются в текстовый файл `stats.txt`.

3. Описание разработки программы

В основу разработки положен нисходящий подход — сначала была определена общая архитектура программы и структуры данных (такие как `GameState` и `Deck`), после чего каждый компонент последовательно детализировался.

Запуск программы сопровождается отображением экрана приветствия в виде ASCII-арта. После перехода в главное меню игрок может ввести свое имя, посмотреть таблицу рекордов или начать игру. Управление в игре осуществляется горячими клавишами: 'H' (Взять карту) и 'S' (Хватит).

Код логически разбит на 6 независимых C-модулей с соответствующими заголовочными файлами — такая структура даёт возможность дорабатывать любую часть программы, не затрагивая остальные:

1. *main.c*;
2. *game.c*;
3. *deck.c*;
4. *stats.c*;
5. *value.c*;
6. *display.c*.

Модуль *main.c* является точкой входа — здесь выполняется первичная настройка консоли (кодировка UTF-8, заголовок окна, размеры), инициализируется генератор случайных чисел и запускается основной цикл приложения с обработкой выбора пользователя в меню.

Модуль *game.c* содержит всю игровую механику: функции раздачи начальных карт, логику хода «Hit» (взять карту) для игрока и дилера, проверку на моментальный «Блэкджек» и финальное определение победителя раунда.

Модуль *deck.c* отвечает за управление колодой: инициализацию массива карт (масти и номиналы), их случайное перемешивание по алгоритму Фишера-Йетса и выдачу карт с автоматическим перетасовыванием при исчерпании колоды.

Модуль *display.c* берёт на себя всю визуальную часть: вывод ASCII-заставки, отрисовку главного меню, таблицы рекордов и формирование игрового стола с цветной индикацией мастей и отрисовкой карт псевдографикой.

Модуль *stats.c* отвечает за работу с файлом статистики *stats.txt*: загрузку существующих данных о победах и поражениях игроков и сохранение новых результатов по завершении каждой партии.

Модуль *value.c* реализует функции подсчёта суммы очков в руке. Ключевая операция выполнена в виде ассемблерной вставки (`__asm`), которая на уровне регистров процессора эффективно вычисляет общую сумму с учётом переменного номинала Туза (1 или 11 очков).

4. Отладка и тестирование

Отладка программы производилась в среде MS Visual Studio. В процессе разработки применялись методы пошагового выполнения кода, контроль значений переменных через окно Watch, а также анализ дампов памяти при работе с массивами карт.

Наибольшую сложность представляла отладка ассемблерного блока в функции `calculateHandValue_Asm`. Первоначальная проблема заключалась в некорректном подсчете очков при наличии нескольких тузов. Если у игрока было два туза, алгоритм мог ошибочно присвоить обоим значение 11, что приводило к моментальному перебору. Для устранения ошибки был введен цикл постобработки `ase_adjust_loop`, который последовательно отнимает по 10 очков за каждого туза, пока сумма не станет меньше или равна 21. Корректность работы регистров `eax` (сумма) и `ebx` (счетчик тузов) проверялась через окно дизассемблера.

При тестировании файловой системы была выявлена ошибка доступа к памяти, если программа запускалась впервые (файл `stats.txt` отсутствовал). Проблема решена добавлением проверки указателя файла на `NULL`, позволяя программе штатно создать файл при первом сохранении.

Проводилось стресс-тестирование модуля колоды. Было подтверждено, что условие порога перемешивания `RESHUFF_THRESHOLD` корректно срабатывает, предотвращая выход за пределы массива при длительных сессиях.

5. Описание программы

5.1 Описание модулей

Программа состоит из 6 основных модулей, которые упрощают процесс разработки и отладки, улучшают читаемость и обеспечивают логическое разделение интерфейса, игровой механики и низкоуровневых вычислений.

Модуль *main.c* является центральным файлом программы. Он координирует работу всех остальных компонентов и управляет жизненным циклом приложения.

setupConsole() — выполняет предварительную настройку окна консоли:

устанавливает размер, заголовок и включает поддержку UTF-8 и ANSI последовательностей.

playRound() — управляет логикой одного игрового раунда: от начальной раздачи до определения победителя и обновления статистики.

waitForInput() — организует ожидание корректного ввода от пользователя (клавиши управления игрой или навигации в меню).

main() — точка входа в программу; инициализирует ресурсы и запускает основной цикл работы с главным меню.

Модуль *game.c* содержит реализацию правил игры «Двадцать одно» и управляет состоянием текущей партии.

dealInitialCards() — инициализирует раздачу: очищает руки игроков и

выдает по две начальные карты игроку и дилеру. *playerHit()* — реализует взятие дополнительной карты игроком с проверкой на возможный «перебор» (BUST).

dealerHit() — добавляет одну карту в руку дилера.

checkInitialBlackjack() — проверяет наличие выигрышной комбинации

«21» сразу после раздачи.

determineWinner() — сравнивает финальные очки игрока и дилера, определяя результат раунда и формируя итоговое сообщение.

Модуль *deck.c* отвечает за формирование колоды и работу с картами. *initDeck()* — создает колоду из заданного количества наборов по 52

карты, заполняя их номиналами и мастями.

shuffleDeck() — перемешивает карты в колоде случайным образом, используя алгоритм Фишера-Йетса.

drawCardFromDeck() — выдает верхнюю карту из колоды и при необходимости инициирует автоматическую перетасовку.

Модуль *display.c* берет на себя все функции визуализации интерфейса в текстовом режиме. *showSplash()* — отображает графическую заставку с названием игры

при запуске. *drawMainMenu()* — отрисовывает главное меню с вариантами выбора

действий.

drawGameScreen() — основная функция отрисовки стола: выводит текущую статистику, сообщение о результате и графическое представление карт в руках. *drawRecordsScreen()* — формирует и выводит на экран таблицу

рекордов из файла. *showNameScreen()* — отрисовывает окно для ввода имени игрока перед началом партии.

Модуль *stats.c* обеспечивает сохранение прогресса и работу с файловой системой.

loadPlayerStats() — загружает персональную статистику игрока по его

имени из файла *stats.txt*.

savePlayerStats() — сохраняет актуальные данные игрока (победы, поражения, ничьи) обратно в файл. *updateStats()* — инкрементирует счетчики статистики в зависимости от результата завершенного раунда.

