

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)**



ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

Лабораторная работа №1

по дисциплине: Основы искусственного интеллекта

тема: «Алгоритм отжига»

Выполнил: ст. группы ПВ-223

Игнатъев Артур Олегович

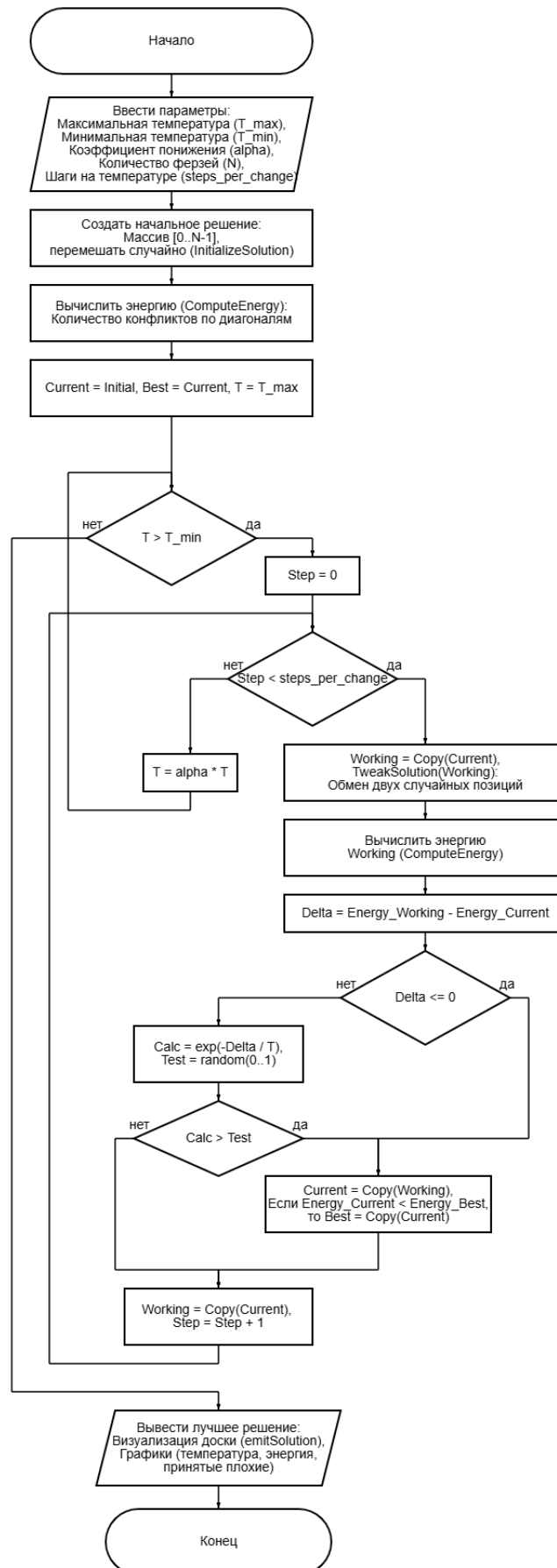
Проверили:

пр. Твердохлеб Виталий Викторович

Белгород 2025 г.

Цель работы: разработка и исследование алгоритма отжига в процессе решения числовой задачи оптимизации.

Ход работы



Тест 1

Стандартные условия

Имитация отжига

Количество ферзей (N > 20):

25

Начальная температура:

30.0

Конечная температура:

0.5

Коэффициент охлаждения (Alpha):

0.98

Шаги на температуру:

100

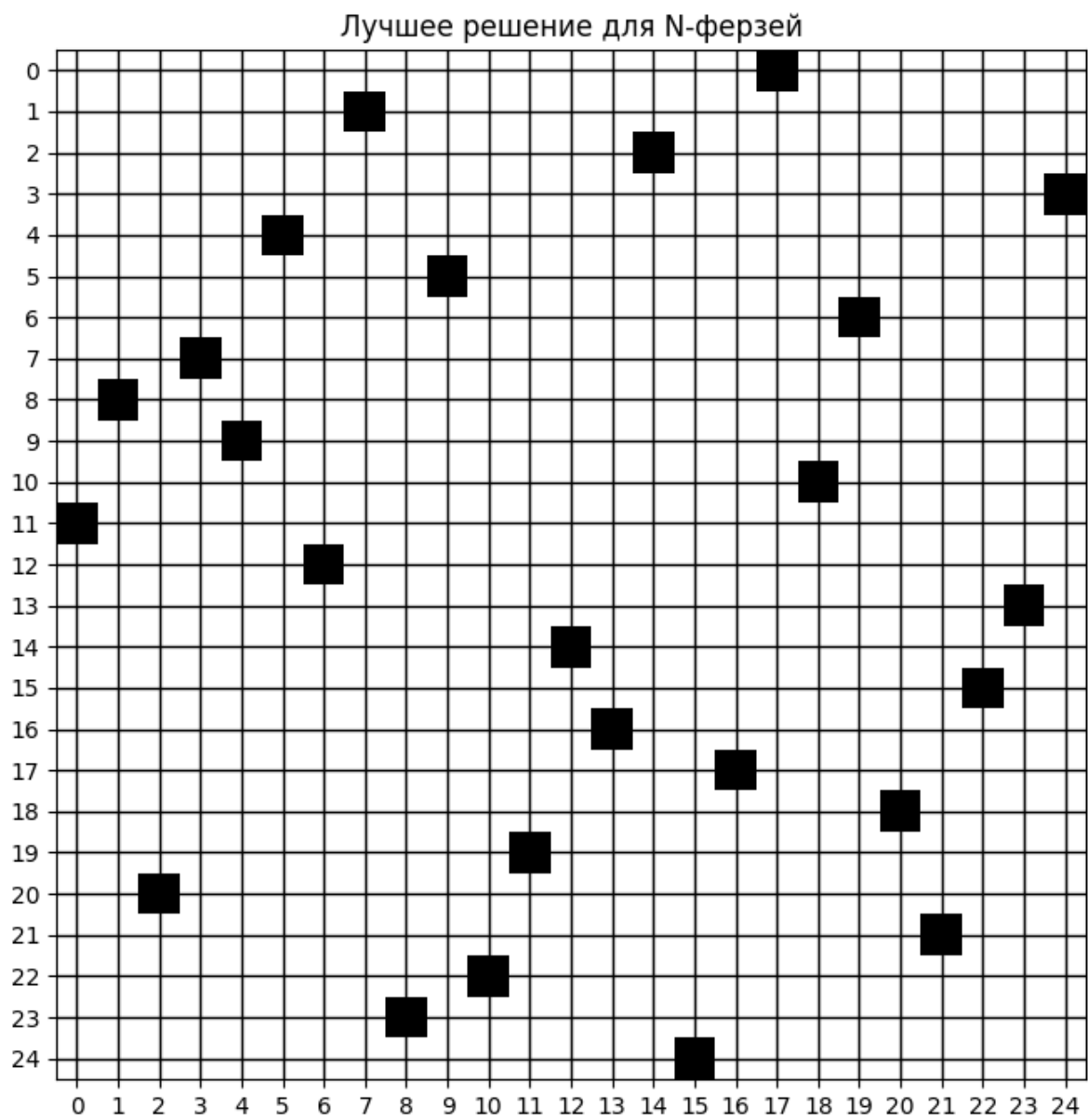
Запустить симуляцию

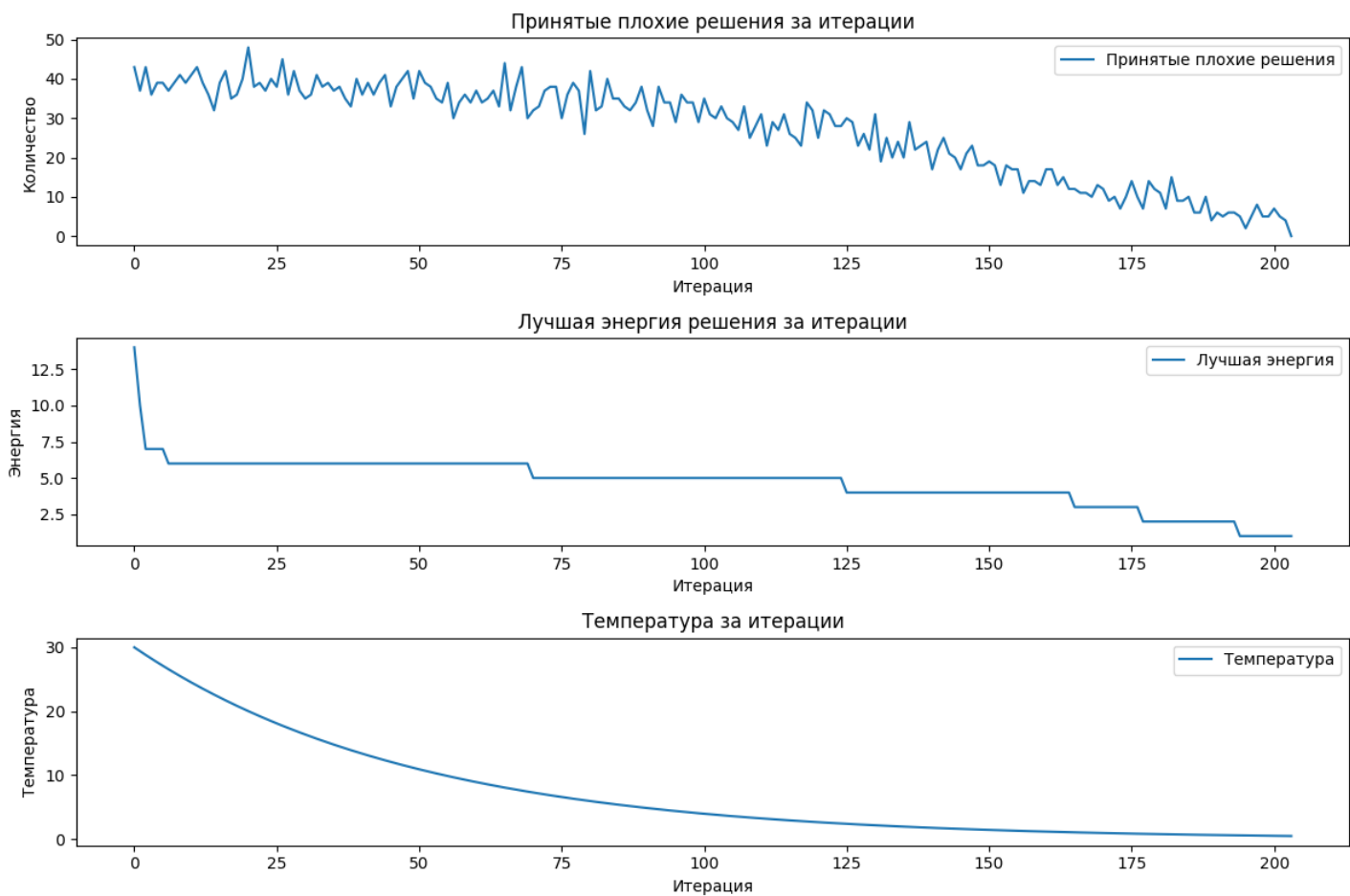
Частичный успех

!

