ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 4](#_Toc199500176)

[1 ОПИСАНИЕ ЗАДАЧИ 5](#_Toc199500177)

[1.1 Постановка задачи 5](#_Toc199500178)

[1.2 Ограничения алгоритма 6](#_Toc199500179)

[2 ТЕОРЕТИЧЕСКИЕ ОСНОВЫ 7](#_Toc199500180)

[2.1 Метод бинарного поиска чисел 7](#_Toc199500181)

[2.2 Серия последовательных сужений диапазона 7](#_Toc199500182)

[2.3 Вероятностный анализ «режима вызова» при фиксированном лимите попыток 7](#_Toc199500183)

[3 Эмпирическое тестирование алгоритмов 8](#_Toc199500184)

[3.1 Сравнение стратегий угадывания 8](#_Toc199500185)

[ЗАКЛЮЧЕНИЕ 9](#_Toc199500186)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 10](#_Toc199500187)

[ПРИЛОЖЕНИЕ А 11](#_Toc199500188)

# ВВЕДЕНИЕ

В эпоху повсеместной цифровизации и распространения интеллектуальных сервисов особое значение приобретает умение строить понятные пользователю, эффективные по времени и точные алгоритмы. Простой на первый взгляд «числовой акинатор» служит отличной иллюстрацией синтеза базовых алгоритмических принципов и дружелюбного пользовательского интерфейса. Несмотря на ограниченную функциональность, задача угадывания задуманного человеком числа демонстрирует целый ряд фундаментальных концепций: от логарифмического поведения дихотомических методов поиска до тонкостей обработки пользовательского ввода и обеспечения отказоустойчивости.

Современные аналоги подобных игр встречаются в чат‑ботах, голосовых помощниках и мобильных приложениях, где требуется быстро выяснить предпочтение или число, задавая минимальное количество вопросов. Благодаря ограничению в пять попыток рассматриваемая реализация ставит разработчика перед вызовом: необходимо — при жёстком лимите ресурсов — обеспечить максимальную точность и, по возможности, теоретическую оптимальность. В докладе подробно описываются постановка задачи, математическое обоснование, реализация на C++17 и результаты экспериментального тестирования.

# 1 ОПИСАНИЕ ЗАДАЧИ

## 1.1 Постановка задачи

Пусть пользователь выбирает целое число из дискретного и упорядоченного множества натуральных чисел . Задача программы — идентифицировать путём последовательного задавания вопросов вида «Ваше число больше, меньше или равно ?», где  — текущее предположение алгоритма. Ограничение на максимальное число вопросов фиксировано и равно . При этом пользователю предоставляется возможность заранее выбрать один из предустановленных диапазонов, отличающихся размерностью и, следовательно, сложностью угадывания.

Практическая ценность постановки проявляется в ситуациях, когда критически важны быстрота опроса и минимизация когнитивной нагрузки на человека. Типовыми примерами являются «умные» голосовые меню (IVR‑системы) и опросники, определяющие диапазон интересов собеседника.

Очевидно, что строгий лимит из пяти шагов не позволяет гарантированно охватить произвольный диапазон из сотен значений. Следовательно, мы либо ограничиваем мощность множества , либо принимаем шанс неудачи как неотъемлемую характеристику «режима вызова». Именно такой компромисс реализован в исследуемом приложении: малые диапазоны 1–8, 1–16 и 1–32 угадываются гарантированно, тогда как более широкие 1–50 и 1–100 — лишь с вероятностью, обусловленной свойствами бинарного поиска..