Модуль *value.c* содержит математический аппарат для расчета очков комбинаций.

calculateHandValue_Asm() — реализует эффективный подсчет суммы очков в руке с помощью ассемблерной вставки, учитывая динамическое значение Туза (1 или 11).

calculateHandValue_C() — альтернативная реализация расчета очков на языке Си для верификации вычислений.

5.2 Разработка функции на языке **Assembler**

Функции, реализованные на языке ассемблера, выполняются быстрее и работают напрямую с регистрами процессора. В качестве разрабатываемой функции была выбрана операция динамического подсчёта суммы очков в руке — *calculateHandValue_Asm()*. Описание функции выполнено на языке C, а вычисления представляют собой ассемблерную вставку в стиле MASM.

Ассемблерная вставка находится в функции *calculateHandValue_Asm()* файла *value.c*. Она последовательно проходит по массиву карт, идентифицирует их номинал (учитывая, что карты с 11 по 13 дают 10 очков, а Туз имеет значение 14), и суммирует результат в регистрах. После основного цикла предусмотрен блок корректировки, который

уменьшает общую сумму, если у игрока есть Тузы и при этом зафиксирован «перебор»

(сумма более 21 очка):

```
__asm{
pushebx
push esi

    mov esi, hand
    mov ecx, count
    mov eax, 0
    mov ebx, 0

    cmpecx, 0
    je asm_done_summing

asm_loop: cmpecx,
0
jleasm_done_summing

    mov  edx, [esi]
    mov dl, dl

    cmp  dl, 14
    je is_ace

    cmp dl, 11
    jlnot_face_cardcmp
    dl, 13
    jgnot_face_card

    add eax, 10 jmpnext_card

not_face_card:
movzxedx, dl
addeax, edx
jmpnext_card

is_ace:  add
    eax, 11
```

```

incedbxjmpnex
t_card

next_card:
add esi, 4
    dec ecx
jmpasm_loop

asm_done_summing:
ace_adjust_loop:
cmpeax, 21
jleasm_finishcmpebx
, 0
jleasm_finish

    sub    eax,    10
dec ebx
jmpace_adjust_loop

asm_finish:
    mov result, eax

    pop esipopebx
}

```

5.3 Графическая часть (Схемы)

На рисунке 1 представлена схема данных.

На рисунке 2 представлена схема ресурсов системы — аппаратные и программные ресурсы.

На рисунке 3 представлена схема работы системы — игровой цикл от запуска до завершения.

На рисунке 4 представлена иерархическая структура программы.

На рисунке 5 представлена схема взаимодействия программы — как модули обмениваются данными между собой.