Лучшая энергия: 1 (не ноль)

ОК





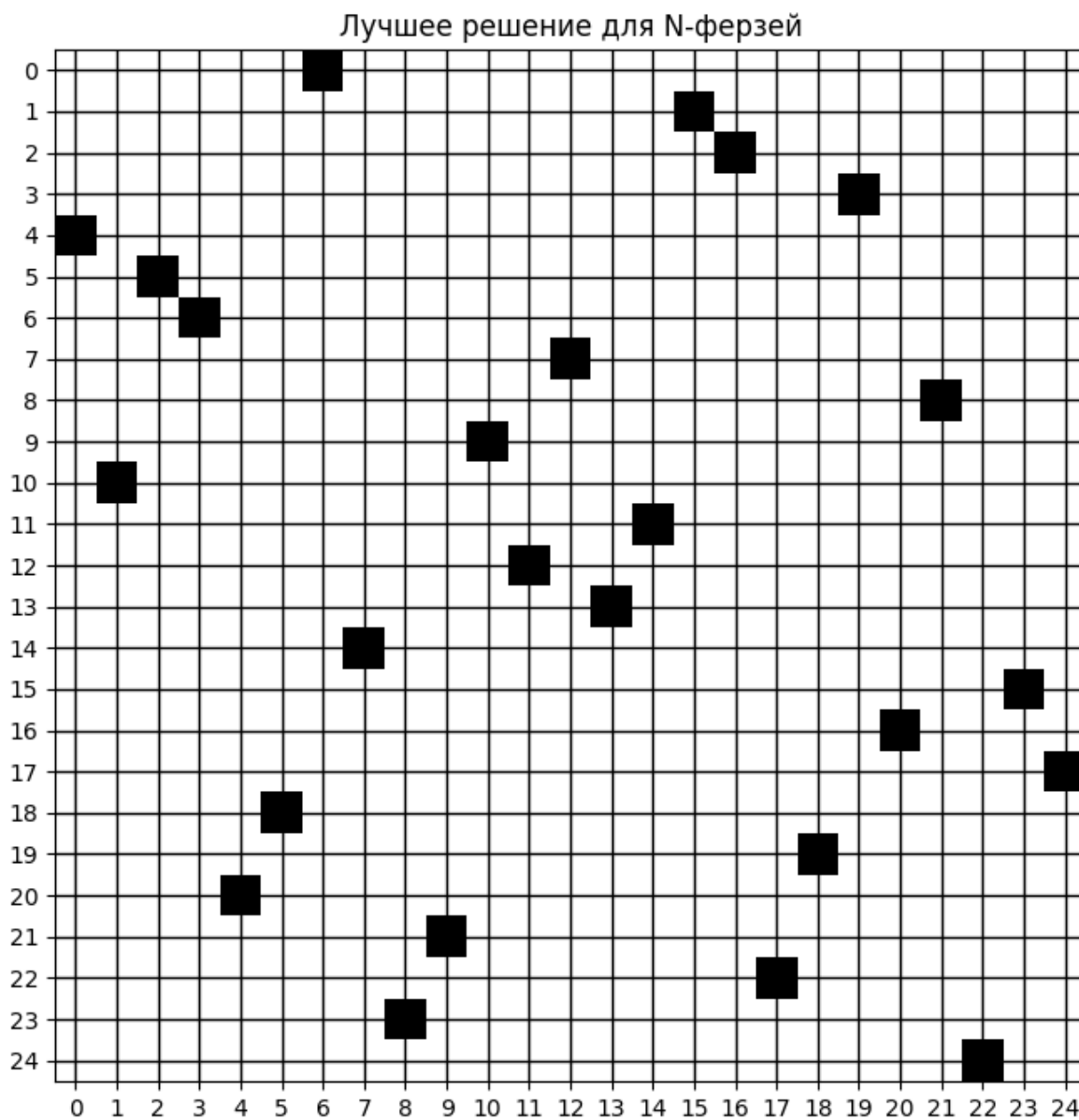
Результат не оптимален (энергия = 1). Графики показывают, что энергия падает с понижением температуры, а вероятность принятия плохих решений тоже уменьшается.

