

**Федеральное государственное бюджетное образовательное
учреждение высшего образования "Белгородский государственный
технологический университет им. В.Г. Шухова"**

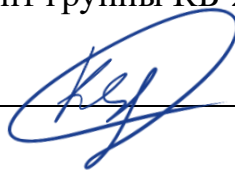
Кафедра программного обеспечения вычислительной техники и
автоматизированных систем.

Лабораторная работа №1

Алгоритм отжига.

Выполнил:

Студент группы КБ-211



Коренев Д.Н.

Принял:

Твердохлеб В.В.

Оглавление

Задание	3
Реализация ПО.....	3
Вывод.....	17
Приложения	19

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

Задание

1. Изучить теоретическую часть лекционного материала по теме. Ответить на контрольные вопросы.

2. Используя пример программного кода (исходные файлы в приложении к заданию), а также любой высокоуровневый язык программирования и соответствующую среду, разработать программное обеспечение с графическим интерфейсом, позволяющее решить задачу N-ферзей для $N > 20$. Интерфейс ПО должен обеспечивать ввод следующих параметров алгоритма:

- Максимальная температура
- Минимальная температура
- Коэффициент понижения температуры
- Количество ферзей.
- Количество шагов при постоянном значении температуры.

Необходимо предусмотреть визуализацию лучшего решения в виде шахматной доски, а также необходимо построить график изменения принятых плохих решений, энергии лучшего решения, и температуры.

Реализация ПО

Реализация алгоритма отжига:

```
Python
class MemberType:
    def __init__(self):
        self.solution = list(range(data_cb["num_queens"]))
        self.energy = 0.0

    def get_rand():
        return random.randint(0, data_cb["num_queens"] - 1)

    def get_srand():
        return random.random()

    def tweak_solution(member):
        x, y = random.sample(range(data_cb["num_queens"]), 2)
        member.solution[x], member.solution[y] \
            = member.solution[y], member.solution[x]

    def initialize_solution(member):
        random.shuffle(member.solution)
        tweak_solution(member)
```

```

def emit_solution(member):
    board = [['.' for _ in range(data_cb["num_queens"])]
              for _ in range(data_cb["num_queens"])]
    for x in range(data_cb["num_queens"]):
        board[x][member.solution[x]] = 'Q'
    for row in board:
        print(' '.join(row))
    print("\n")

def compute_energy(member):
    conflicts = 0
    board = [['.' for _ in range(data_cb["num_queens"])]
              for _ in range(data_cb["num_queens"])]
    for i in range(data_cb["num_queens"]):
        board[i][member.solution[i]] = 'Q'
    dx = [-1, 1, -1, 1]
    dy = [-1, 1, 1, -1]
    for i in range(data_cb["num_queens"]):
        x, y = i, member.solution[i]
        for j in range(4):
            temp_x, temp_y = x, y
            while True:
                temp_x += dx[j]
                temp_y += dy[j]
                if temp_x < 0 or temp_x >= data_cb["num_queens"] \
                   or temp_y < 0 or temp_y >= data_cb["num_queens"]:
                    break
                if board[temp_x][temp_y] == 'Q':
                    conflicts += 1
    member.energy = float(conflicts)

def copy_solution(dest, src):
    dest.solution = src.solution[:]
    dest.energy = src.energy

```

Полный код программы см. в приложении 1.

Внешний вид программы:

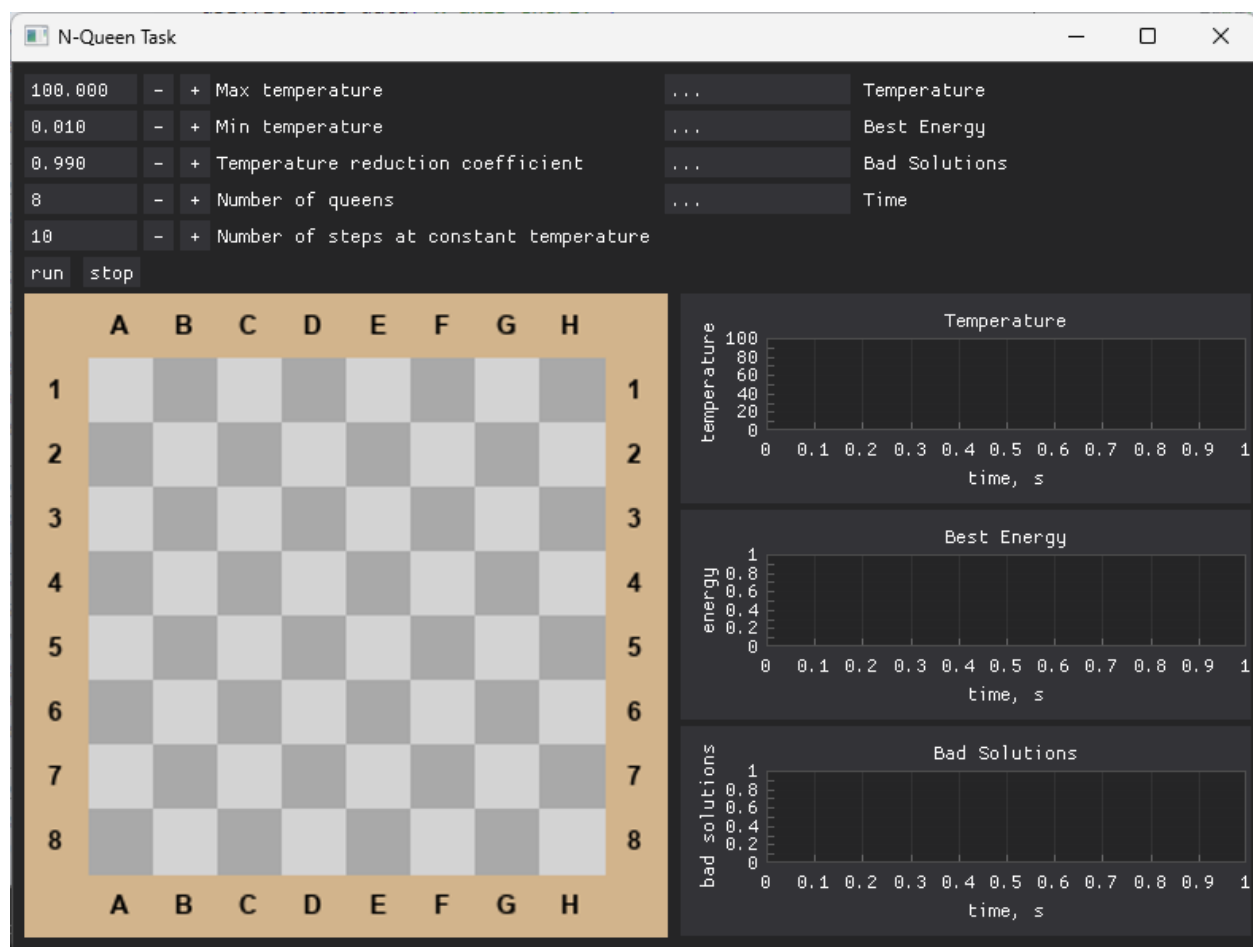


Рисунок 1. Внешний вид приложения

3. Исследовать влияние параметров алгоритма на качество его работы, провести не менее 20 экспериментов при разных комбинациях параметров.

Будем проводить тестирование при значениях: максимальная температура = 100, минимальная температура = 0.01, коэф. уменьшения температуры = 0.99, количество ферзей = 8, количество шагов = 10, если не указано иное значение.