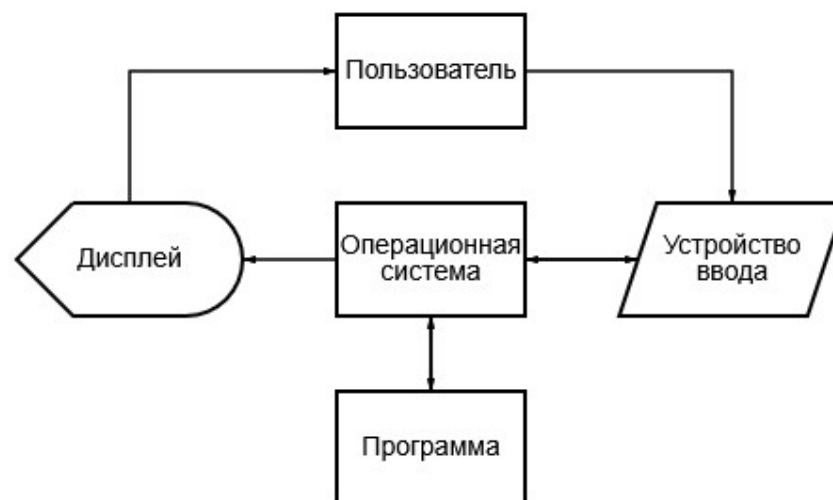


Рисунок 1 – Схема данных



Рисунок 2 – Схема ресурсов системы

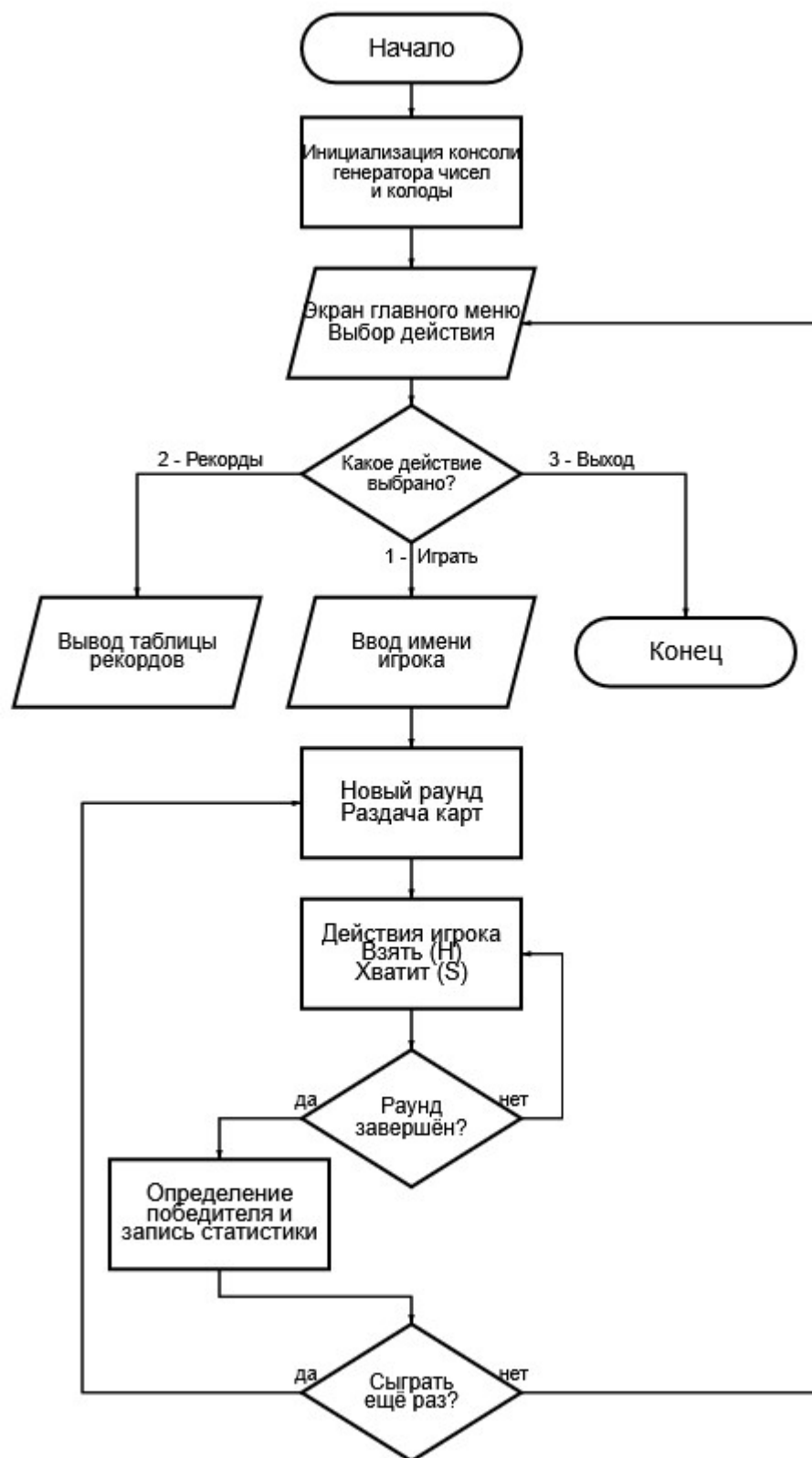


Рисунок 3 – Схема работы системы

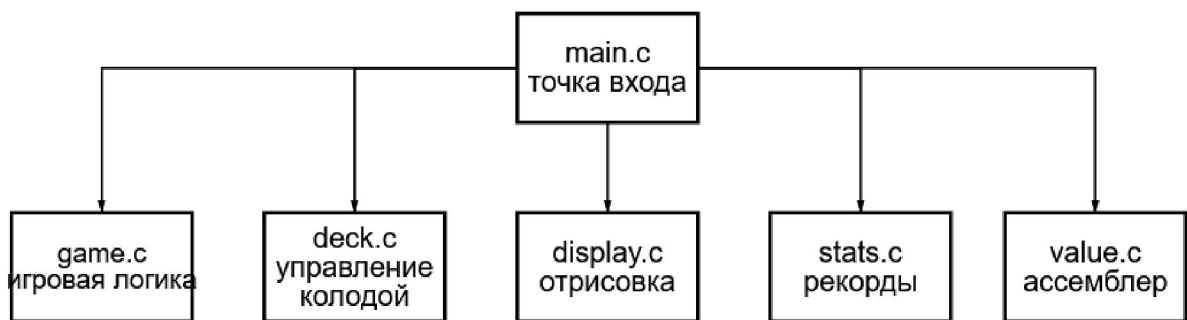


Рисунок 4 – Иерархическая структура программы

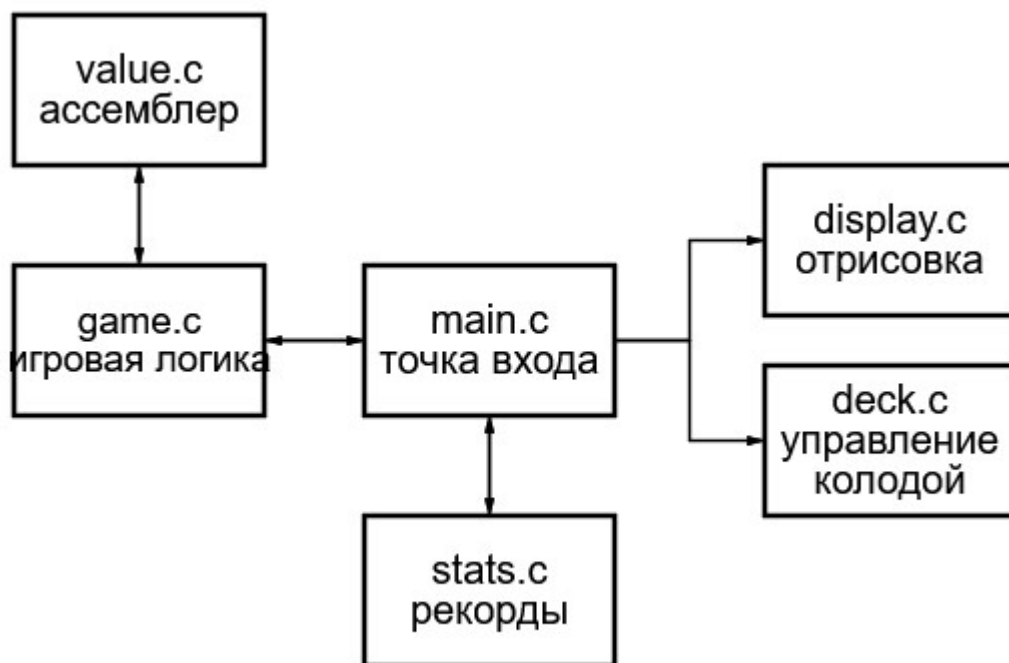


Рисунок 5 – Схема взаимодействия программы

6. Руководство пользователя

Программа предназначена для игры в «Двадцать одно». Управление осуществляется исключительно клавиатурой. Приложение не требует установки и запускается через исполняемый файл.

«Двадцать одно» — классическая математическая карточная игра, где игрок подбирает карты, стремясь набрать сумму очков, максимально близкую к 21, но не превышающую её. Игровое поле представляет собой консольный экран, на котором отображаются текущие карты игрока и алгоритмического оппонента (дилера), а также текущий счетчик очков. В свой ход игрок принимает решение: взять дополнительную карту для увеличения итоговой суммы или остановиться, зафиксировав результат. Каждая карта имеет свой номинал: числовые карты дают соответствующее количество очков, «картинки» оцениваются в 10 очков, а туз динамически приносит 1 или 11 очков для составления оптимальной комбинации. После остановки игрока в игру вступает оппонент, который действует по строгому алгоритму — он обязан добирать карты до достижения минимум 17 очков. Победа присуждается, если игрок набрал больше очков, чем оппонент, или если оппонент превысил лимит в 21 очко. Поражение наступает немедленно при превышении игроком суммы в 21 очко («перебор»), либо если итоговый счет оппонента оказался выше.