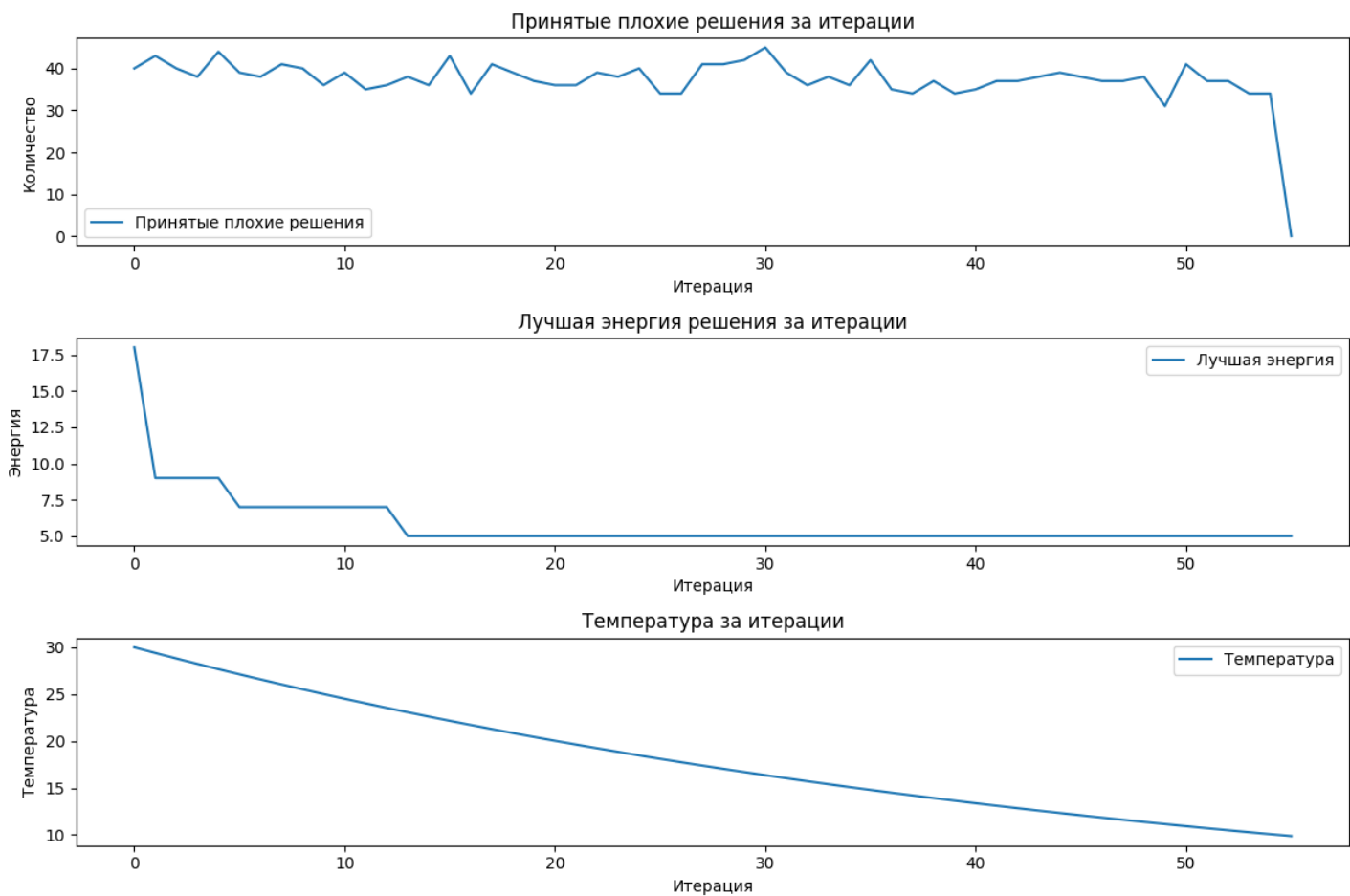
Тест 2

Увеличим конечную температуру

Имитация отжига	
Количество ферзей (N > 20):	25
Начальная температура:	30.0
Конечная температура:	10
Коэффициент охлаждения (Alpha):	0.98
Шаги на температуру:	100
<input type="button" value="Запустить симуляцию"/>	

Имитация отжига	
Количество ферзей (N > 20):	25
Начальная температура:	30.0
Конечная температура:	10
Коэффициент охлаждения (Alpha):	0.98
Шаги на температуру:	100
<input type="button" value="Запустить симуляцию"/>	





Лучшая энергия составила 5. Графики показывают множество плохих решений, которые “застряли на одном уровне”. Однако алгоритм достаточно быстро нашел лучшую энергию.

Тест 3

Увеличим начальную температуру

Имитация отжига

Количество ферзей (N > 20):

25

Начальная температура:

500.0

Конечная температура:

0.5

Коэффициент охлаждения (Alpha):

0.98

Шаги на температуру:

100

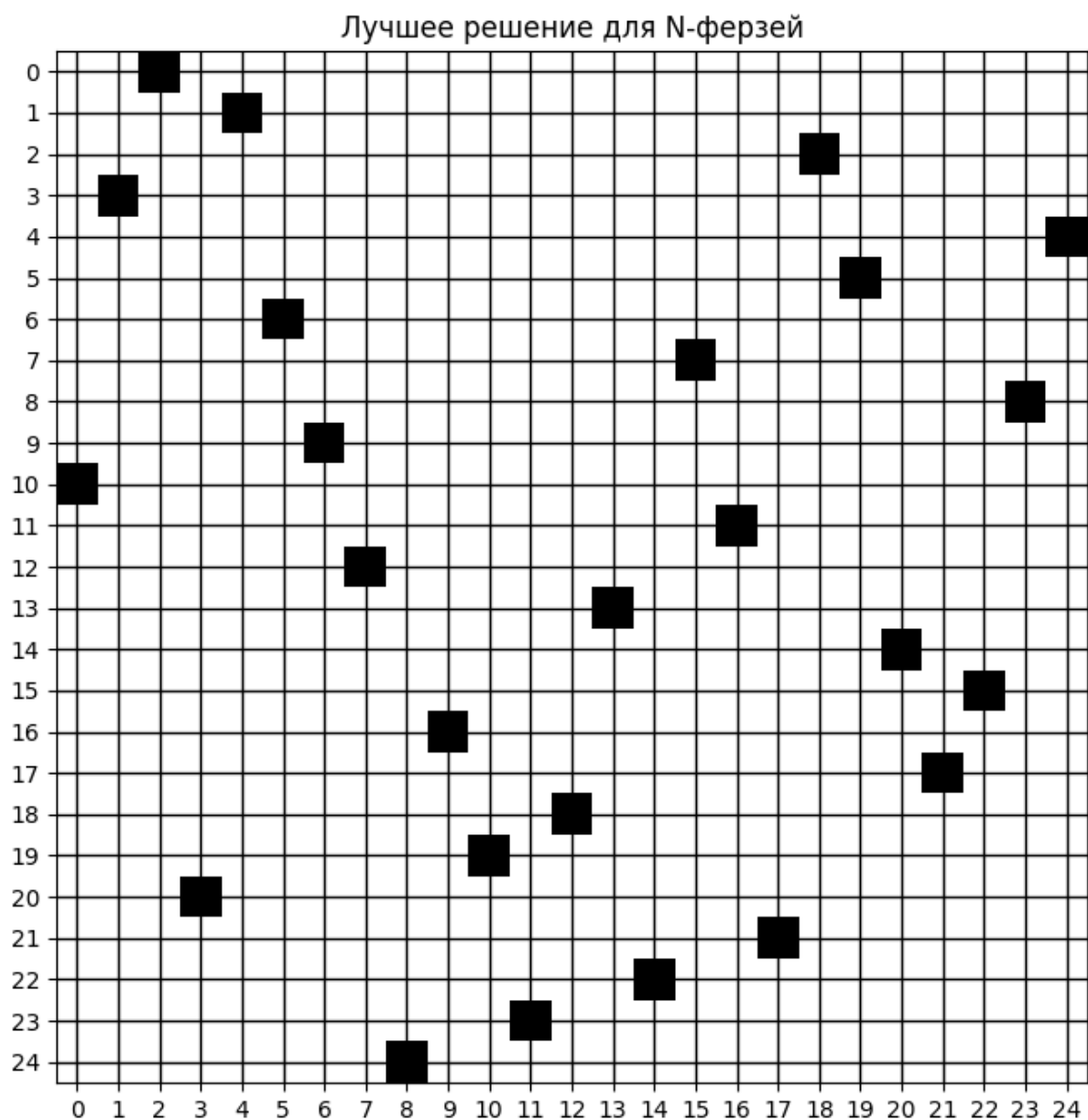
Запустить симуляцию

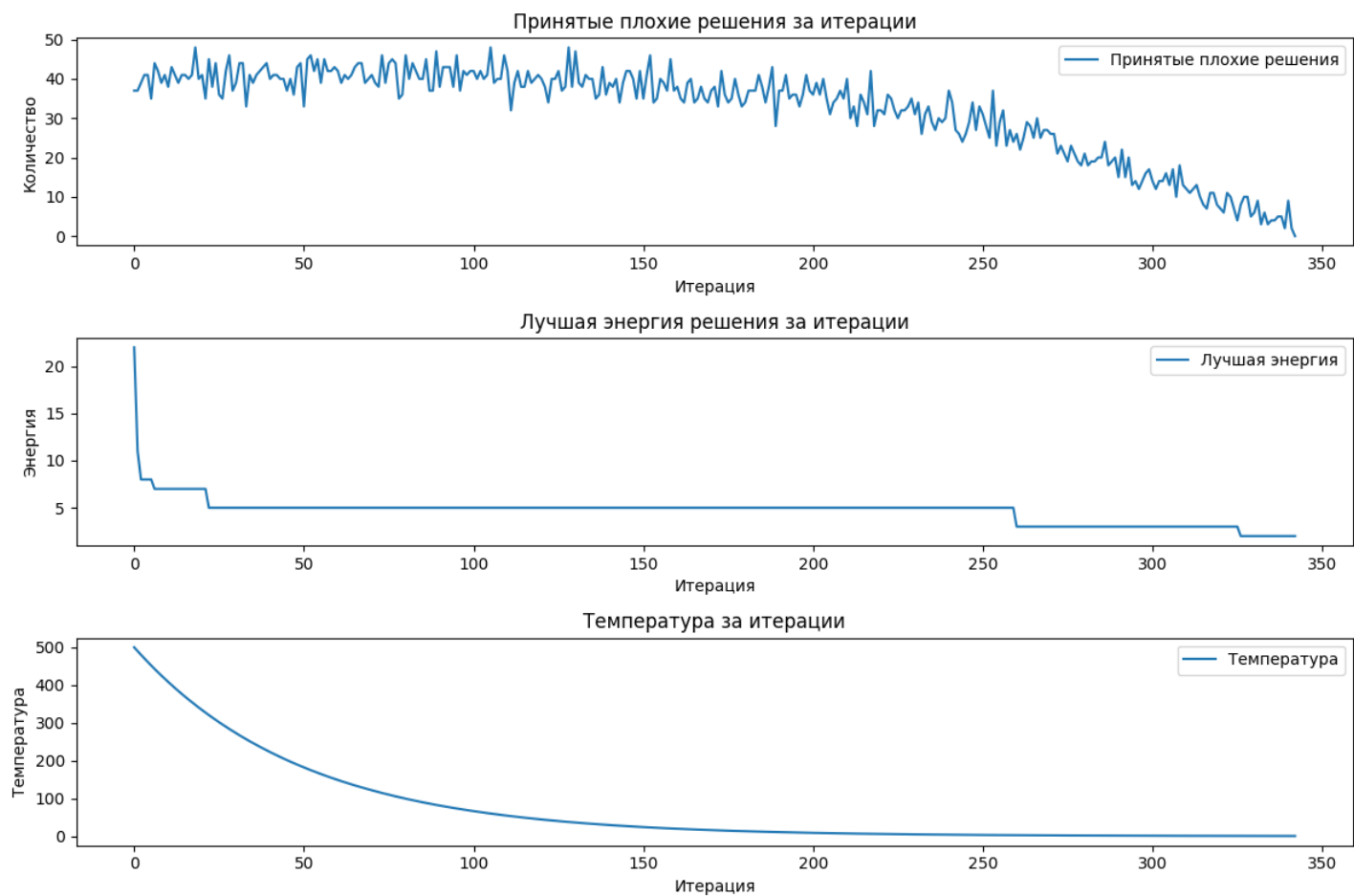
Частичный успех

!

Лучшая энергия: 2 (не ноль)