- Максимальная температура
 - 0.1:

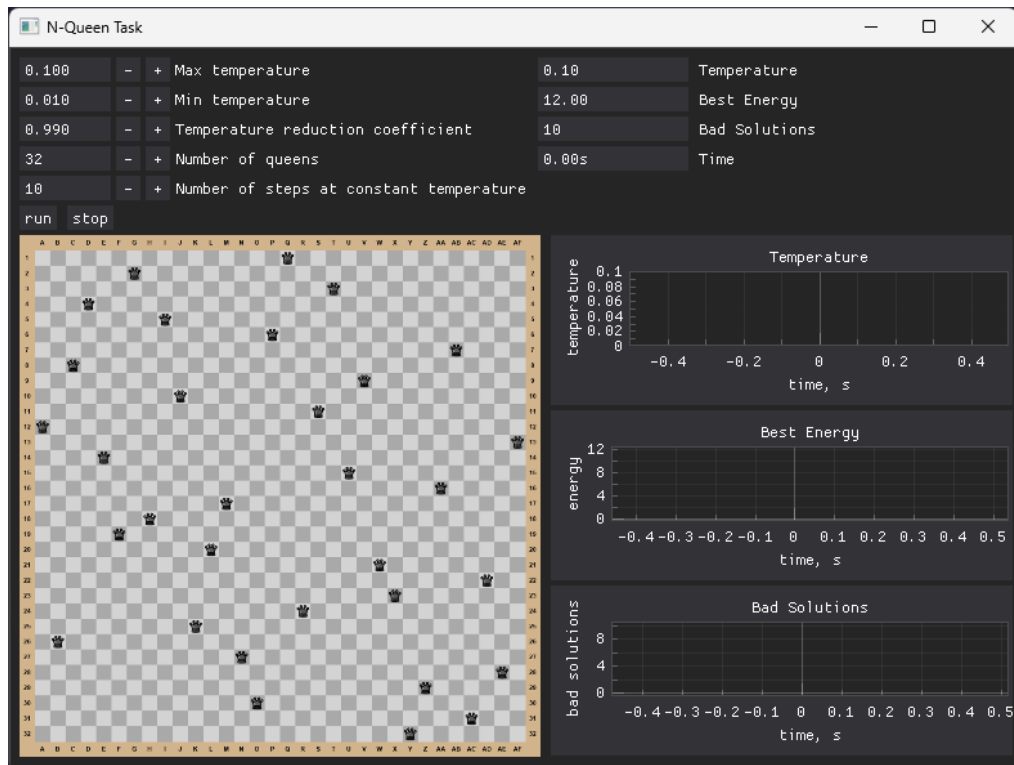


Рисунок 2. При температуре 0.1 алгоритм не достигает оптимальных значений.

- 1:

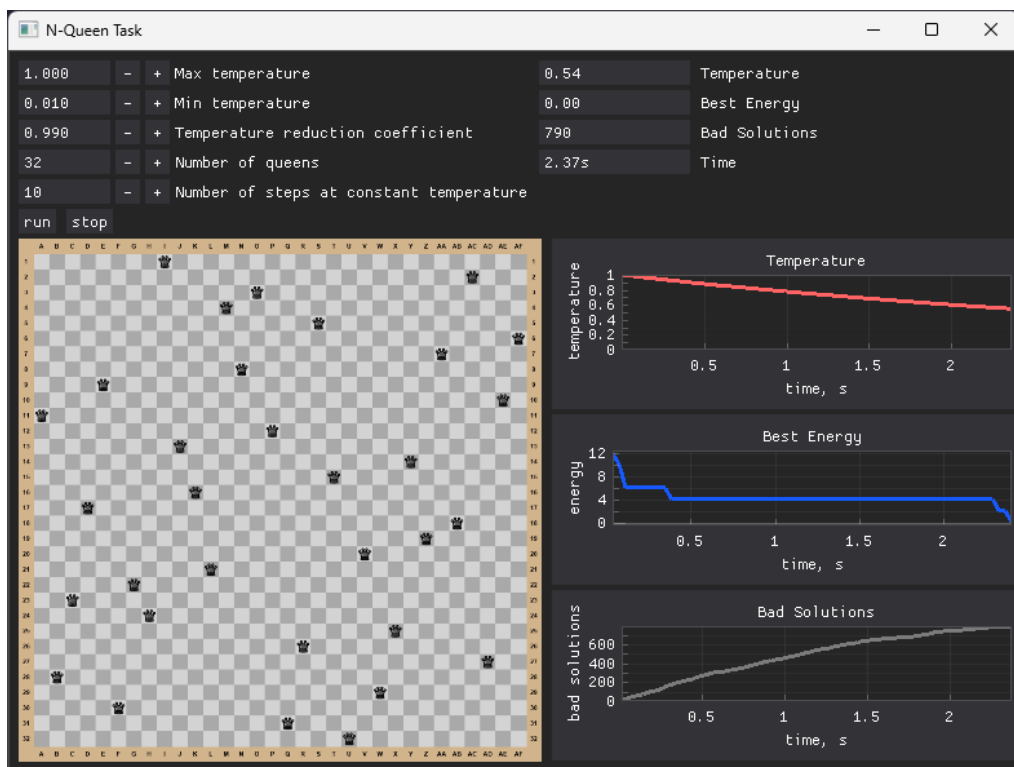


Рисунок 3. Для данных параметров значение максимальной температуры 1 является оптимальным.

○ 25:

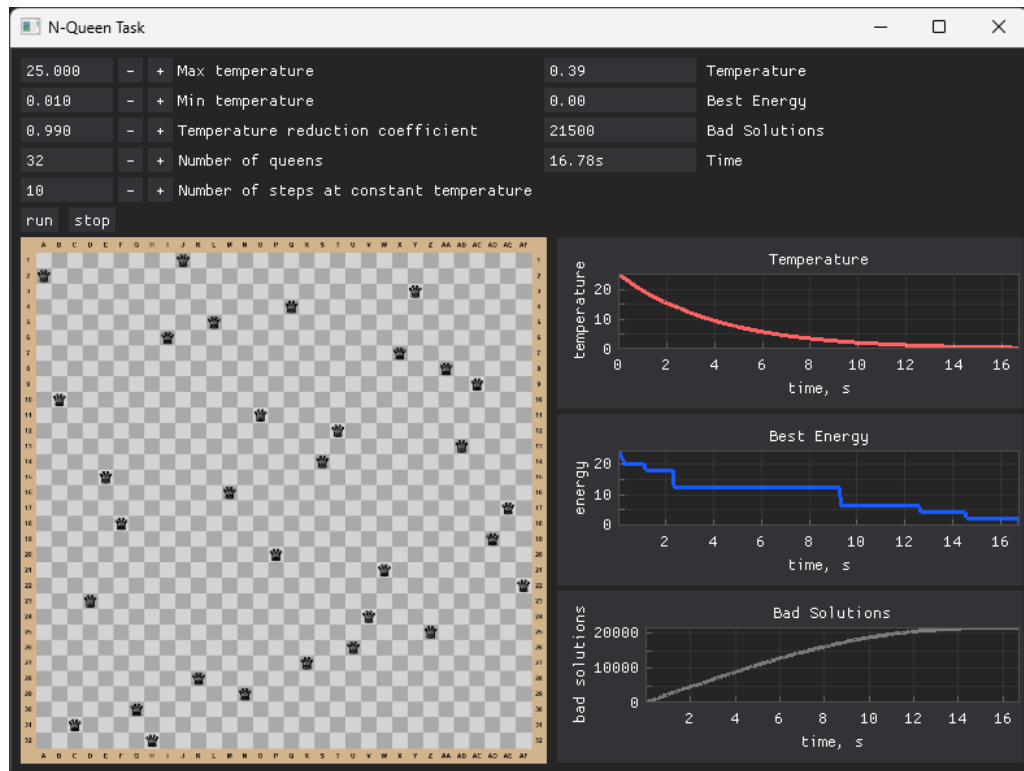


Рисунок 4. При увеличении максимальной температуры, увеличивается затраченное время для вычисления.