6.1 Заставка и Меню

При запуске программы открывается экран приветствия в виде стилизованного текста "TWENTY ONE", сгенерированного ASCII-символами. После нажатия клавиши Enter пользователь попадает в Главное меню.



Рисунок 6 – Главное меню

В меню доступны три опции, выбираемые цифровыми клавишами: 1 - Начать игру, 2 - Рекорды, 3 - Выход. При выборе начала игры программа запросит ввод имени пользователя.

6.2 Игровое поле и процесс игры

После ввода имени на экране отобразится игровой стол. Сверху показана статистика текущего игрока. По центру расположена рука дилера (первая карта скрыта) и рука игрока. Все карты отрисованы в виде рамок с соответствующими мастями и номиналами.

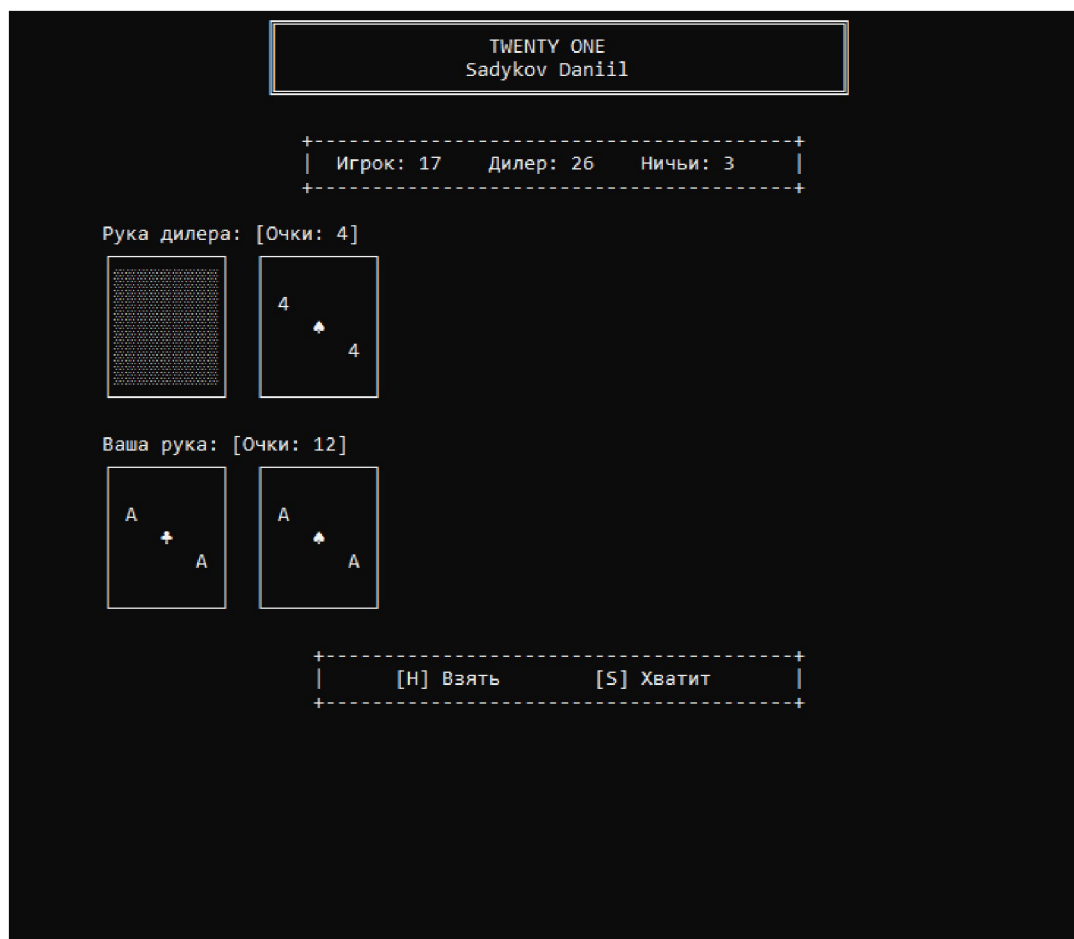


Рисунок 7 – Игровое поле

Внизу экрана располагается меню действий. Нажатие клавиши 'H' (Hit) позволяет взять дополнительную карту. Нажатие клавиши 'S' (Stand) останавливает добор и передает ход дилеру. Каждое действие сопровождается звуковыми сигналами (используется системный динамик).

6.3 Победа и поражение

Если сумма очков игрока превышает 21, он немедленно проигрывает. Если игрок нажал 'S', дилер открывает свою скрытую карту и добирает карты, пока его счет не достигнет 17 очков. После этого производится сравнение результатов.

На экран выводится сообщение о результатах ("Вы выиграли!", "Дилер выиграл." или "Ничья!"). Статистика обновляется автоматически. Пользователю предлагается нажать 'Y' для новой раздачи или 'N' для выхода в Главное меню.

Заключение

В ходе выполнения курсовой работы была успешно разработана программа — консольная карточная игра «Двадцать одно», реализованная на языке программирования Си с использованием ассемблерной вставки.

Данный проект позволил закрепить навыки работы со сложными структурами данных, указателями и массивами, а также продемонстрировал эффективное применение средств операционной системы Windows (WinAPI) для построения интерфейсов в текстовом режиме.

Основные достижения и реализованные функции:

Реализована полная игровая механика, включающая: генерацию и случайное перемешивание колоды (алгоритм Фишера-Йетса); логику ходов игрока (добор/остановка) и автоматизированный алгоритм дилера; динамический подсчёт очков с учётом плавающего номинала туза (1 или 11).

Применены низкоуровневые оптимизации: вычисление суммы очков в руке реализовано на уровне машинного кода (вставка `__asm`); работа с регистрами процессора (`eax`, `ebx`, `ecx`) напрямую для повышения эффективности цикла; прямое управление памятью средствами языка Си.

Разработан консольный пользовательский интерфейс: графическая ASCII-заставка и система меню с защищенным вводом имени; наглядная отрисовка игрового стола и карт с использованием псевдографики; применение функций WinAPI для цветовой индикации мастей (черви и бубны).