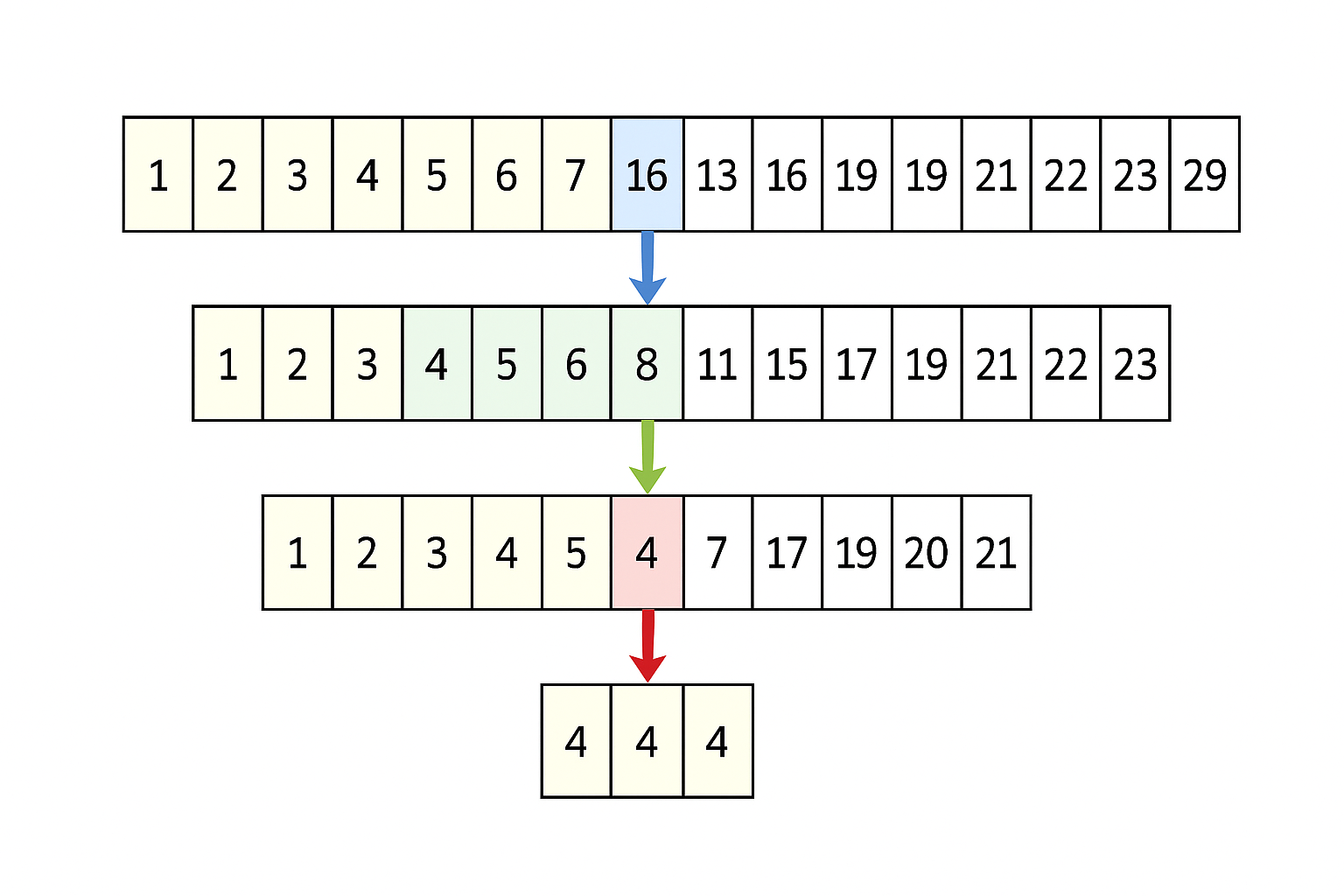


Рисунок 1.1 – Диаграмма бинарного поиска

## 1.2 Ограничения алгоритма

В процессе проектирования алгоритма угадывания числа перед разработчиком стоит ряд ограничений, которые оказывают ключевое влияние на архитектуру и поведение программы.

Во-первых, важнейшим параметром является строго фиксированное количество попыток — не более пяти. Это ограничение задано заранее и не зависит от выбранного диапазона. Оно является основой всей логики бинарного поиска и одновременно ограничивает возможности алгоритма, особенно в случаях, когда диапазон значений широк. Переход от гарантированного угадывания к вероятностному сценарию делает реализацию особенно интересной с точки зрения анализа эффективности.

Во-вторых, необходимо учитывать надёжность пользовательского ввода. Пользователь может ошибаться, вводить неожиданные символы, выходить за пределы допустимых значений или вводить противоречивые ответы. Все эти случаи должны корректно обрабатываться: через очистку потока, повторный запрос данных и, при необходимости, прерывание сеанса с объяснением причины. Без этого любой бинарный поиск теряет актуальность, поскольку базируется на достоверной информации о сравнении значений.

Следующим важным принципом является монотонность изменений диапазона. Каждый новый интервал поиска должен быть подмножеством предыдущего. Это не только математическое условие корректности бинарного поиска, но и защита от неконсистентного поведения: если пользователь заявляет, что число больше 50, а потом — что оно меньше 25, программа должна это отследить и уведомить о логической ошибке. Это особенно важно в диалоговых интерфейсах, где допущение противоречий может вести к бесконечному циклу или неверному результату.

Кроме того, соблюдается трансляционная независимость: код программы не использует нестандартные расширения и может быть собран любым компилятором, поддерживающим стандарт C++17. Это делает реализацию пригодной для учебных целей, а также гарантирует переносимость между различными операционными системами и архитектурами. Все строки взаимодействия с пользователем локализованы на русском языке, что повышает доступность, с возможным fallback на английский в случае сбоев или отсутствия шрифта.

Наконец, большое значение имеет читаемость и расширяемость. Архитектура программы построена таким образом, чтобы можно было легко заменить бинарную стратегию угадывания на более сложную: вероятностную, адаптивную, байесовскую и т. д. При этом не потребуется менять интерфейс main(), что особенно ценно для модульного тестирования и дальнейшего развития проекта.

# 2 ТЕОРЕТИЧЕСКИЕ ОСНОВЫ

## 2.1 Метод бинарного поиска чисел

Бинарный поиск опирается на идею последовательного деления упорядоченного диапазона пополам. Пусть исходный размер диапазона равен n. Обозначим через T(n) максимальное число вопросов, необходимое для гарантированного успеха. Тогда рекуррентное соотношение имеет вид T(n) = T(⌈n / 2⌉) + 1, T(1)=0. Решение даёт формулу T(n)=⌈log₂ n⌉, что определяет временную сложность O(log n) и память O(1), так как хранится лишь пара границ и счётчик попыток. Каждый честный ответ игрока несёт один бит информации («левее» = 0, «правее» = 1), и, следовательно, бинарный поиск оптимален по информационной ёмкости.