○ 100:

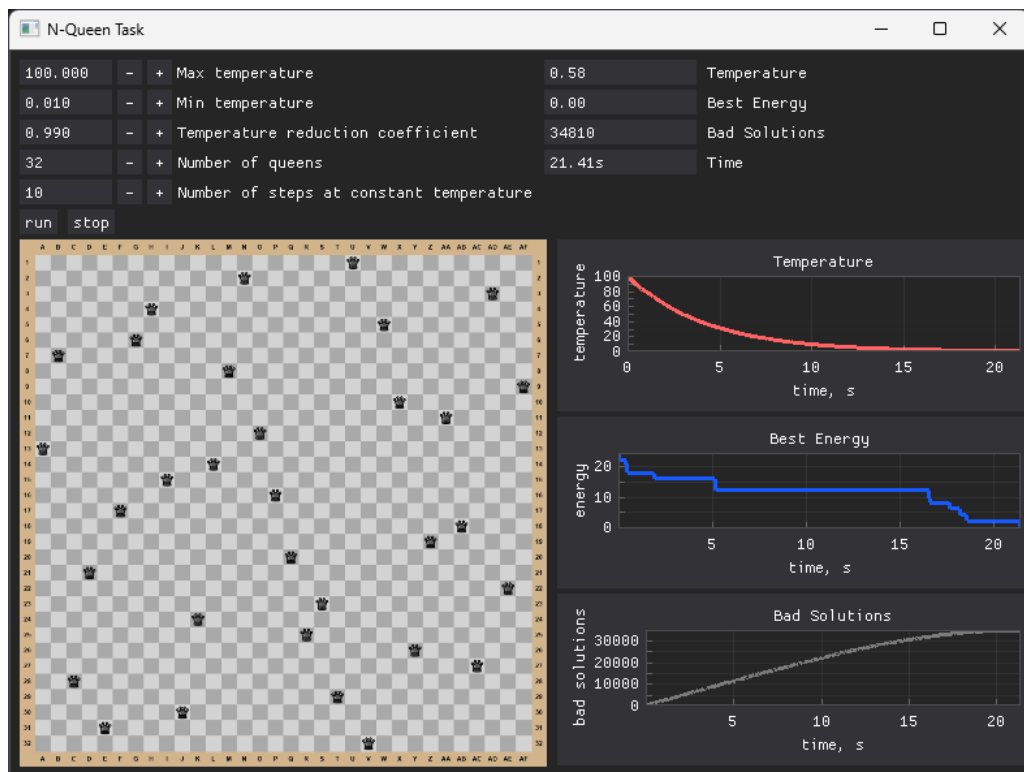


Рисунок 5. При увеличении максимальной температуры, увеличивается затраченное время для вычисления.

- Минимальная температура
 - 0:

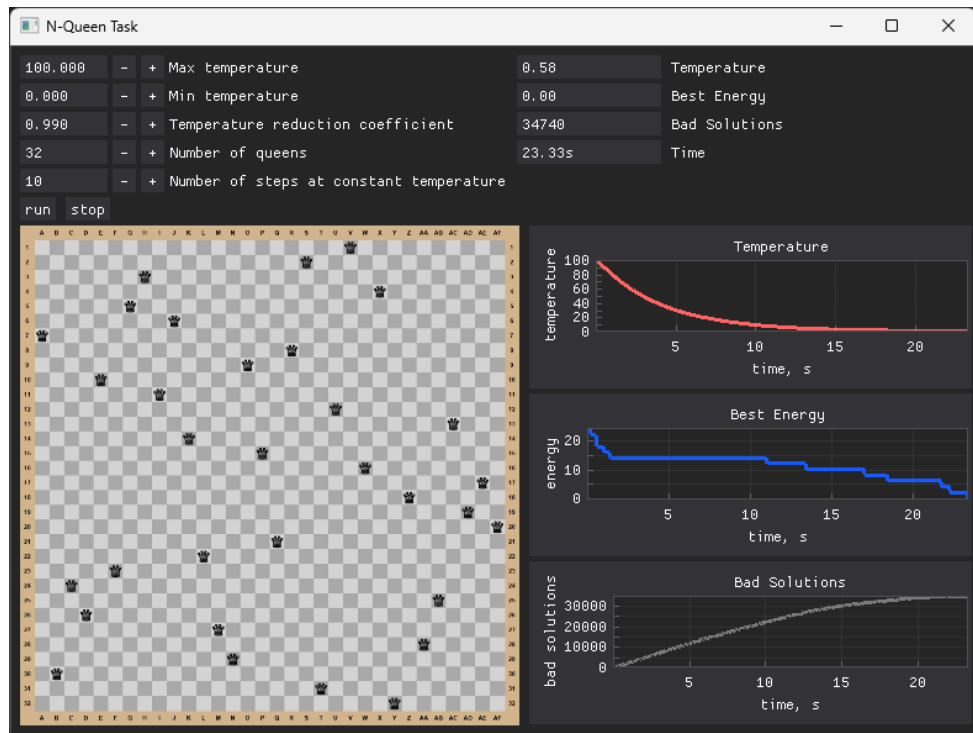


Рисунок 6. При значении минимальной температуры 0 наблюдается нормальная работа алгоритма.

- 10:

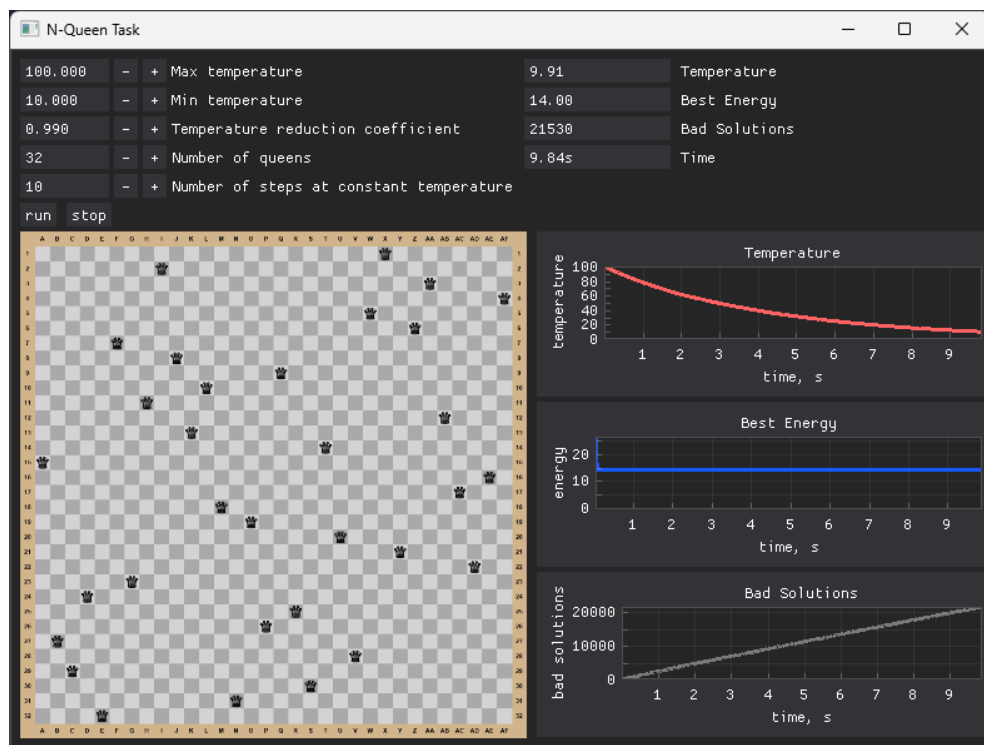


Рисунок 7. При значении минимальной температуры 10 алгоритм не успевает найти оптимальное решение.