ОК






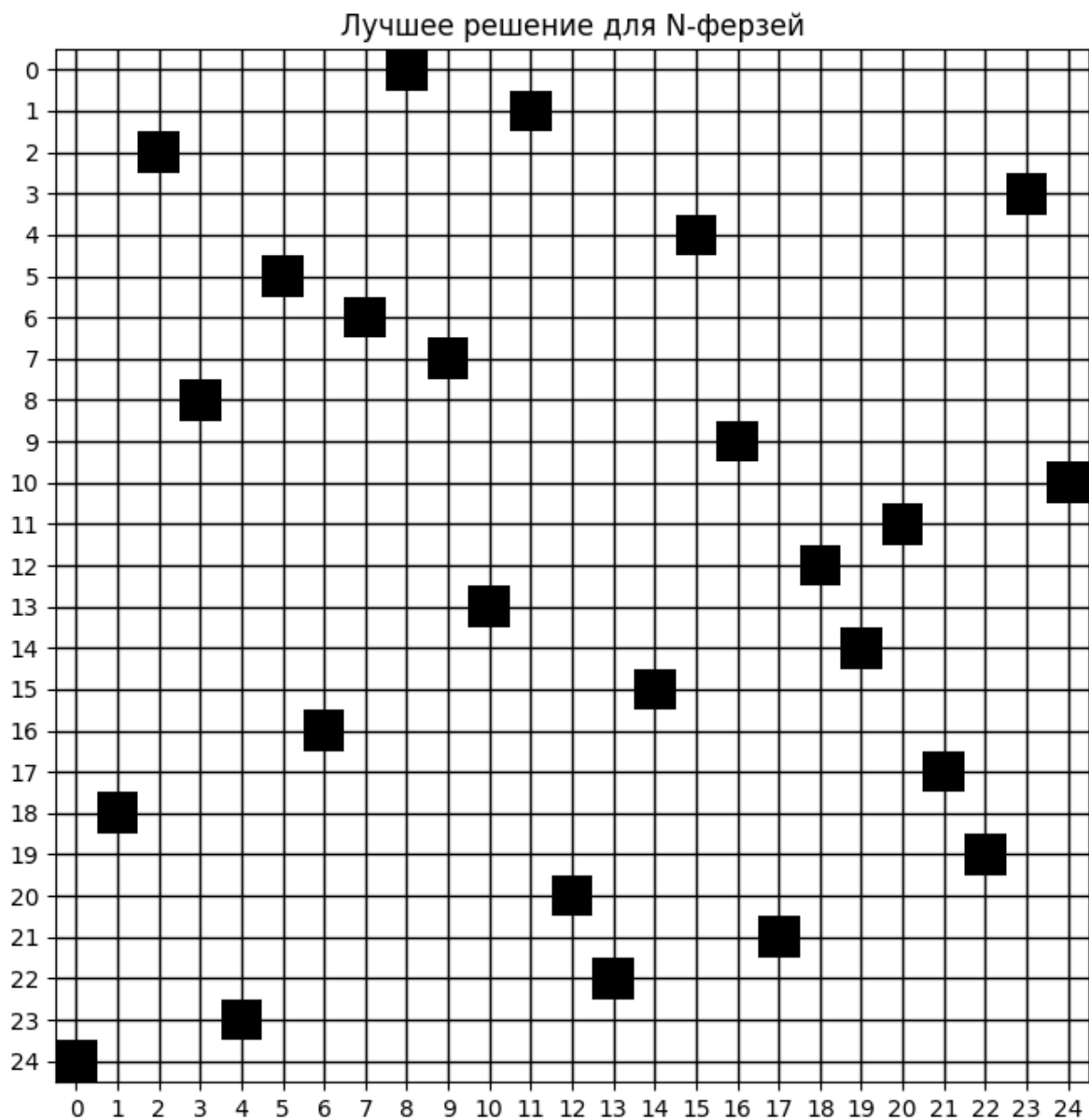
Лучшее решение 2. Видим что при сильно увеличенной температуре алгоритму тяжело быстро находить верные решения

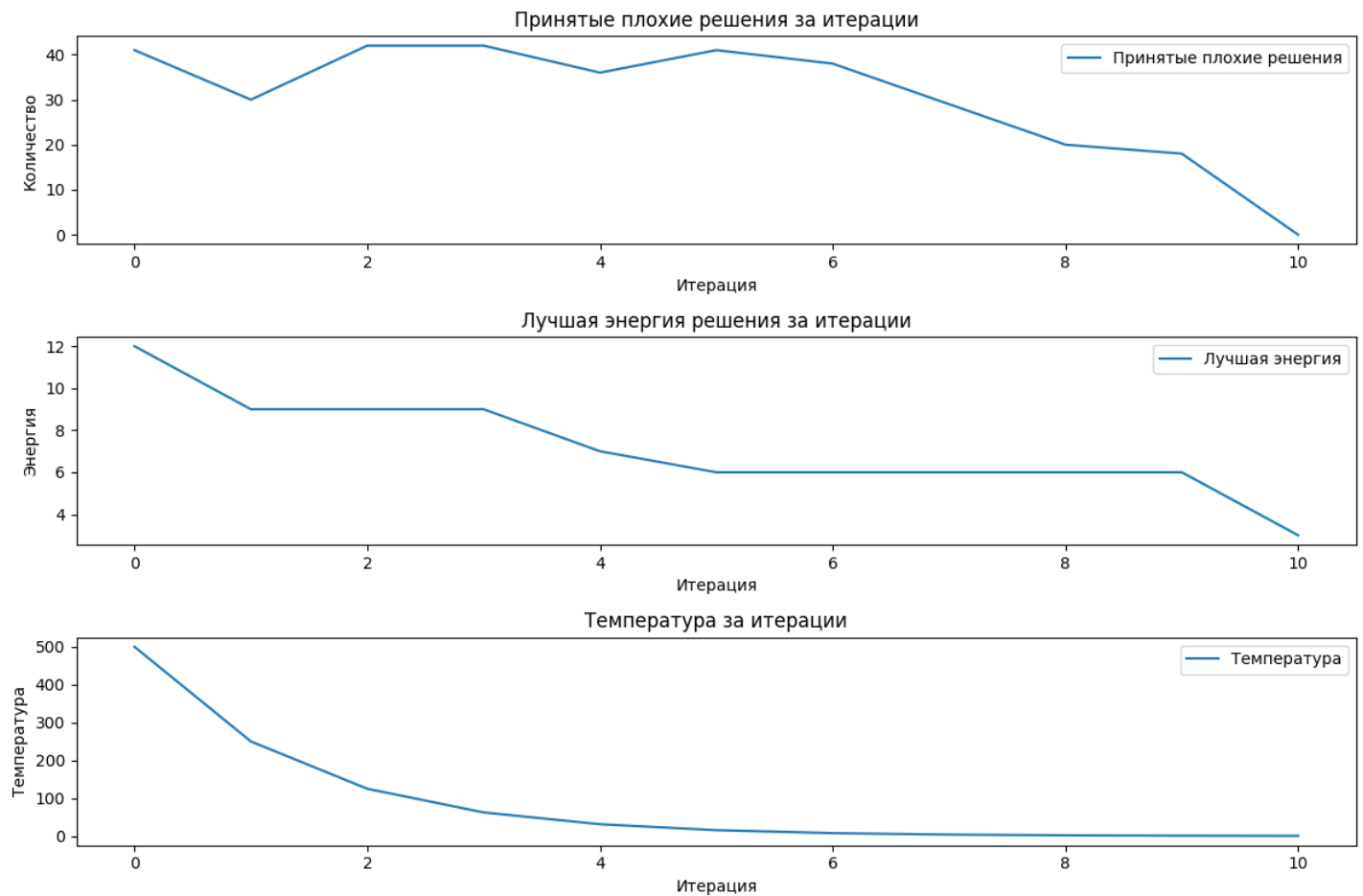
Тест 4

Уменьшим коэффициент охлаждения

Имитация отжига	
Количество ферзей ($N > 20$):	25
Начальная температура:	500.0
Конечная температура:	0.5
Коэффициент охлаждения (Alpha):	0.5
Шаги на температуру:	100
<input type="button" value="Запустить симуляцию"/>	

Частичный успех	
	Лучшая энергия: 3 (не ноль)
<input type="button" value="ОК"/>	





Лучшее решение 3. Быстрое снижение температуры позволило меньше просчитывать

Вывод: в рамках лабораторной работы мы создали и проанализировали алгоритм отжига для решения задачи числовой оптимизации. Этот метод эффективен в ситуациях, когда невозможно явно вычислить производную функции, полагаясь на подход Монте-Карло, где решение подбирается случайным образом. Однако такое ограничение снижает точность и сильно зависит от конкретного случая, а также может оказаться весьма медленным, требуя множества итераций. Кроме того, алгоритм отжига не всегда гарантирует нахождение решения.