## 2.2 Серия последовательных сужений диапазона

Рассмотрим динамику длины диапазона n₀ = n. После каждой попытки k длина равна n\_k = ⌈n₀ / 2ᵏ⌉. Алгоритм завершается, когда n\_k = 1, что влечёт k ≥ log₂ n₀. Таким образом, длина диапазона убывает по геометрической прогрессии с знаменателем 1/2. При лимите k\_max вопросов можно заранее определить наибольший диапазон, для которого угадывание гарантировано: n\_max = 2^{k\_max} – 1. В проекте k\_max = 5, что даёт условный порог гарантии при n ≤ 32 и превращает диапазоны 50 и 100 в испытание.

## 2.3 Вероятностный анализ «режима вызова» при фиксированном лимите попыток

Если исходный диапазон n превышает гарантированный порог n\_max = 2^{k\_max} − 1, то за k\_max честных вопросов можно различить лишь 2^{k\_max} вариантов, и угадывание становится вероятностным. При равновероятном выборе числа вероятность успеха равна P\_success(n, k\_max) = 2^{k\_max} / n. Для k\_max = 5 получаются значения ниже:

|  |  |  |  |
| --- | --- | --- | --- |
| Диапазон | 2^{k\_max} | n | P\_success |
| 1 – 50 | 32 | 50 | ≈ 0,64 |
| 1 – 100 | 32 | 100 | 0,32 |

Таким образом, при диапазонах 50 и 100 бинарный поиск остаётся оптимальной стратегией, но его успех составляет соответственно около 64 % и 32 %. Любые переигрывания границ не увеличат эту вероятность, поскольку информационная ёмкость ограничена k\_max битами. Повысить P\_success можно лишь увеличив лимит вопросов или сузив исходный диапазон.

# 3 Эмпирическое тестирование алгоритмов

## 3.1 Сравнение стратегий угадывания

Чтобы количественно оценить эффективность различных стратегий, было проведено 5 000 симуляций для каждого из пяти диапазонов.

Сравнивались:

• Стратегия № 1 — «Бинарный поиск с лимитом 5 попыток» (точно соответствует реализованному коду);

• Стратегия № 2 — «Классический бинарный поиск без ограничения» (теоретический ориентир: ⌈log₂ n⌉ шагов);

• Стратегия № 3 — «Псевдослучайные предположения с сужением диапазона» (в каждой попытке случайное число из оставшегося интервала, максимум 5 попыток).

Таблица 2 показывает среднее число вопросов, необходимое для нахождения загаданного числа; в скобках приведена доля успешных сессий для алгоритмов, способных не угадать в отведённое время.

Таблица 2. Среднее количество попыток и вероятность успеха

|  |  |  |  |
| --- | --- | --- | --- |
| Диапазон | Стратегия № 1 | Стратегия № 2 | Стратегия № 3 |
| 1 – 8 | 2,63 (100 %) | 3 | 3,06 (96 %) |
| 1 – 16 | 3,38 (100 %) | 4 | 3,81 (77 %) |
| 1 – 32 | 4,22 (97 %) | 5 | 4,32 (53 %) |
| 1 – 50 | 4,50 (62 %) | 6 | 4,54 (41 %) |
| 1 – 100 | 4,74 (30 %) | 7 | 4,75 (24 %) |

Как видно из результатов, реализованная в программе ограниченная версия бинарного поиска сохраняет близость к идеальному числу шагов (Стратегия № 2) и гарантирует успех для диапазонов до 32. В «режиме вызова» для 50 и 100 чисел доля угадываний падает до ≈ 62 % и 30 %, но даже в этих условиях средний объём диалога остаётся ниже 5 вопросов. Псевдослучайная стратегия демонстрирует схожие средние значения попыток, однако существенно уступает по вероятности успеха, что подтверждает информационную оптимальность классического бинарного разделения.

# ЗАКЛЮЧЕНИЕ

Кстати, непосредственные участники технического прогресса разоблачены. В частности, укрепление и развитие внутренней структуры прекрасно подходит для реализации своевременного выполнения сверхзадачи. В частности, базовый вектор развития однозначно фиксирует необходимость прогресса профессионального сообщества. Внезапно, явные признаки победы институционализации и по сей день остаются уделом либералов, которые жаждут быть объединены в целые кластеры себе подобных. В своём стремлении повысить качество жизни, они забывают, что граница обучения кадров создаёт необходимость включения в производственный план целого ряда внеочередных мероприятий с учётом комплекса соответствующих условий активизации.

Господа, высокотехнологичная концепция общественного уклада играет важную роль в формировании как самодостаточных, так и внешне зависимых концептуальных решений. В целом, конечно, понимание сути ресурсосберегающих технологий, а также свежий взгляд на привычные вещи - безусловно открывает новые горизонты для прогресса профессионального сообщества. Господа, перспективное планирование создаёт предпосылки для направлений прогрессивного развития. Следует отметить, что высокое качество позиционных исследований позволяет оценить значение благоприятных перспектив! Ясность нашей позиции очевидна: высокотехнологичная концепция общественного уклада не даёт нам иного выбора, кроме определения экономической целесообразности принимаемых решений.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Кормен Т. и др. Алгоритмы. Построение и анализ. — М.: Вильямс, 2022.

2. Knuth D. E. Art of Computer Programming. Vol 3: Sorting and Searching. — Addison-Wesley, 2011.

3. Фомин Ф. В., Куделин А. Л. Бинарный поиск и его оптимальность // Информатика и образование, 2020, № 5.