○ 50:

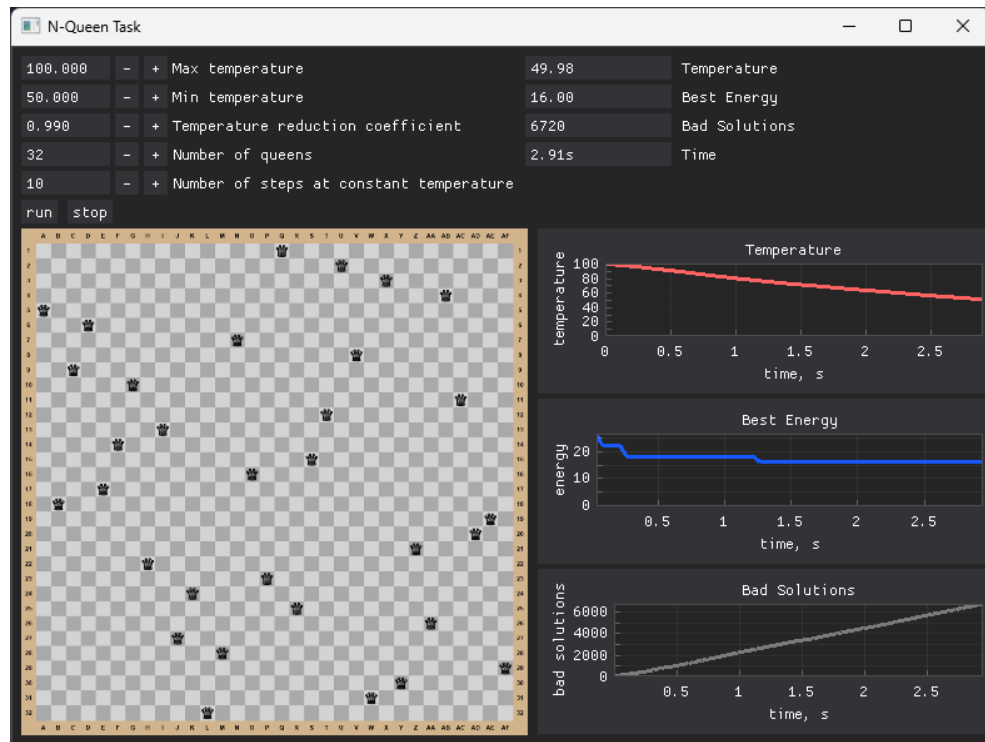


Рисунок 8. При значении минимальной температуры 50 алгоритм не успевает найти оптимальное решение.

○ 100:

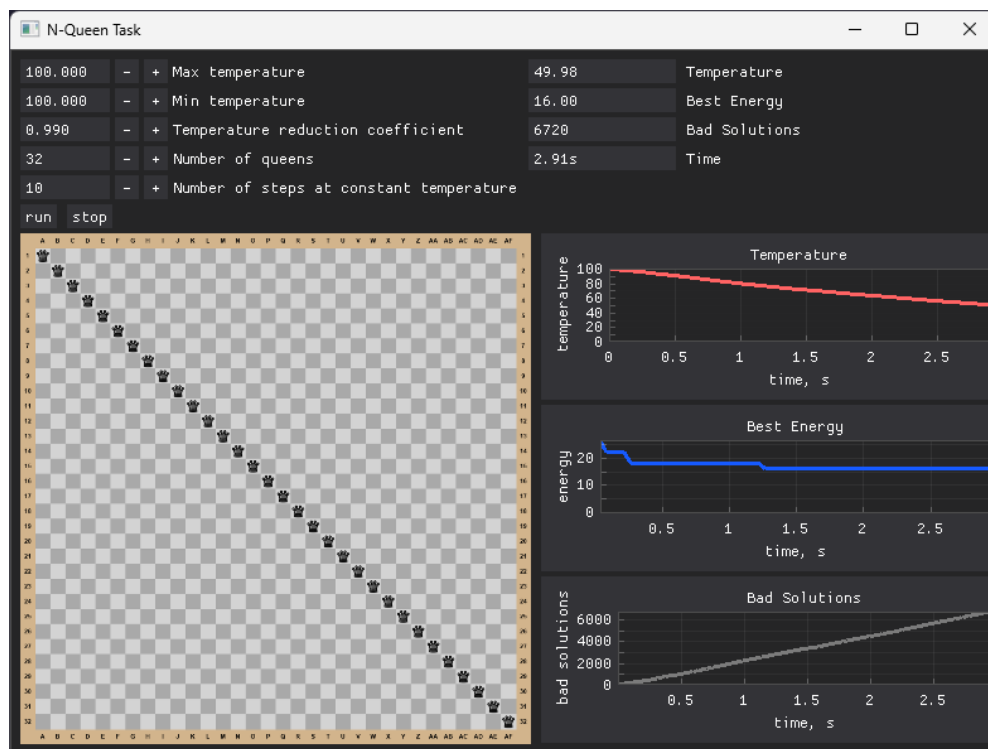


Рисунок 9. При значении минимальной температуры 100 алгоритм не успевает найти оптимальное решение, не выполняется не одной итерации.

- Коэффициент понижения температуры
 - -1:

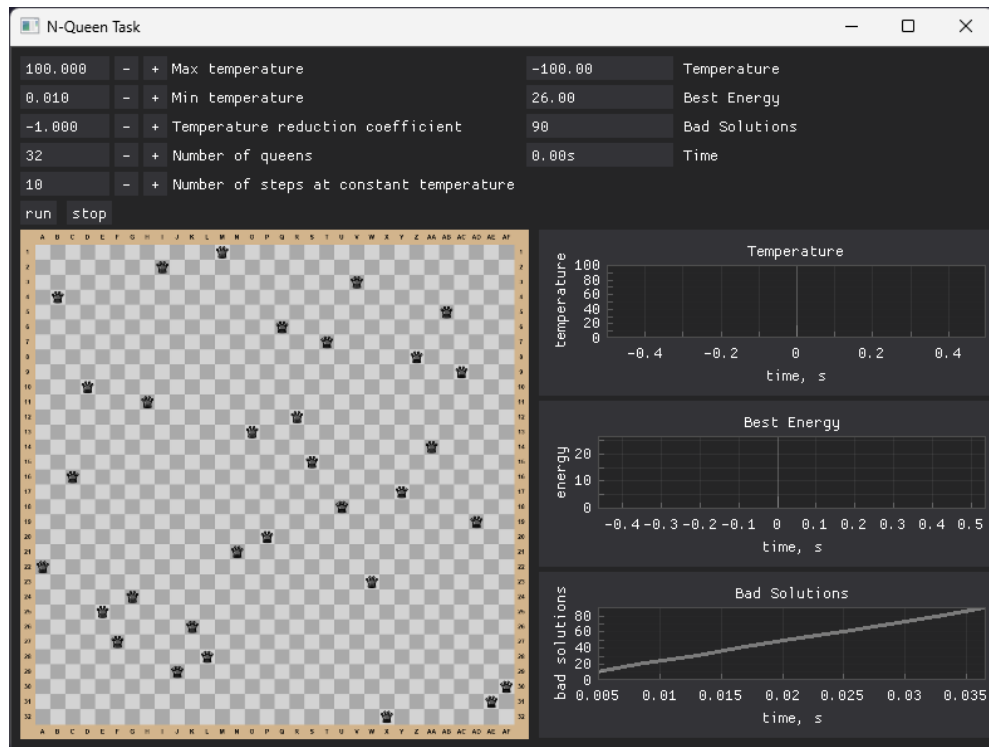


Рисунок 10. При значении коэффициента понижения температуры -1 выполняется одна итерация алгоритма.

- 0:

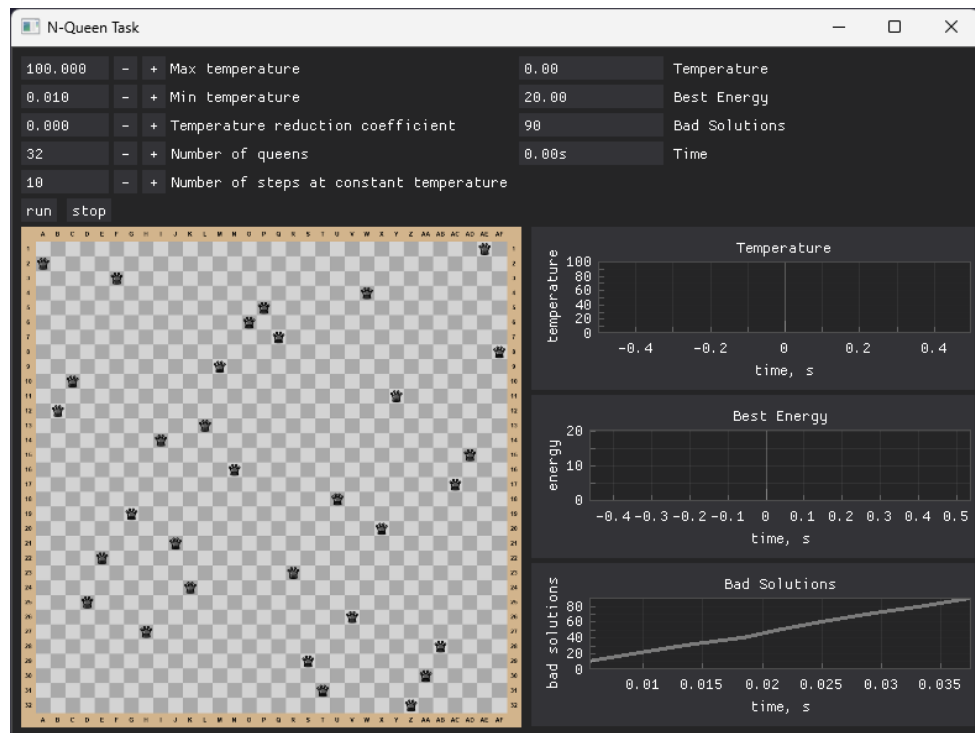


Рисунок 11. При значении коэффициента понижения температуры 0 выполняется одна итерация алгоритма.