Контрольные вопросы

1. Почему алгоритм отжига представляет собой процесс генерации случайных чисел?

Алгоритм отжига использует случайные числа на нескольких этапах: начальное решение генерируется как случайная перестановка, а в процессе поиска улучшений применяется модификация (например, обмен позиций ферзей) и критерий Метрополиса, где случайное число определяет принятие худшего решения с вероятностью $\exp(-\Delta E/T)$. Эта стохастическая природа позволяет исследовать пространство решений, избегая застревания в локальных минимумах, что делает его процессом генерации случайных чисел.

2. Какие причины обуславливают необходимость принимать в алгоритме отжига конечную температуру, отличную от нуля?

Ненулевая конечная температура сохраняет ненулевую вероятность принятия улучшений ($\exp(-\Delta E/T) > 0$ даже при малых ΔE), что предотвращает "заморозку" алгоритма в локальном минимуме. Если температура становится нулевой, вероятность принятия худших решений падает до нуля, что превращает процесс в детерминированный поиск, ограничивая способность находить глобальный оптимум.

3. Опишите функцию вероятности допуска и ее роль в алгоритме отжига?

Функция вероятности допуска определяется как $P(\Delta E) = \exp(-\Delta E/T)$, где ΔE — разница энергий между текущим и новым решением, T — текущая температура. Она определяет шанс принятия решения с ухудшением энергии: при высокой T вероятность велика, что позволяет исследовать пространство, а при низкой T — мала, сужая поиск к оптимальному решению. Это обеспечивает баланс между случайным блужданием и сходимостью к минимуму.

4. Укажите различия между понятиями начальное решение, текущее решение и рабочее решение, а также способы их представления в задаче о N-ферзях.

- Начальное решение: Случайная перестановка чисел $[0..N-1]$, представляющая позиции ферзей по строкам для каждого столбца (например, $[3, 1, 4, 2]$ для $N=4$). Это стартовая точка алгоритма.
- Текущее решение: Активное решение на данном этапе, обновляемое при принятии нового (например, после обмена позиций), также массив $[0..N-1]$.
- Рабочее решение: Временная копия текущего решения, модифицированная случайным образом (например, обмен двух позиций), используемая для проверки улучшения. Все три представлены как массивы длиной N , где индекс — столбец, значение — строка.

5. В чем смысл понятия энергии, и ее значений в алгоритме отжига?

Энергия в алгоритме — это количественная мера "качества" решения, в задаче N -ферзей представляющая число диагональных конфликтов между ферзями (пары, где $|\text{row}_i - \text{row}_j| = |\text{col}_i - \text{col}_j|$). Значение энергии 0 означает оптимальное решение (нет атак), а любое положительное значение (например, 3) указывает на количество оставшихся конфликтов, которые алгоритм стремится минимизировать.

6. Приведите комментарий к основным частям программы решения задачи о N -ферзях.

- Инициализация (`initialize_solution`): Создает случайный массив позиций ферзей, задавая начальную точку.
- Вычисление энергии (`compute_energy`): Подсчитывает конфликты по диагоналям через двойной цикл, определяя текущую "стоимость" решения.
- Модификация (`tweak_solution`): Выполняет обмен двух случайных позиций для генерации нового решения.
- Цикл по температуре (`while T > T_min`): Управляет охлаждением, вызывая внутренний цикл.
- Внутренний цикл (`for steps`): Выполняет заданное число попыток модификаций на текущей температуре.
- Критерий допуска (`if delta <= 0 or exp(-delta/T) > random`): Решает, принимать ли новое решение, обновляя текущее и лучшее.

- Визуализация (`visualize_board`, `plot_graphs`): Отображает доску и графики для анализа результата.

7. Охарактеризуйте основные этапы алгоритма отжига на примере решения классической задачи размещения N-ферзей.

- Инициализация: Генерируется случайное размещение, например, $[2, 0, 3, 1]$ для $N=4$, с энергией (например, 2 конфликта).
- Оценка энергии: Подсчитываются диагональные атаки (для $[2, 0, 3, 1]$ — ферзи в (0,2) и (2,3) конфликтуют).
- Случайная модификация: Обмен, например, 2 и 0 дает $[0, 2, 3, 1]$, новая энергия проверяется.
- Критерий допуска: Если новая энергия ниже или $\exp(-\Delta E/T) >$ случайное число (при $T=10$), решение принимается.
- Понижение температуры: T уменьшается ($T = 0.98 * T$), цикл повторяется до T_{\min} .
- Вывод: Лучшее решение (например, $[1, 3, 0, 2]$ с энергией 0) визуализируется.

8. Какими способами в зависимости от сложности решаемой проблемы производится оптимизация алгоритма отжига?

- Настройка параметров: Увеличение T_{\max} и шагов для сложных задач (большие N), снижение α для медленного охлаждения.
- Инициализация: Использование эвристик (например, частично корректное начальное размещение) вместо случайного.
- Параллелизация: Распределение итераций на несколько потоков для ускорения.
- Адаптивное охлаждение: Динамическая корректировка α в зависимости от сходимости энергии.
- Ограничение пространства: Исключение заведомо плохих решений для сокращения вычислений.