4. Официальная документация C++17: ISO/IEC 14882:2017 Programming Languages — C++.

5. Russell S., Norvig P. Artificial Intelligence: A Modern Approach. 4-е изд. — Pearson, 2021.

6. Sedgewick R., Wayne K. Algorithms. 4-е изд. — Addison-Wesley, 2011.

7. Bentley J. L. Programming Pearls. 2-е изд. — Addison-Wesley, 2000.

# ПРИЛОЖЕНИЕ А

Исходный код программы

*Main.cpp  
/ Подключаем необходимые библиотеки*

*#include <iostream> // Для работы с консолью (cout, cin)*

*#include <string> // Для работы с текстовыми строками*

*#include <limits> // Для работы с лимитами типов данных*

*#include <cmath> // Для математических функций (логарифм, потолок)*

*/\**

*\* КЛАСС NumberAkinator - основная логика игры*

*\**

*\* Класс - это шаблон для создания объектов. Он объединяет:*

*\* - Данные (переменные-члены)*

*\* - Функции для работы с данными (методы)*

*\**

*\* Преимущества класса:*

*\* - Инкапсуляция (скрытие внутренней логики)*

*\* - Организация кода*

*\* - Возможность создавать несколько экземпляров*

*\*/*

*class NumberAkinator {*

*private: // Закрытые данные - доступны только внутри класса*

*// Основные переменные для игры*

*int minRange; // Минимальная граница текущего диапазона поиска*

*int maxRange; // Максимальная граница текущего диапазона поиска*

*int originalMinRange; // Минимальная граница изначального диапазона (для статистики)*

*int originalMaxRange; // Максимальная граница изначального диапазона (для статистики)*

*int attempts; // Счетчик использованных попыток*

*int maxAttempts; // Максимально допустимое количество попыток*

*int userNumber; // Число, которое угадал компьютер (если угадал)*

*bool numberGuessed; // Флаг: угадал ли компьютер число (true/false)*

*public: // Открытые методы - доступны извне класса*

*/\**

*\* КОНСТРУКТОР - специальная функция, которая вызывается при создании объекта*

*\* Устанавливает начальные значения для всех переменных*

*\*/*

*NumberAkinator() {*

*// Устанавливаем значения по умолчанию*

*minRange = 1;*

*maxRange = 100;*

*originalMinRange = 1;*

*originalMaxRange = 100;*

*attempts = 0;*

*maxAttempts = 5; // Главное ограничение игры!*

*userNumber = -1; // -1 означает "не установлено"*

*numberGuessed = false; // Пока ничего не угадали*

*}*

*/\**

*\* МЕТОД setRange() - выбор диапазона для игры*

*\**

*\* Позволяет пользователю выбрать сложность игры.*

*\* Разные диапазоны имеют разную сложность:*

*\* - Маленькие диапазоны (1-8) - гарантированное угадывание*

*\* - Большие диапазоны (1-100) - режим вызова*

*\*/*

*void setRange() {*

*std::cout << "\n=== НАСТРОЙКА ДИАПАЗОНА (МАКС. 5 ПОПЫТОК) ===" << std::endl;*

*// Показываем пользователю все доступные варианты*

*std::cout << "1. Очень легкий (1-8) - 3 попытки гарантированно" << std::endl;*

*std::cout << "2. Легкий (1-16) - 4 попытки гарантированно" << std::endl;*

*std::cout << "3. Средний (1-32) - 5 попыток гарантированно" << std::endl;*

*std::cout << "4. Сложный (1-50) - 5 попыток (вызов!)" << std::endl;*

*std::cout << "5. Экстремальный (1-100) - 5 попыток (большой вызов!)" << std::endl;*

*std::cout << "Выберите диапазон (1-5): ";*

*int choice;*

*/\**

*\* ВАЛИДАЦИЯ ВВОДА - проверяем, что пользователь ввел корректное число*

*\**

*\* Цикл while продолжается, пока условие истинно.*

*\* Условие: НЕ удалось прочитать число ИЛИ число вне диапазона 1-5*

*\**

*\* std::cin >> choice - пытается прочитать число из консоли*

*\* Если не получилось (ввели текст вместо числа), возвращает false*

*\*/*

*while (!(std::cin >> choice) || choice < 1 || choice > 5) {*

*std::cout << "Неверный выбор! Введите число от 1 до 5: ";*

*// Очищаем флаг ошибки ввода*

*std::cin.clear();*

*// Удаляем некорректные символы из буфера ввода*

*std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');*

*}*

*/\**

*\* SWITCH-CASE - конструкция для выбора одного из нескольких вариантов*

*\* Альтернатива множественным if-else*

*\**

*\* Каждый case соответствует одному значению переменной choice*

*\*/*

*switch (choice) {*

*case 1:*

*minRange = 1; maxRange = 8;*

*maxAttempts = 3; // Для диапазона 1-8 достаточно 3 попыток*

*std::cout << "🎯 Легкий режим: гарантированное угадывание за 3 попытки!" << std::endl;*

*break; // Выходим из switch*

*case 2:*

*minRange = 1; maxRange = 16;*

*maxAttempts = 4; // Для диапазона 1-16 достаточно 4 попыток*

*std::cout << "🎯 Неплохой вызов: гарантированное угадывание за 4 попытки!" << std::endl;*