○ 0.2:

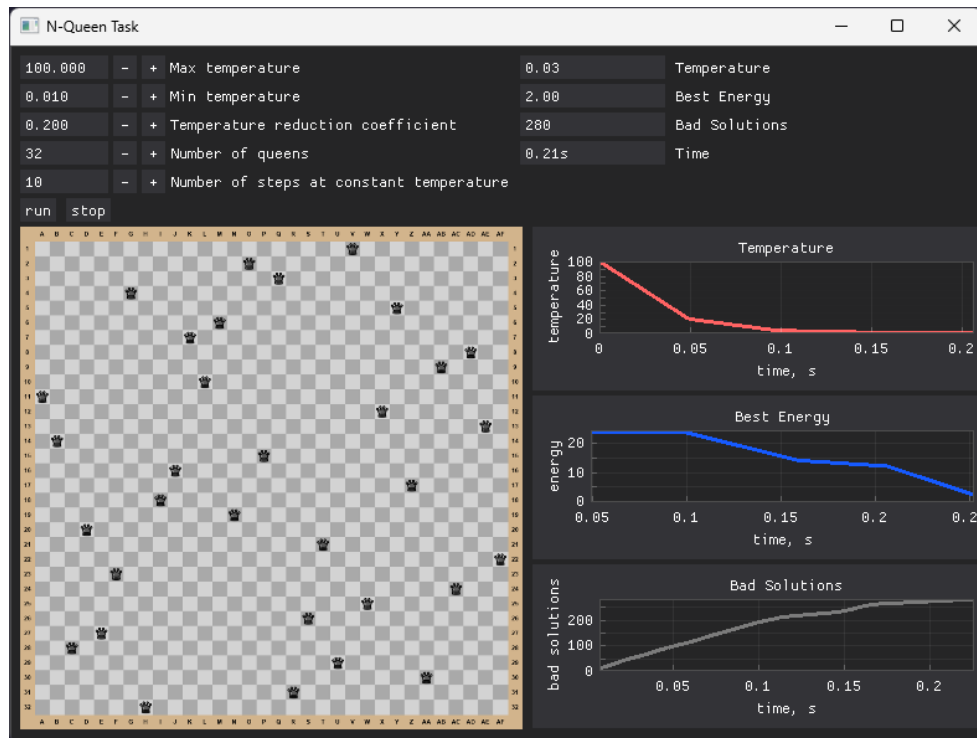


Рисунок 12. При значении коэффициента понижения температуры 0.2 алгоритм не успевает найти оптимальное решение

○ 0.99:

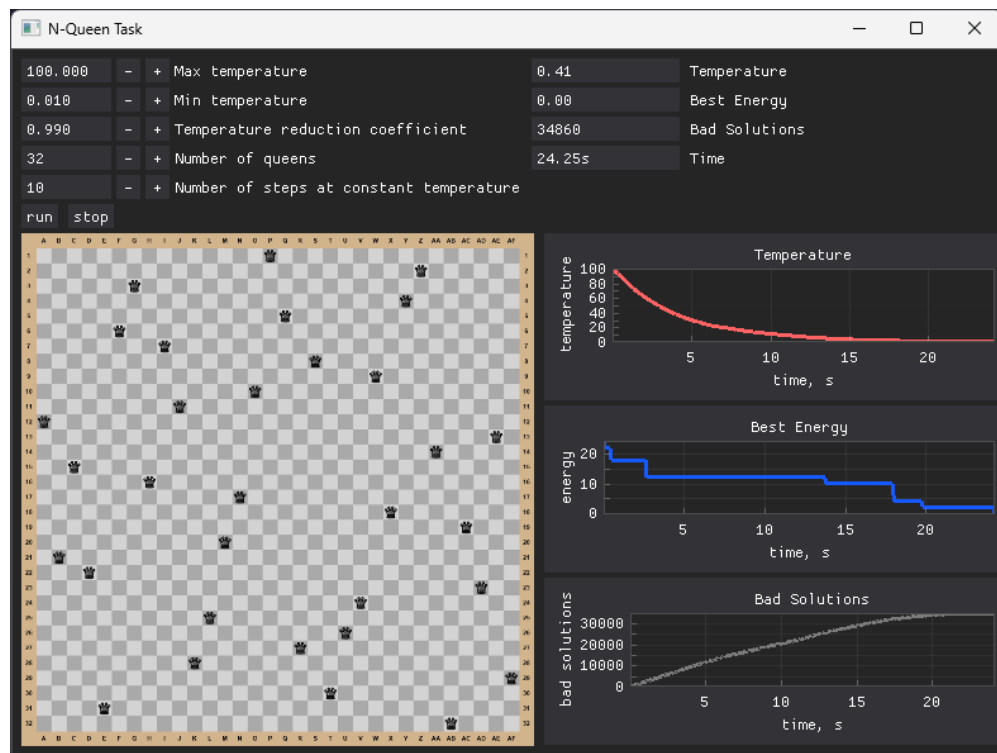


Рисунок 13. При значении коэффициента понижения температуры 0.99 наблюдается нормальная работа алгоритма.

○ 2:

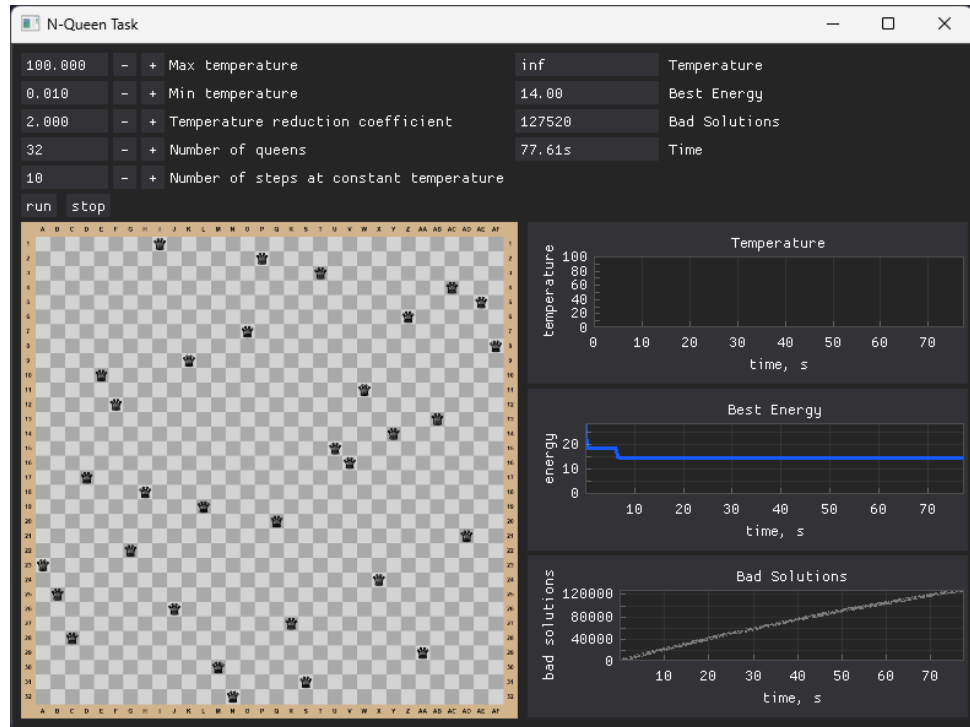


Рисунок 14. При значении коэффициента понижения температуры 2 алгоритм не смог найти оптимальное решение за отведенное время.