Реализована система сохранения результатов: чтение и запись персональной статистики (победы, поражения, ничьи) в файл `stats.txt`; формирование и полноэкранный вывод общей таблицы рекордов.

Выполненная работа продемонстрировала эффективность использования языка Си и встроенного ассемблера для разработки детерминированных логических игр. Программа успешно реализует все математические правила игры «Двадцать одно», сочетая структурное процедурное программирование с низкоуровневой оптимизацией ключевых алгоритмов. Приобретённые в ходе работы навыки имеют важное значение для дальнейшего профессионального роста в области системного программирования и освоения архитектуры ЭВМ.

Список используемых материалов

1. Керниган Б. В., Ритчи Д. М. Язык программирования С. — Пер. с англ. — М.: Вильямс, 2015. — 304 с.
2. Прата С. Язык программирования С. Лекции и упражнения. — Пер. с англ. — М.: Вильямс, 2015. — 928 с.
3. Абель П. Ассемблер. Язык и программирование для IBM PC. — Пер. с англ. — М.: Энтропия, 2001. — 736 с.
4. Таненбаум Э., Остин Т. Архитектура компьютера. — 6-е изд. — СПб.: Питер, 2013. — 816 с.
5. Рихтер Д. Windows для профессионалов: создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows. — 4е изд. — СПб.: Питер, 2008. — 720 с.
6. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ. — 3-е изд. — М.: Вильямс, 2013. — 1328 с.
7. Официальная документация Microsoft C/C++ [Электронный ресурс]. — URL: <https://learn.microsoft.com/ru-ru/cpp/c-language/>
8. Руководство по ассемблеру MASM [Электронный ресурс]. — URL:

9. Документация WinAPI (ConsoleReference) [Электронный ресурс]. — URL: <https://learn.microsoft.com/en-us/windows/console/>

10. Справочник по стандартной библиотеке Си (C Standard Library) [Электронный ресурс]. — URL: <https://en.cppreference.com/w/c>

Приложение А. Листинг программы

В данном приложении представлены ключевые фрагменты исходного кода программы.

Файл **main.c**

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <conio.h>
#include <windows.h>
#include <string.h>
#include <wchar.h>

#include "deck.h"
#include "game.h"
#include "display.h"
#include "stats.h"

static void setupConsole(void)
{
    system("mode con cols=92 lines=40"); system("title TWENTY ONE"); SetConsoleCP(65001); SetConsoleOutputCP(65001);

    HANDLE hOut = GetStdHandle(STD_OUTPUT_HANDLE);
    if (hOut != INVALID_HANDLE_VALUE) {          DWORD
        dwMode = 0;
        GetConsoleMode(hOut, &dwMode);
        dwMode |= 0x0004; /* ENABLE_VIRTUAL_TERMINAL_PROCESSING */
        SetConsoleMode(hOut, dwMode);
    }
}

static wchar_t getKey(void)
{
    return _getwch();
}
```

```

static char waitForInput(int gameState)
{
    wchar_t key;
    while (1) {
        key
        = getKey();
        if (gameState == STATE_PLAYING) {
            /* L'P' = 0x0420, L'p' = 0x0440, L'bl' = 0x042B, L'bl' = 0x044B */
            if (key == L'H' || key == L'h' || key == 0x0420 || key == 0x0440) return 'H';
            if (key == L'S' || key == L's' || key == 0x042B || key == 0x044B) return 'S';
        } else {
            /* L'H' = 0x041D, L'h' = 0x043D, L'T' = 0x0422, L't' = 0x0442 */
            if (key == L'Y' || key == L'y' || key == 0x041D || key == 0x043D) return
            'Y';
            if (key == L'N' || key == L'n' || key == 0x0422 || key == 0x0442)
            return 'N';
        }
    }
}

```

```

static void playRound(Deck* deck, GameStats* stats)
{
    GameState state;
    dealInitialCards(deck, &state);

```

```

    drawGameScreen(stats, &state, 1);

```

```

        int bjResult = checkInitialBlackjack(&state);
        if (bjResult != 2) {
            if (bjResult == 1) updateStats(stats, 1);
            else if
            (bjResult == -1) updateStats(stats, -1);
            else
            updateStats(stats, 0);

```

```

        savePlayerStats(stats); drawGameScreen(stats,
        &state, 0);
        return;
    }

```

```

    while (state.gameState == STATE_PLAYING) { drawGameScreen(stats,
    &state, 1);

```

```

        char action = waitForInput(state.gameState);

```

```

        if (action == 'H') {
            int busted = playerHit(deck, &state);
            if (busted) {
                updateStats(stats,
                -1);
                savePlayerStats(stats);
                drawGameScreen(stats, &state, 0);
                Beep(150, 500);
            }

```

```

        return;
    }
    if (getPlayerValue(&state) == 21) {
        break;
    }
    } else if (action == 'S')
    {
        break;
    }
}

drawGameScreen(stats, &state, 0);
Sleep(800);

while (getDealerValue(&state, 0) < 17)
{ dealerHit(deck, &state); drawGameScreen(stats,
&state, 0);
Sleep(800);
}

int result = determineWinner(&state); updateStats(stats,
result);
savePlayerStats(stats);

drawGameScreen(stats, &state, 0);
    if (result == 1) {
Beep(523, 150);
Sleep(50);
Beep(659, 150);
Sleep(50);
Beep(784, 250);
    }
    else if (result == -1) {
Beep(300, 200);
Sleep(100);
Beep(200, 400);
    }
    else {
Beep(440, 150);
Sleep(50);
Beep(440, 150);
    }
}

int main(void)
{
    Deck deck;
    GameStats stats;

```

```

srand((unsigned int)time(NULL)); setupConsole();
initDeck(&deck, 6);

showSplash();

while (1) {
drawMainMenu();
wchar_t menuKey = getKey();

    if (menuKey == L'1')
    {
        char
        name[MAX_NAME_LEN];
        showNameScreen(name);
        loadPlayerStats(name, &stats);

        int playAgain = 1;
        while (playAgain) {
            playRound(&deck, &stats);

            char
            key
            =
            waitForInput(STATE_GAME_OVER); if (key == 'N')
            {
                playAgain = 0;
            }
        }
    } else if (menuKey == L'2') {
        GameStatsrecords[MAX_RECORDS];
        int count = 0;
        loadAllStats(records, &count);
        drawRecordsScreen(records, count);
        _getch();
    } else if (menuKey == L'3')
    {
        break;
    }
}

clearScreen();
printf("\n\n\n");
printf("    СПАСИБО ЗА ИГРУ В TWENTY ONE!\n"); printf("\n");
printf("    [ ♠ ] [ ♣ ] [ ♥ ] [ ♦ ]\n");
printf("\n");
printf("    ДОСКОРЫХ ВСТРЕЧ!\n");