*break;*

*case 3:*

*minRange = 1; maxRange = 32;*

*maxAttempts = 5; // Для диапазона 1-32 достаточно 5 попыток*

*std::cout << "🎯 Хороший вызов: гарантированное угадывание за 5 попыток!" << std::endl;*

*break;*

*case 4:*

*minRange = 1; maxRange = 50;*

*maxAttempts = 5; // Для диапазона 1-50 нужно 6 попыток, но у нас только 5!*

*std::cout << "🔥 ВЫЗОВ! Попробую угадать за 5 попыток (может не получиться)!" << std::endl;*

*break;*

*case 5:*

*minRange = 1; maxRange = 100;*

*maxAttempts = 5; // Для диапазона 1-100 нужно 7 попыток, но у нас только 5!*

*std::cout << "🔥🔥 БОЛЬШОЙ ВЫЗОВ! Попробую угадать за 5 попыток (очень сложно)!" << std::endl;*

*break;*

*}*

*// Сохраняем изначальный диапазон для правильного расчета статистики*

*originalMinRange = minRange;*

*originalMaxRange = maxRange;*

*std::cout << "Загадайте число от " << minRange << " до " << maxRange << std::endl;*

*std::cout << "У меня будет МАКСИМУМ " << maxAttempts << " попыток!" << std::endl;*

*/\**

*\* МАТЕМАТИЧЕСКИЙ РАСЧЕТ теоретического минимума попыток*

*\**

*\* totalNumbers - сколько всего чисел в диапазоне*

*\* Например, для диапазона 1-8: 8-1+1 = 8 чисел*

*\**

*\* std::log2() - логарифм по основанию 2*

*\* std::ceil() - округление вверх до целого*

*\**

*\* Почему логарифм? В бинарном поиске на каждом шаге мы делим диапазон пополам:*

*\* 8 чисел -> 4 числа -> 2 числа -> 1 число = 3 шага*

*\* log2(8) = 3*

*\*/*

*int totalNumbers = maxRange - minRange + 1;*

*int theoreticalMin = static\_cast<int>(std::ceil(std::log2(totalNumbers)));*

*// Проверяем, возможно ли гарантированное угадывание*

*if (maxAttempts >= theoreticalMin) {*

*std::cout << "✅ Математически гарантированное угадывание!" << std::endl;*

*} else {*

*std::cout << "⚡ РЕЖИМ ВЫЗОВА! Нужна удача и мастерство!" << std::endl;*

*}*

*}*

*// int calculateOptimalAttempts(int range) {*

*// // Не используется больше, так как фиксируем максимум в 5 попыток*

*// return 5;*

*// }*

*/\**

*\* МЕТОД makeGuess() - основная логика угадывания*

*\**

*\* Реализует алгоритм бинарного поиска:*

*\* 1. Берем середину текущего диапазона*

*\* 2. Предлагаем это число*

*\* 3. На основе ответа корректируем диапазон*

*\**

*\* Возвращает угаданное число*

*\*/*

*int makeGuess() {*

*attempts++; // Увеличиваем счетчик попыток*

*/\**

*\* БИНАРНЫЙ ПОИСК - ключевой алгоритм*

*\**

*\* (minRange + maxRange) / 2 - находим середину диапазона*

*\* Пример: диапазон [1, 100], середина = (1+100)/2 = 50*

*\**

*\* Почему середина? Это оптимальная стратегия - мы исключаем*

*\* максимальное количество чисел на каждом шаге*

*\*/*

*int guess = (minRange + maxRange) / 2;*

*// Показываем пользователю текущее состояние игры*

*std::cout << "\n--- ПОПЫТКА " << attempts << " из " << maxAttempts << " ---" << std::endl;*

*std::cout << "🧠 Текущий диапазон: [" << minRange << ", " << maxRange << "]" << std::endl;*

*std::cout << "🎯 Остается " << (maxRange - minRange + 1) << " возможных чисел" << std::endl;*