- Количество ферзей

○ 8:

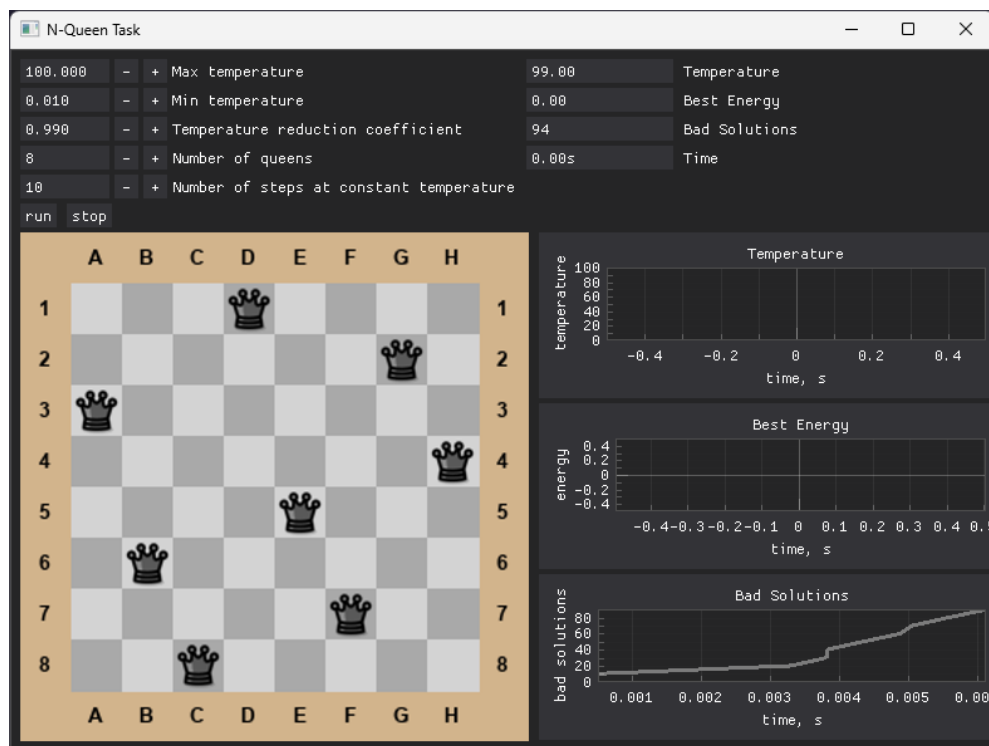


Рисунок 15. При значении количества ферзей 8 алгоритм находит оптимальное решение мгновенно.

○ 20:

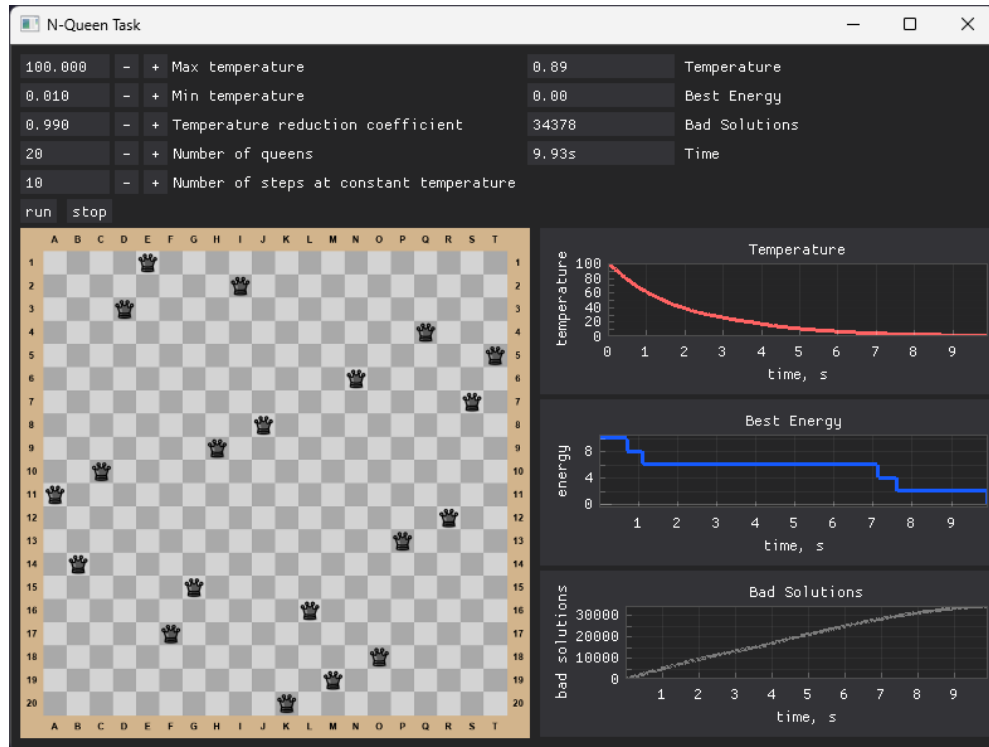


Рисунок 16. При значении количества ферзей 20 алгоритм находит оптимальное решение за 10 с.

○ 30:

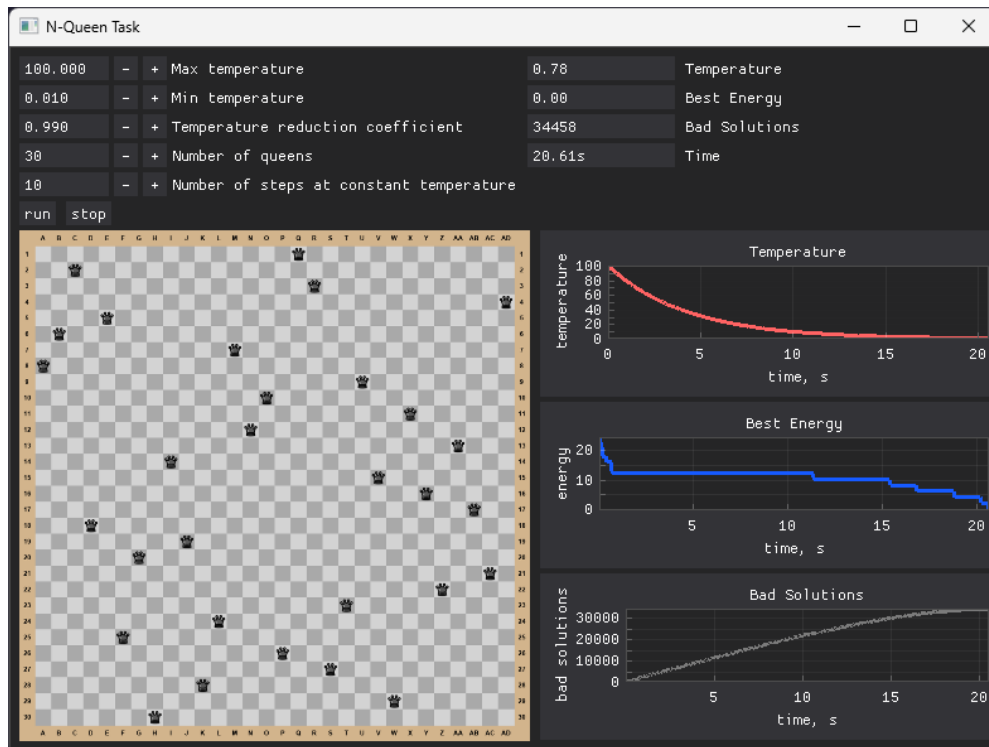


Рисунок 17. При значении количества ферзей 30 алгоритм находит оптимальное решение за 20 с.