Sleep(1200);
return 0;
}

```

Файл value.c

```
#include "value.h"

int calculateHandValue_C(const int* hand, int count)
{
    int total = 0;
    int aces = 0;

    for (int i = 0; i < count; i++)
    {
        int card = hand[i];
        if (card >= 11 && card <= 13) {
            total += 10;
        }
        else if (card == 14)
        {
            total += 11;
            aces++;
        }
        else
        {
            total += card;
        }
    }

    while (total > 21 && aces > 0)
    {
        total -= 10;
        aces--;
    }

    return total;
}

int calculateHandValue_Asm(const int* hand, int count)
{
    int result;

    __asm
    {
        push ebx
        push esi

        mov esi, hand
        mov ecx, count
        mov eax, 0
        mov ebx, 0

        cmpecx, 0    je
        asm_done_summing

    asm_loop: cmpecx, 0
        jleasm_done_summing

        mov edx, [esi]
```

```

        mov dl, dl

cmp dl, 14
    je is_ace

cmp dl, 11
    jlnot_face_cardcmpd
l, 13 jgnot_face_card

        add eax, 10
    jmpnext_card

not_face_card: movzxdx,
dl
add eax, edx
    jmpnext_card

is_ace: add
    eax, 11
    incebx
    jmpnext_card

next_card:
    add esi, 4
    dec ecx
    jmpasm_loop

asm_done_summing:
ace_adjust_loop:
    cmpeax, 21
    jleasm_finishcmpebx,
0
    jleasm_finish

        sub eax, 10
    dec ebx
    jmpace_adjust_loop

asm_finish:
    mov result, eax

    pop esi
    pop ebx
}

    return result;
}

```

Файл game.c

```
#include "game.h"
#include "value.h"
#include <string.h>
#include <windows.h>

static int calcHandValue(const int* hand, int count)
{
    if (count == 0) return 0;
    return calculateHandValue_Asm(hand, count);
}

int getPlayerValue(const GameState* state)
{
    return calcHandValue(state->playerCards, state->playerCount);
}

int getDealerValue(const GameState* state, int hideFirst)
{
    if (hideFirst && state->dealerCount > 1) {
        return calcHandValue(state->dealerCards + 1, state->dealerCount - 1);
    }
    return calcHandValue(state->dealerCards, state->dealerCount);
}

void dealInitialCards(Deck* deck, GameState* state)
{
    state->playerCount = 0;
    state->dealerCount = 0;    state->gameState
= STATE_PLAYING;    state->message = NULL;

    memset(state->playerCards, 0, sizeof(state->playerCards)); memset(state->playerSuits, 0,
sizeof(state->playerSuits)); memset(state->dealerCards, 0, sizeof(state->dealerCards));
memset(state->dealerSuits, 0, sizeof(state->dealerSuits)); drawCardFromDeck(deck,
&state->playerCards[0], &state->playerSuits[0]);    state->playerCount = 1;

    drawCardFromDeck(deck, &state->dealerCards[0], &state->dealerSuits[0]);    state->dealerCount
= 1;

    drawCardFromDeck(deck, &state->playerCards[1], &state->playerSuits[1]);    state->playerCount
= 2;

    drawCardFromDeck(deck, &state->dealerCards[1], &state->dealerSuits[1]);    state->dealerCount
= 2;
}

int playerHit(Deck* deck, GameState* state)
{
```

```

    if (state->playerCount >= MAX_HAND_SIZE) return 0;

drawCardFromDeck(deck,
&state->playerCards[state->playerCount],
&state->playerSuits[state->playerCount]);    state->playerCount++;

    if (getPlayerValue(state) > 21)
    {
        state->gameState = STATE_GAME_OVER;
state->message = "ПЕРЕБОР! Вы проиграли."; return
1;
    }
    return 0;
}

void dealerHit(Deck* deck, GameState* state)
{
    if (state->dealerCount >= MAX_HAND_SIZE) return;

drawCardFromDeck(deck,
&state->dealerCards[state->dealerCount],
&state->dealerSuits[state->dealerCount]);    state->dealerCount++;
}

int checkInitialBlackjack(GameState* state)
{
    int playerValue = getPlayerValue(state);
    int dealerValue = calcHandValue(state->dealerCards, state->dealerCount);
int playerHasBJ = (state->playerCount == 2 && playerValue == 21);    int
dealerHasBJ = (state->dealerCount == 2 && dealerValue == 21);

    if
        (playerHasBJ && dealerHasBJ)
    {
        state->gameState = STATE_GAME_OVER;
state->message = "Двойной Блэкджек! Ничья!";
Beep(440, 150);
Sleep(50);
Beep(440, 150);
        return 0;
    } else if (playerHasBJ) {
        state->gameState = STATE_GAME_OVER;
state->message = "БЛЭКДЖЕК! Вы выиграли!";
Beep(784, 100);
Sleep(30);
Beep(988, 100);
Sleep(30);
Beep(1175, 100);
Sleep(30); Beep(1319,
300);
        return 1;
    }
}

```

```

    } else if (dealerHasBJ) {
        state->gameState = STATE_GAME_OVER;
        state->message = "Блэкджеку дилера! Вы проиграли.";
        Beep(300, 200);
        Sleep(100); Beep(200,
        400);
        return -1;
    }

    return 2; /* Нет блэкджека */
}

int determineWinner(GameState* state)
{
    state->gameState = STATE_GAME_OVER;

    int playerValue = getPlayerValue(state);
    int dealerValue = calcHandValue(state->dealerCards, state->dealerCount);

    if (playerValue > 21) {
        state->message = "ПЕРЕБОР! Дилер выиграл."; return -
        1;
    }

    if (dealerValue > 21) {
        state->message = "У дилера перебор! Вы выиграли!";
        return 1;
    }

    if (playerValue > dealerValue) {
        state->message = "Вы выиграли!";
        return 1;
    }
    else if (dealerValue > playerValue) {
        state->message = "Дилер выиграл.";
        return -1;
    }
    else {
        state->message = "Ничья!"; return
        0;
    }
}

```

Файл display.c