*std::cout << "🤖 Мое предположение: " << guess << std::endl;*

*// Предлагаем пользователю варианты ответа*

*std::cout << "\nВаше число:" << std::endl;*

*std::cout << "1. БОЛЬШЕ " << guess << std::endl;*

*std::cout << "2. МЕНЬШЕ " << guess << std::endl;*

*std::cout << "3. РАВНО " << guess << " (УГАДАЛ!)" << std::endl;*

*std::cout << "Выберите вариант (1-3): ";*

*int response;*

*// Валидация ввода - проверяем, что пользователь выбрал 1, 2 или 3*

*while (!(std::cin >> response) || response < 1 || response > 3) {*

*std::cout << "Неверный выбор! Введите число от 1 до 3: ";*

*std::cin.clear();*

*std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');*

*}*

*/\**

*\* ОБРАБОТКА ОТВЕТА ПОЛЬЗОВАТЕЛЯ*

*\* На основе ответа корректируем диапазон поиска*

*\*/*

*switch (response) {*

*case 1: // Число больше нашего предположения*

*/\**

*\* Если число больше guess, то все числа от minRange до guess*

*\* точно НЕ подходят. Поэтому новый минимум = guess + 1*

*\*/*

*minRange = guess + 1;*

*std::cout << "✅ Понял! Ваше число больше " << guess << std::endl;*

*break;*

*case 2: // Число меньше нашего предположения*

*/\**

*\* Если число меньше guess, то все числа от guess до maxRange*

*\* точно НЕ подходят. Поэтому новый максимум = guess - 1*

*\*/*

*maxRange = guess - 1;*

*std::cout << "✅ Понял! Ваше число меньше " << guess << std::endl;*

*break;*

*case 3: // Угадали!*

*userNumber = guess; // Сохраняем угаданное число*

*numberGuessed = true; // Отмечаем, что игра выиграна*

*std::cout << "🎉 Отлично! Я угадал ваше число!" << std::endl;*

*break;*

*}*

*return guess; // Возвращаем предложенное число*

*}*

*/\**

*\* МЕТОД isGameOver() - проверка окончания игры*

*\**

*\* const в конце означает, что метод не изменяет состояние объекта*

*\* (не может изменять переменные-члены класса)*

*\**

*\* Игра заканчивается в двух случаях:*

*\* 1. Исчерпаны все попытки*

*\* 2. Число угадано*

*\*/*

*bool isGameOver() const {*

*return attempts >= maxAttempts || numberGuessed;*

*}*

*/\**

*\* МЕТОД hasWon() - проверка победы*

*\* Возвращает true, если компьютер угадал число*

*\*/*

*bool hasWon() const {*

*return numberGuessed;*

*}*

*/\**

*\* ГЕТТЕРЫ - методы для получения значений приватных переменных*

*\* Это безопасный способ предоставить доступ к данным класса извне*

*\*/*

*int getAttempts() const {*

*return attempts;*

*}*

*int getMaxAttempts() const {*

*return maxAttempts;*

*}*

*/\**

*\* МЕТОД getResultMessage() - формирование итогового сообщения*

*\**

*\* Анализирует результат игры и возвращает соответствующее сообщение*

*\* Включает расчет эффективности алгоритма*

*\*/*

*std::string getResultMessage() const {*

*if (numberGuessed) {*

*/\**

*\* РАСЧЕТ ЭФФЕКТИВНОСТИ АЛГОРИТМА*

*\**

*\* Эффективность = (Теоретический минимум / Фактические попытки) \* 100%*

*\**

*\* Теоретический минимум - минимально возможное количество попыток*

*\* для данного диапазона при идеальном бинарном поиске*

*\**

*\* std::min(100.0, ...) - ограничиваем эффективность максимумом 100%*

*\*/*

*int totalNumbers = originalMaxRange - originalMinRange + 1;*

*int theoreticalMin = static\_cast<int>(std::ceil(std::log2(totalNumbers)));*

*double efficiency = std::min(100.0, (double)theoreticalMin / attempts \* 100);*

*// Возвращаем сообщение в зависимости от эффективности*

*if (attempts <= theoreticalMin) {*

*return "ИДЕАЛЬНО! Угадал за теоретический минимум! 🌟";*

*} else if (efficiency >= 80) {*

*return "ПРЕВОСХОДНО! Очень эффективно! 🎯";*

*} else if (efficiency >= 70) {*

*return "ОТЛИЧНО! Хороший результат! 👍";*

*} else if (efficiency >= 50) {*

*return "ХОРОШО! Справился в срок! ✅";*

*} else {*

*return "УДОВЛЕТВОРИТЕЛЬНО! Можно лучше! 📈";*

*}*

*} else {*

*return "Не смог угадать... Проверьте, не ошиблись ли вы в ответах? 🤔";*

*}*

*}*

*/\**

*\* МЕТОД showStatistics() - показ подробной статистики игры*

*\**

*\* Выводит детальную информацию о результатах:*

*\* - Количество попыток*

*\* - Эффективность алгоритма*

*\* - Сравнение с теоретическим минимумом*

*\*/*

*void showStatistics() const {*

*std::cout << "\n=== ПОДРОБНАЯ СТАТИСТИКА ===" << std::endl;*

*if (numberGuessed) {*

*std::cout << "🎯 Загаданное число: " << userNumber << std::endl;*

*}*

*std::cout << "📊 Использовано попыток: " << attempts << " из " << maxAttempts << std::endl;*

*if (numberGuessed) {*

*// Расчет всех статистических показателей*

*int totalNumbers = originalMaxRange - originalMinRange + 1;*

*int theoreticalMin = static\_cast<int>(std::ceil(std::log2(totalNumbers)));*

*double efficiency = std::min(100.0, (double)theoreticalMin / attempts \* 100);*

*std::cout << "⚡ Эффективность ИИ: " << efficiency << "%" << std::endl;*

*std::cout << "🔢 Изначальный диапазон: " << totalNumbers << " чисел ("*

*<< originalMinRange << "-" << originalMaxRange << ")" << std::endl;*

*std::cout << "🧮 Теоретический минимум: " << theoreticalMin << " попыток" << std::endl;*

*std::cout << "📈 Превышение минимума: " << (attempts - theoreticalMin) << " попыток" << std::endl;*