○ 50:

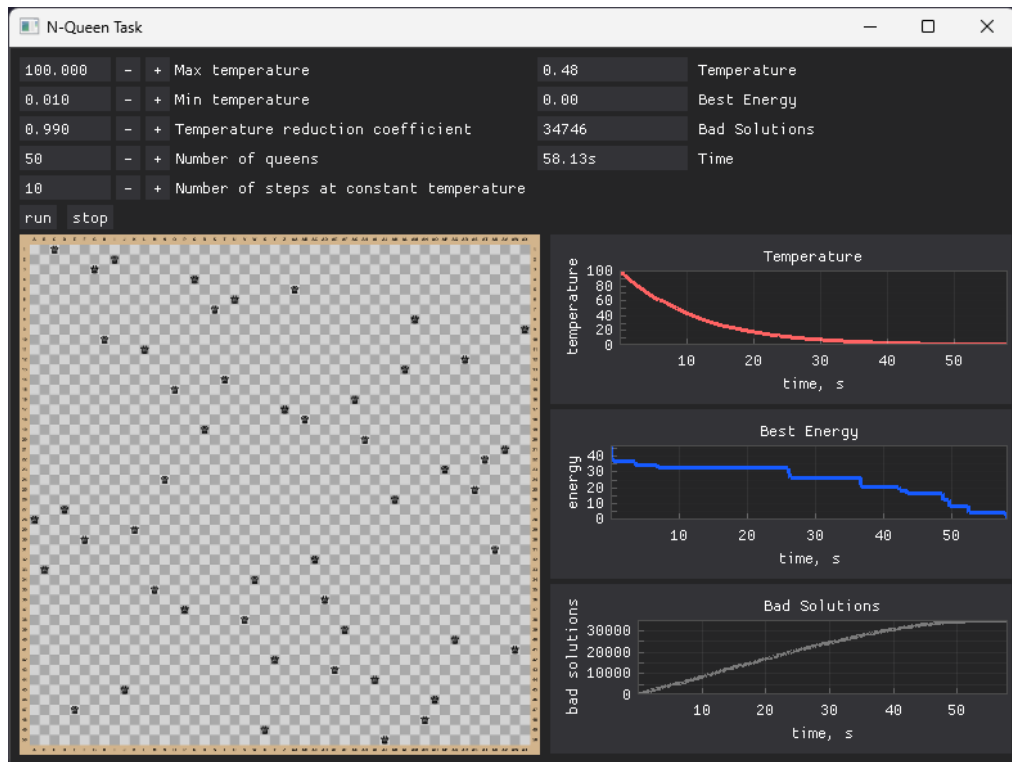


Рисунок 18. При значении количества ферзей 50 алгоритм находит оптимальное решение за 60 с.

○ 100:

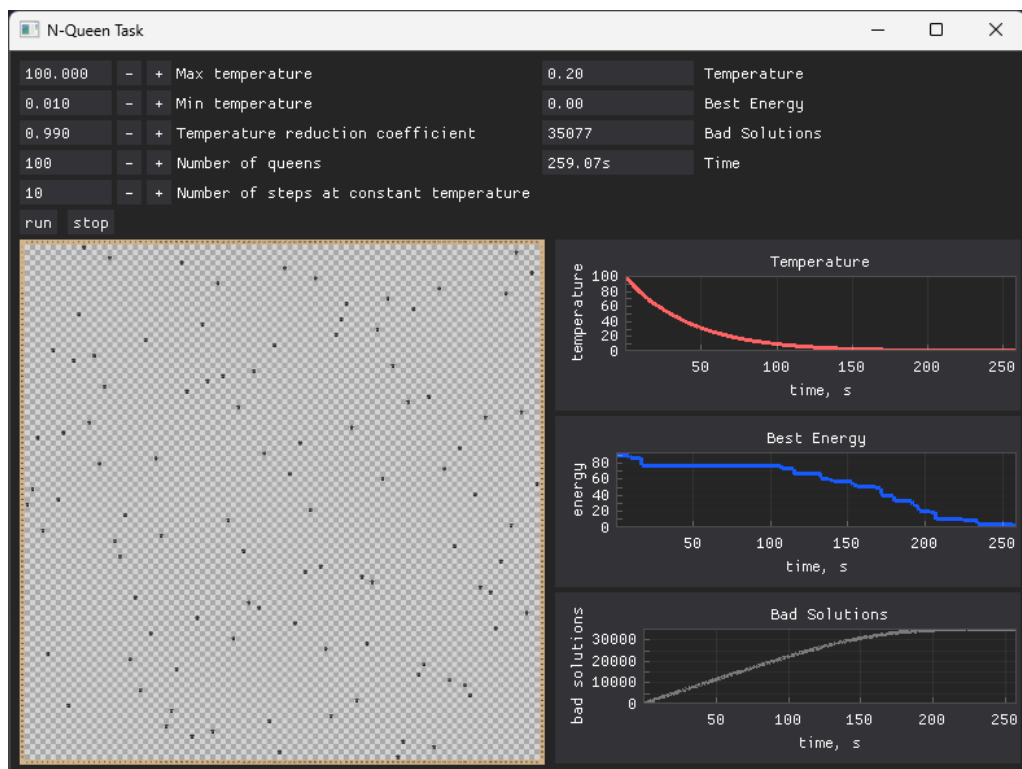


Рисунок 19. При значении количества ферзей 100 алгоритм находит оптимальное решение за 260 с.

- Количество шагов при постоянном значении температуры
 - 1:

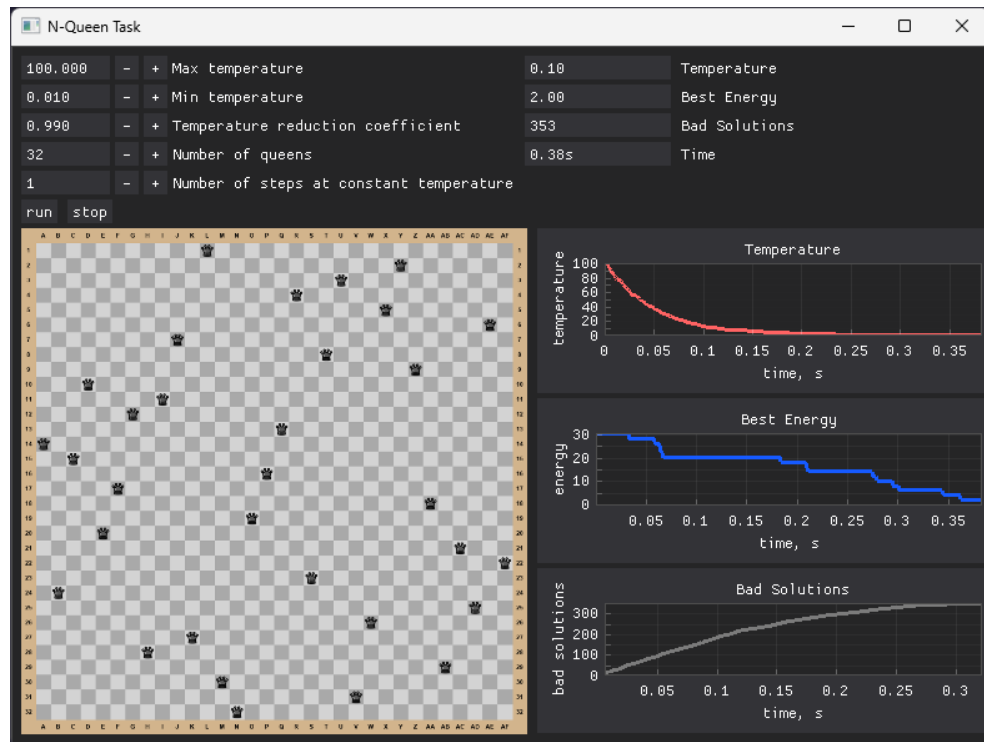


Рисунок 20. При значении количества шагов 1 алгоритм не смог найти оптимальное решение.