```

#define _CRT_SECURE_NO_WARNINGS
#include "display.h"
#include <windows.h>
#include <stdio.h>

```

```

#define CARD_WIDTH 11
#define CARD_GAP 2

static void gotoxy(int x, int y)
{
    COORD coord;
    coord.X = (SHORT)x;
    coord.Y = (SHORT)y;
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coord);
}

static const char* getRankSymbol(int value)
{
    switch (value) {
        case 2: return "2"; case 3: return "3"; case 4: return "4";
        case 5: return "5"; case 6: return "6"; case 7: return "7";
        case 8: return "8"; case 9: return "9"; case 10: return "10";
        case 11: return "J"; case 12: return "Q"; case 13: return "K";
        case 14: return "A"; default: return "?";
    }
}

static const char* getSuitSymbol(int suit)
{
    switch (suit) {
        case SUIT_SPADES: return "\xe2\x99\xa0";
        case SUIT_HEARTS: return "\xe2\x99\xa5";
        case SUIT_DIAMONDS: return "\xe2\x99\xa6";
        case SUIT_CLUBS: return "\xe2\x99\xa3";
        default: return "?";
    }
}

static void setCardColor(int suit)
{
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    WORD color;
    if (suit == SUIT_HEARTS || suit == SUIT_DIAMONDS) {
        color = FOREGROUND_RED | FOREGROUND_INTENSITY;
    } else {
        color = FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_BLUE |
        FOREGROUND_INTENSITY;
    }
    SetConsoleTextAttribute(hConsole, color);
}

static void resetColor(void)

```

```

{
SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),
    FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_BLUE |
    FOREGROUND_INTENSITY);
}

```

```

static void drawCard(int x, int y, int cardValue, int suit, int faceDown)

```

```

{
    if (faceDown) { gotoxy(x, y);
printf("┌───────────┐"); gotoxy(x, y + 1);
printf("│ ██████████ │"); gotoxy(x, y +
2); printf("│ ██████████ │"); gotoxy(x, y
+ 3); printf("│ ██████████ │"); gotoxy(x,
y + 4); printf("│ ██████████ │");
gotoxy(x, y + 5);
printf("│ ██████████ │"); gotoxy(x, y +
6); printf("└───────────┘"); return;
}

```

```

setCardColor(suit);

```

```

    const char* rank = getRankSymbol(cardValue);
    const char* suitSym = getSuitSymbol(suit);

```

```

gotoxy(x, y); printf("┌───────────┐");
gotoxy(x, y + 1); printf("│      │"); gotoxy(x,
y + 2); printf("│ %-8s │", rank); gotoxy(x, y +
3); printf("│  %s  │", suitSym); gotoxy(x, y
+ 4); printf("│ %8s │", rank); gotoxy(x, y + 5);
printf("│      │"); gotoxy(x, y + 6);
printf("└───────────┘"); resetColor();
}

```

```

static void drawHand(int startX, int startY, const int* cards, const int* suits, int count, int hideFirst)

```

```

{
    int xPos = startX; for (int
i = 0; i < count; i++) {
        int faceDown = (i == 0 &&hideFirst) ? 1 : 0;
drawCard(xPos, startY, cards[i], suits[i], faceDown);
xPos += CARD_WIDTH + CARD_GAP;
    }
}

```

```

void clearScreen(void)

```

```

{
    system("cls");
}

```

```

void showSplash(void)

```



```

printf(" _____ \n"); gotoxy(10, 4);
printf(" |__| | | /__\\ \n"); gotoxy(10, 5);
printf(" | |__ _ _ | |__ _ | | |__ _ \n");
gotoxy(10, 6); printf(" |\\ \\ ^\\ / _ \\ ' \\ | | | | | | | ' _
\\ / _ \\ \n"); gotoxy(10, 7); printf(" | |\\ \\ V V / _ / | | | | | | | |
| |__| | | | | _/ \n"); gotoxy(10, 8);
printf(" | _ | \\ \\ ^ \\ / \\ \\ | | | | \\ \\ | \\ \\ , | \\ \\ _ / | _ | | \\ \\ | \n");
gotoxy(10, 9); printf(" _ / | \n"); gotoxy(10, 10);
printf(" |__ / \n");

```

```

resetColor();

```

```

gotoxy(15, 12); printf("1. Начать игру"); gotoxy(15,
14); printf("2. Рекорды"); gotoxy(15, 16); printf("3.
Выход");
gotoxy(15, 20); printf("Выберите действие (1-3): ");
}

```

```

void drawRecordsScreen(const GameStats* records, int count)
{
clearScreen();

```

```

    int startX = 15;
    int y = 3;

```

```

gotoxy(startX, y++); printf("
.===== \n");
gotoxy(startX, y++); printf(" | ТАБЛИЦА РЕКОРДОВ | \n");
gotoxy(startX, y++); printf("
|===== \n");
gotoxy(startX, y++); printf(" | Имя игрока | Игры | Победы | Поражения | Ничьи | \n");
gotoxy(startX, y++);
printf(" |-----+-----+-----+-----+----- \n");

```

```

    if (count == 0)
    { gotoxy(startX, y++);
printf(" | Список рекордов пуст... | \n");
}
    else {
        for (int i = 0; i < count && i < 15; i++)
        { gotoxy(startX, y++);
printf(" | %-16s | %-6d | %-6d | %-9d | %-6d | \n",
records[i].name, records[i].totalGames, records[i].playerWins, records[i].dealerWins,
records[i].pushes);
        }
    }
}

```