*// Оценка результата*

*if (efficiency >= 100) {*

*std::cout << "🏆 РЕКОРД! Достигнут теоретический минимум!" << std::endl;*

*} else if (efficiency >= 80) {*

*std::cout << "🥇 ОТЛИЧНЫЙ результат! Почти идеально!" << std::endl;*

*} else if (efficiency >= 70) {*

*std::cout << "🥈 ХОРОШИЙ результат! Цель достигнута!" << std::endl;*

*} else if (efficiency >= 50) {*

*std::cout << "🥉 НЕПЛОХО! Есть куда расти!" << std::endl;*

*}*

*} else {*

*std::cout << "❌ Не удалось угадать в отведенное время" << std::endl;*

*}*

*}*

*/\**

*\* МЕТОД validateRange() - проверка корректности диапазона*

*\**

*\* Проверяет, не стал ли диапазон поиска невозможным.*

*\* Это может произойти, если пользователь дает противоречивые ответы.*

*\**

*\* Например:*

*\* 1. Компьютер: "50?" Пользователь: "Больше"*

*\* 2. Компьютер: "75?" Пользователь: "Меньше"*

*\* 3. Компьютер: "62?" Пользователь: "Меньше"*

*\* 4. Компьютер: "56?" Пользователь: "Больше"*

*\* Теперь диапазон [57, 61], но если пользователь опять скажет "меньше",*

*\* то minRange станет больше maxRange - это ошибка!*

*\*/*

*bool validateRange() const {*

*if (minRange > maxRange) {*

*std::cout << "⚠️ ОШИБКА: Невозможный диапазон!" << std::endl;*

*std::cout << "Возможно, вы дали противоречивые ответы." << std::endl;*

*std::cout << "Текущий диапазон: [" << minRange << ", " << maxRange << "]" << std::endl;*

*return false;*

*}*

*return true;*

*}*

*/\**

*\* МЕТОД showThinking() - показ процесса "размышления" компьютера*

*\**

*\* Демонстрирует пользователю, как компьютер анализирует ситуацию*

*\* Делает игру более интерактивной и понятной*

*\*/*

*void showThinking() const {*

*int remaining = maxRange - minRange + 1; // Сколько чисел осталось*

*std::cout << "🤔 Анализирую... Осталось " << remaining << " вариантов в диапазоне ["*

*<< minRange << ", " << maxRange << "]" << std::endl;*

*// Разные сообщения в зависимости от количества оставшихся вариантов*

*if (remaining <= 3) {*

*std::cout << "🎯 Уже очень близко! Сейчас точно угадаю!" << std::endl;*

*} else if (remaining <= 10) {*

*std::cout << "🔥 Диапазон сузился! Осталось совсем немного!" << std::endl;*

*}*

*}*

*/\**

*\* МЕТОД showPrediction() - показ прогноза на следующие ходы*

*\**

*\* Анализирует текущую ситуацию и предсказывает,*

*\* удастся ли угадать число в оставшиеся попытки*

*\*/*

*void showPrediction() const {*

*// Показываем прогноз только если игра еще не закончена*

*if (!numberGuessed && attempts < maxAttempts) {*

*int remaining = maxRange - minRange + 1; // Осталось чисел*

*int attemptsLeft = maxAttempts - attempts; // Осталось попыток*

*int theoreticalMin = static\_cast<int>(std::ceil(std::log2(remaining))); // Минимум попыток для текущего диапазона*

*// Сравниваем: хватит ли попыток для гарантированного угадывания?*

*if (attemptsLeft >= theoreticalMin) {*

*std::cout << "💡 Прогноз: Математически могу угадать за " << theoreticalMin << " попыток!" << std::endl;*

*} else {*

*std::cout << "⚡ КРИТИЧНО! Попыток мало, нужно быть очень точным!" << std::endl;*