9. Укажите классы задач, в которых использование алгоритма отжига может быть эффективным?

- NP-трудные задачи: Например, коммивояжер, N-ферзи, где перебор всех решений невозможен.
- Оптимизация без производной: Сложные функции (например, дискретные или недифференцируемые).
- Комбинаторная оптимизация: Расписание, упаковка, где требуется минимизация конфликтов.
- Машинное обучение: Подбор гиперпараметров или кластеризация с нечеткими границами.
- Физические симуляции: Оптимизация кристаллических структур или молекулярного моделирования.

Код программы:

```
import random
import math
import matplotlib.pyplot as plt
import numpy as np
from tkinter import *
from tkinter import messagebox

def solve_n_queens_sa(N, initial_temp, final_temp, alpha, steps_per_change):
    def initialize_solution():
        sol = list(range(N))
        random.shuffle(sol)
        return sol

    def tweak_solution(sol):
        sol_copy = sol[:]
        x = random.randint(0, N-1)
        y = random.randint(0, N-1)
        while x == y:
            y = random.randint(0, N-1)
        sol_copy[x], sol_copy[y] = sol_copy[y], sol_copy[x]
        return sol_copy

    def compute_energy(sol):
        conflicts = 0
        for i in range(N):
            for j in range(i+1, N):
                if abs(sol[i] - sol[j]) == abs(i - j):
                    conflicts += 1
        return conflicts

    current_sol = initialize_solution()
    current_energy = compute_energy(current_sol)
    best_sol = current_sol[:]
    best_energy = current_energy

    temps = []
    best_energies = []
    accepted_bads_list = []

    temp = initial_temp
    iteration = 0
    while temp > final_temp:
        temps.append(temp)
        best_energies.append(best_energy)

        accepted_bad_this = 0
        for _ in range(steps_per_change):
            working_sol = tweak_solution(current_sol)
            working_energy = compute_energy(working_sol)
            delta = working_energy - current_energy
            if delta <= 0 or math.exp(-delta / temp) > random.random():
                current_sol = working_sol[:]
                current_energy = working_energy
                if delta > 0:
                    accepted_bad_this += 1
                if current_energy < best_energy:
                    best_sol = current_sol[:]
                    best_energy = current_energy

        accepted_bads_list.append(accepted_bad_this)
        temp *= alpha
        iteration += 1

    # Append final state
```

```

temps.append(temp)
best_energies.append(best_energy)
accepted_bads_list.append(0) # No accepts at final

return best_sol, best_energy, temps, best_energies, accepted_bads_list

def visualize_board(sol, N):
    board = np.zeros((N, N))
    for col in range(N):
        row = sol[col]
        board[row, col] = 1
    plt.figure(figsize=(8, 8))
    plt.imshow(board, cmap='binary')
    plt.title('Лучшее решение для N-ферзей')
    plt.xticks(range(N))
    plt.yticks(range(N))
    plt.grid(True, color='black', linewidth=1)
    plt.show()

def plot_graphs(temps, best_energies, accepted_bads):
    iterations = list(range(len(temps)))

    plt.figure(figsize=(12, 8))

    plt.subplot(3, 1, 1)
    plt.plot(iterations, accepted_bads, label='Принятые плохие решения')
    plt.xlabel('Итерация')
    plt.ylabel('Количество')
    plt.title('Принятые плохие решения за итерации')
    plt.legend()

    plt.subplot(3, 1, 2)
    plt.plot(iterations, best_energies, label='Лучшая энергия')
    plt.xlabel('Итерация')
    plt.ylabel('Энергия')
    plt.title('Лучшая энергия решения за итерации')
    plt.legend()

    plt.subplot(3, 1, 3)
    plt.plot(iterations, temps, label='Температура')
    plt.xlabel('Итерация')
    plt.ylabel('Температура')
    plt.title('Температура за итерации')
    plt.legend()

    plt.tight_layout()
    plt.show()

def run_simulation():
    try:
        N = int(entry_n.get())
        initial_temp = float(entry_initial_temp.get())
        final_temp = float(entry_final_temp.get())
        alpha = float(entry_alpha.get())
        steps = int(entry_steps.get())

        if N <= 20:
            raise ValueError("N должно быть больше 20")
        if alpha >= 1 or alpha <= 0:
            raise ValueError("Alpha должно быть между 0 и 1")

        best_sol, best_energy, temps, best_energies, accepted_bads = solve_n_queens_sa(
            N, initial_temp, final_temp, alpha, steps
        )

        if best_energy == 0:
            messagebox.showinfo("Успех", f"Решение найдено! Энергия: {best_energy}")

```



```
    else:
        messagebox.showwarning("Частичный успех", f"Лучшая энергия: {best_energy} (не ноль)")

    visualize_board(best_sol, N)
    plot_graphs(temps, best_energies, accepted_bads)

except ValueError as e:
    messagebox.showerror("Ошибка ввода", str(e))

# GUI Setup
root = Tk()
root.title("Имитация отжига")

Label(root, text="Количество ферзей (N > 20):").grid(row=0, column=0)
entry_n = Entry(root)
entry_n.grid(row=0, column=1)
entry_n.insert(0, "25")

Label(root, text="Начальная температура:").grid(row=1, column=0)
entry_initial_temp = Entry(root)
entry_initial_temp.grid(row=1, column=1)
entry_initial_temp.insert(0, "30.0")

Label(root, text="Конечная температура:").grid(row=2, column=0)
entry_final_temp = Entry(root)
entry_final_temp.grid(row=2, column=1)
entry_final_temp.insert(0, "0.5")

Label(root, text="Коэффициент охлаждения (Alpha):").grid(row=3, column=0)
entry_alpha = Entry(root)
entry_alpha.grid(row=3, column=1)
entry_alpha.insert(0, "0.98")

Label(root, text="Шаги на температуру:").grid(row=4, column=0)
entry_steps = Entry(root)
entry_steps.grid(row=4, column=1)
entry_steps.insert(0, "100")

Button(root, text="Запустить симуляцию", command=run_simulation).grid(row=5, column=0,
columnspan=2)

root.mainloop()
```