```

gotoxy(startX, y++); printf(" '===== '

```



```

isValid = 1;
}
else {
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    SetConsoleTextAttribute(hConsole, FOREGROUND_RED | FOREGROUND_INTENSITY);

    gotoxy(36, 19);
    printf("Имя не может быть пустым!");

    resetColor(); Sleep(1500);
    }
    }
    else {
        strcpy(playerName, "Player"); isValid
        = 1;
    }
    }
}

void drawGameScreen(const GameStats* stats, const GameState* state, int hideDealerFirst)
{
    clearScreen();

    gotoxy(22, 0);
    printf("┌───────────────────────────────────────────────────────────────────────────────────┐");
    gotoxy(22, 1); printf("│                TWENTY ONE                │"); gotoxy(22, 2); printf("│                Sadykov Daniil                │"); gotoxy(22, 3);
    printf("└───────────────────────────────────────────────────────────────────────────────────┘");

    gotoxy(25, 5); printf("┌───────────────────────────────────────────────────────────────────────────────────┐");
    gotoxy(25, 6); printf("│ Игрок: %-3d Дилер: %-3d Ничьи: %-3d │", stats->playerWins, stats->dealerWins, stats->pushes);
    gotoxy(25, 7); printf("└───────────────────────────────────────────────────────────────────────────────────┘");

    int dealerVal = getDealerValue(state, hideDealerFirst); gotoxy(8,
9); printf("Рукадилера:");
    if (dealerVal > 0) printf(" [Очки: %d]", dealerVal);
    drawHand(8, 10, state->dealerCards, state->dealerSuits, state->dealerCount, hideDealerFirst);

    int playerVal = getPlayerValue(state); gotoxy(8,
18); printf("Вашарука:");
    if (playerVal > 0) printf(" [Очки: %d]", playerVal);
    drawHand(8, 19, state->playerCards, state->playerSuits, state->playerCount, 0);

    gotoxy(26, 27); printf("┌───────────────────────────────────────────────────────────────────────────────────┐");

    if (state->gameState == STATE_PLAYING) {
        gotoxy(26, 28); printf("│ [H] Взять [S] Хватит │");
    }
}

```

```

    } else {
        if (state->message) {
            HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
            WORD color = FOREGROUND_GREEN | FOREGROUND_INTENSITY;
            if (strstr(state->message, "проиграли") || strstr(state->message, "Дилервыиграл"))
            {
                color = FOREGROUND_RED | FOREGROUND_INTENSITY;
            } else if (strstr(state->message, "Ничья")) {
                color = FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_INTENSITY;
            }
            SetConsoleTextAttribute(hConsole, color); gotoxy(8,
26); printf("%s", state->message); resetColor();
        }
        gotoxy(26, 28); printf("|   [Y] Еще раз   [N] Выход   |");
    }
    gotoxy(26, 29); printf("+-----+"); gotoxy(0,
30);
}

```

Файл deck.c

```

#include "deck.h"
#include <stdlib.h>

static const int standardValues[CARDS_PER_DECK] = {
    2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
    2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
    2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
    2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14
};

static const int standardSuits[CARDS_PER_DECK] = {
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
    3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3
};

void initDeck(Deck* deck, int numDecks)
{
    if (numDecks < 1) numDecks = 1;
    if (numDecks > MAX_DECKS) numDecks = MAX_DECKS;

    deck->totalCards = numDecks * CARDS_PER_DECK;

    for (int d = 0; d < numDecks; d++) {        int offset
= d * CARDS_PER_DECK;        for (int i = 0; i <
CARDS_PER_DECK;        i++)
    {
        deck->values[offset + i] =
standardValues[i];
    }
}

```

```

        deck->suits[offset + i] = standardSuits[i];
    }
}

    deck->top = 0;
shuffleDeck(deck);
}

void shuffleDeck(Deck* deck)
{
    for (int i = deck->totalCards - 1; i > 0; i--) { int
j = rand() % (i + 1);

    int temp = deck->values[i];
    deck->values[i] = deck->values[j];
    deck->values[j] = temp;

    temp = deck->suits[i];
    deck->suits[i] = deck->suits[j];
    deck->suits[j] = temp;
    }
    deck->top = 0;
}

void drawCardFromDeck(Deck* deck, int* outValue, int* outSuit)
{
    if (deck->top >= deck->totalCards - RESHUFF_THRESHOLD) { shuffleDeck(deck);
    }

    *outValue = deck->values[deck->top];
    *outSuit = deck->suits[deck->top]; deck->top++;
}

```

Файл stats.c

```

#define _CRT_SECURE_NO_WARNINGS
#include "stats.h"
#include <stdio.h>
#include <string.h>

void loadAllStats(GameStats* records, int* count)
{
    *count = 0;
    FILE* fp = fopen(STATS_FILE, "r"); if
(!fp) return;

    while (*count < MAX_RECORDS && fscanf(fp, "%31s %d %d %d %d",
records[*count].name, &records[*count].playerWins,
&records[*count].dealerWins,

```

```

&records[*count].pushes,
&records[*count].totalGames) == 5)
{
    (*count)++;
}
fclose(fp);
}

void saveAllStats(const GameStats* records, int count)
{
    FILE* fp = fopen(STATS_FILE, "w");
    if (!fp) return;

    for (int i = 0; i < count; i++) {
        fprintf(fp, "%s %d %d %d %d\n", records[i].name, records[i].playerWins, records[i].dealerWins,
            records[i].pushes, records[i].totalGames);
    }
    fclose(fp);
}

void loadPlayerStats(const char* name, GameStats* stats)
{
    GameStats records[MAX_RECORDS];
    int count = 0;
    loadAllStats(records, &count);

    for (int i = 0; i < count; i++) {
        if (strcmp(records[i].name, name) == 0) {
            *stats = records[i];
        }
    }
    return;
}

strncpy(stats->name, name, MAX_NAME_LEN - 1);
stats->name[MAX_NAME_LEN - 1] = '\0';
stats->playerWins = 0; stats->dealerWins = 0;
stats->pushes = 0;
stats->totalGames = 0;
}

void savePlayerStats(const GameStats* stats)
{
    GameStats records[MAX_RECORDS];
    int count = 0;
    loadAllStats(records, &count);

    int found = 0; for (int i = 0; i < count; i++) { if
        (strcmp(records[i].name, stats->name) == 0)
        { records[i] = *stats; found = 1;

```

```

        break;
    }
}

if (!found && count < MAX_RECORDS) {
    records[count++] = *stats;
}

saveAllStats(records, count);
}

void updateStats(GameStats* stats, int result)
{
    if (result > 0) stats->playerWins++;
    else if (result < 0) stats->dealerWins++;
    else stats->pushes++;

    stats->totalGames++;
}

```