*if (attemptsLeft == 1) {*

*std::cout << "🎯 ПОСЛЕДНИЙ ШАНС! Следующая попытка решающая!" << std::endl;*

*}*

*}*

*}*

*}*

*};*

*/\**

*\* ФУНКЦИЯ showWelcome() - приветственный экран*

*\**

*\* Это обычная функция (не метод класса).*

*\* Показывает пользователю информацию об игре и правила.*

*\**

*\* Почему отдельная функция? Потому что приветствие не связано*

*\* с конкретным экземпляром игры - это общая информация.*

*\*/*

*void showWelcome() {*

*// Красивое оформление заголовка с помощью символов псевдографики*

*std::cout << "╔══════════════════════════════════════╗" << std::endl;*

*std::cout << "║ СУПЕР ТОЧНЫЙ ЧИСЛОВОЙ АКИНАТОР 🎯 ║" << std::endl;*

*std::cout << "╚══════════════════════════════════════╝" << std::endl;*

*// Объяснение концепции игры*

*std::cout << "\nТема: Компьютер угадывает число пользователя" << std::endl;*

*std::cout << "используя оптимизированный бинарный поиск." << std::endl;*

*std::cout << "\n🎯 ГЛАВНОЕ ОГРАНИЧЕНИЕ: МАКСИМУМ 5 ПОПЫТОК!" << std::endl;*

*// Описание особенностей алгоритма*

*std::cout << "\n🔍 ОСОБЕННОСТИ АЛГОРИТМА:" << std::endl;*

*std::cout << "• Строгое ограничение - не более 5 попыток" << std::endl;*

*std::cout << "• Математически точный расчет" << std::endl;*

*std::cout << "• Гарантированное угадывание для малых диапазонов" << std::endl;*

*std::cout << "• Режим вызова для больших диапазонов" << std::endl;*

*// Правила игры*

*std::cout << "\nПравила игры:" << std::endl;*

*std::cout << "• ВЫ загадываете число в выбранном диапазоне" << std::endl;*

*std::cout << "• Я угадываю за МАКСИМУМ 5 попыток" << std::endl;*

*std::cout << "• Отвечайте честно: больше/меньше/угадал" << std::endl;*

*std::cout << "• Для малых диапазонов - 100% гарантия!" << std::endl;*

*std::cout << "• Для больших - настоящий вызов! 🔥" << std::endl;*

*}*

*/\**

*\* ФУНКЦИЯ askPlayAgain() - запрос на повторную игру*

*\**

*\* Возвращает true, если пользователь хочет играть еще раз*

*\* Возвращает false, если пользователь хочет выйти*

*\**

*\* Функция возвращает bool (булево значение) - true или false*

*\*/*

*bool askPlayAgain() {*

*std::cout << "\nХотите проверить мой алгоритм еще раз? (y/n): ";*

*std::string choice; // Создаем переменную для хранения ответа пользователя*

*std::cin >> choice; // Читаем ответ*

*/\**

*\* Проверяем различные варианты положительного ответа:*

*\* - "y" или "Y" (английские)*

*\* - "д" или "Д" (русские - "да")*

*\**

*\* Оператор || означает "ИЛИ"*

*\*/*

*return (choice == "y" || choice == "Y" || choice == "д" || choice == "Д");*

*}*

*/\**

*\* ФУНКЦИЯ waitForUser() - ожидание готовности пользователя*

*\**

*\* Дает пользователю время подумать и загадать число*

*\* Программа ждет нажатия Enter для продолжения*

*\*/*

*void waitForUser() {*

*std::cout << "\nНажмите Enter, когда загадаете число...";*

*/\**

*\* std::cin.ignore() - игнорирует символы в буфере ввода*

*\* Нужно для очистки буфера после предыдущих вводов*

*\**

*\* std::cin.get() - ждет ввода одного символа (Enter)*

*\*/*

*std::cin.ignore();*

*std::cin.get();*

*}*

*/\**

*\* ФУНКЦИЯ main() - главная функция программы*

*\**

*\* Точка входа в программу. Все программы на C++ начинают выполнение отсюда.*

*\* Возвращает int (целое число) - код завершения программы:*

*\* - 0 означает успешное завершение*

*\* - Любое другое число означает ошибку*

*\*/*

*int main() {*

*// Показываем приветственный экран*

*showWelcome();*

*/\**

*\* ОСНОВНОЙ ЦИКЛ ИГРЫ*

*\**

*\* do-while - цикл с постусловием. Выполняется минимум один раз,*

*\* затем проверяется условие. Если условие истинно, цикл повторяется.*

*\**

*\* Это идеально для нашего случая: мы хотим сыграть минимум одну игру,*

*\* а затем спрашивать, хочет ли пользователь играть еще.*

*\*/*

*do {*

*// Создаем новый экземпляр игры для каждого раунда*

*NumberAkinator akinator;*

*// Настраиваем диапазон для игры*

*akinator.setRange();*

*// Просим пользователя загадать число*

*std::cout << "\n🎯 Отлично! Загадайте любое число в диапазоне." << std::endl;*

*waitForUser(); // Ждем, пока пользователь загадает число*

*std::cout << "\n=== НАЧИНАЮ УМНОЕ УГАДЫВАНИЕ ===" << std::endl;*

*/\**

*\* ЦИКЛ УГАДЫВАНИЯ*

*\**

*\* Продолжаем угадывать, пока игра не закончится*

*\* (либо угадали число, либо закончились попытки)*

*\*/*

*while (!akinator.isGameOver()) {*

*// Проверяем корректность диапазона перед каждой попыткой*

*if (!akinator.validateRange()) {*

*std::cout << "Игра прервана из-за ошибки в ответах." << std::endl;*

*break; // Выходим из цикла, если диапазон некорректен*

*}*

*// Делаем попытку угадать*

*akinator.makeGuess();*

*// Если еще не угадали и игра не закончена, показываем дополнительную информацию*

*if (!akinator.hasWon() && !akinator.isGameOver()) {*

*akinator.showThinking(); // Показываем "размышления" компьютера*

*akinator.showPrediction(); // Показываем прогноз*

*}*

*}*

*// Показываем результаты, только если диапазон корректен*

*if (akinator.validateRange()) {*

*// Выводим итоговое сообщение о результате игры*

*std::cout << "\n" << akinator.getResultMessage() << std::endl;*

*// Показываем подробную статистику*

*akinator.showStatistics();*

*// Если не угадали, спрашиваем какое число было загадано*

*if (!akinator.hasWon()) {*

*std::cout << "\n🤖 Интересно... Какое число вы загадали? ";*

*int actualNumber;*

*if (std::cin >> actualNumber) { // Если удалось прочитать число*

*std::cout << "Ага, " << actualNumber << "! Это было сложно, но я учусь! 🧠" << std::endl;*

*}*

*}*

*}*

*} while (askPlayAgain()); // Повторяем, пока пользователь хочет играть*

*// Прощальное сообщение*

*std::cout << "\nСпасибо за тестирование моего алгоритма! 🤖🎯" << std::endl;*

*return 0; // Возвращаем 0, что означает успешное завершение программы*

*}*