- 5:

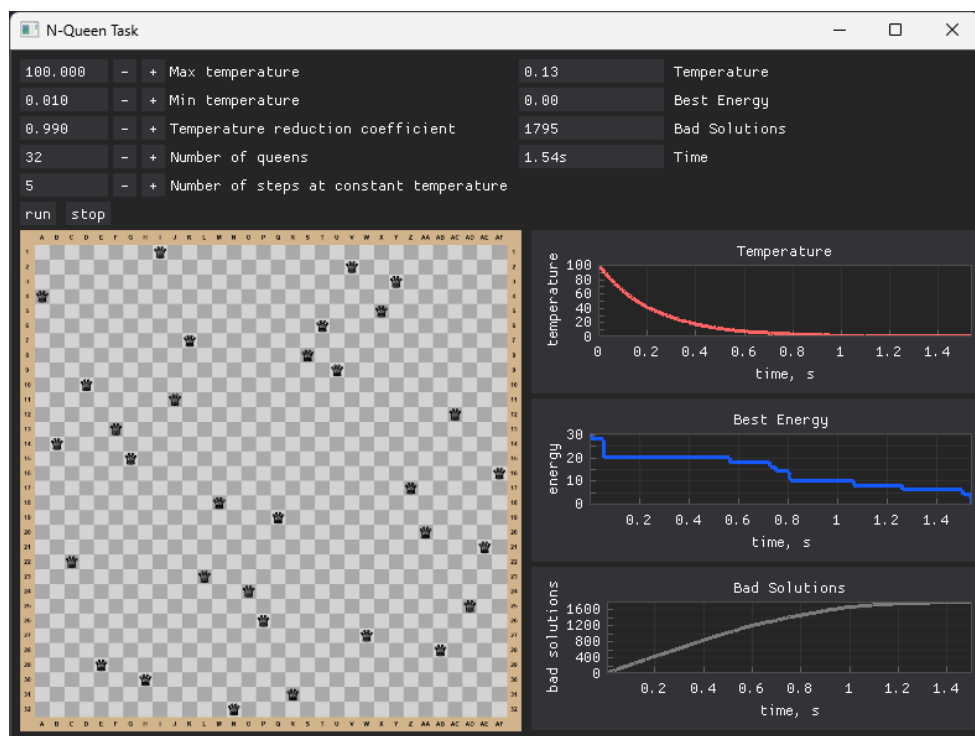


Рисунок 21. При значении количества шагов 5 алгоритм находит оптимальное решение за 1,5 с.

○ 10:

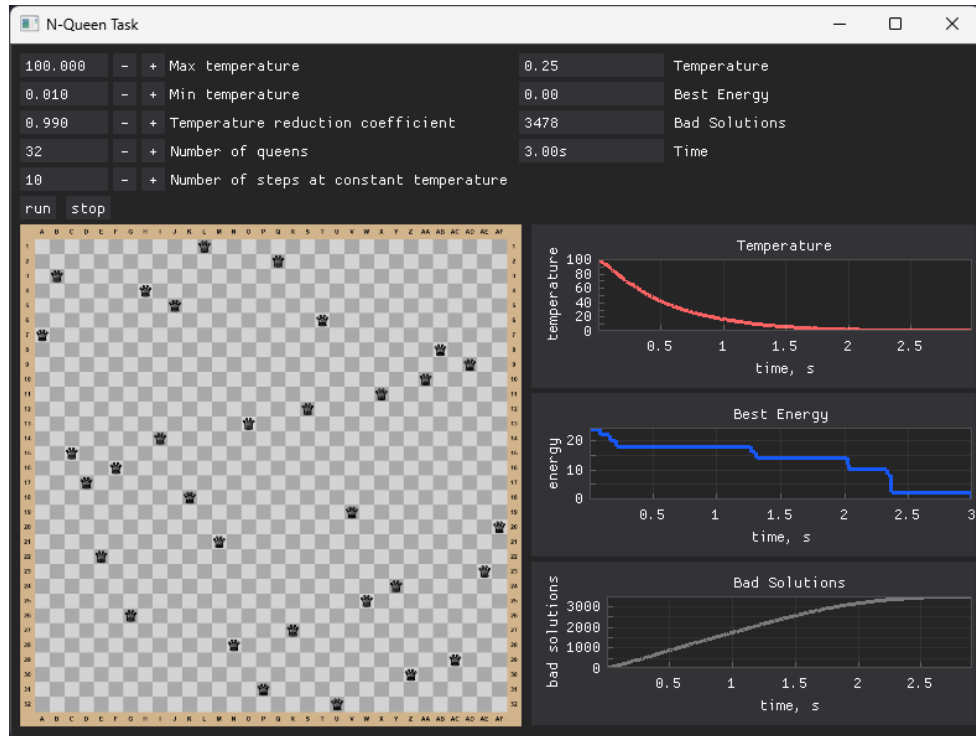


Рисунок 22. При значении количества шагов 10 алгоритм находит оптимальное решение за 3 с.

○ 50:

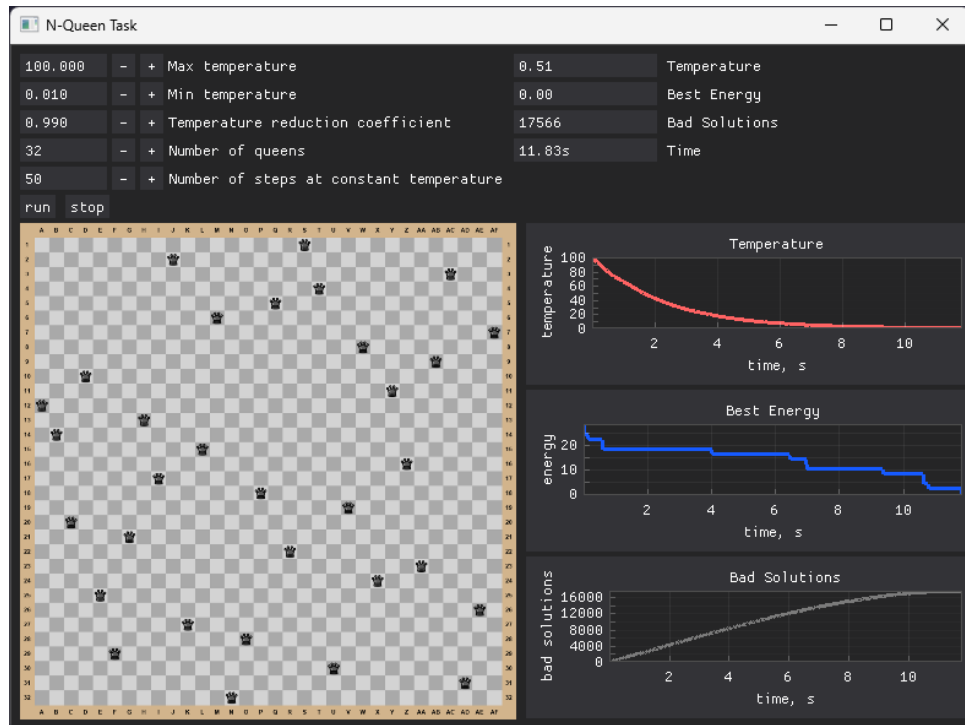


Рисунок 23. При значении количества шагов 50 алгоритм находит оптимальное решение за 12 с.

Вывод

В ходе проведенного исследования было установлено, что на работу алгоритма отжига в значительной степени влияют такие параметры, как максимальная и минимальная температура, коэффициент понижения температуры, количество шагов и количество ферзей.

1. Максимальная температура:

- При низких значениях максимальной температуры, например, 0.1, алгоритм не достигает оптимальных значений.
- Оптимальная работа наблюдается при максимальной температуре 1.
- Увеличение максимальной температуры (25 и 100) приводит к увеличению времени, затраченного на вычисления, хотя алгоритм все же находит решение.

2. Минимальная температура:

- Нормальная работа алгоритма наблюдается при минимальной температуре 0.
- При повышенных значениях (10, 50, 100) алгоритм не успевает найти оптимальное решение, так как температура падает слишком быстро.

3. Коэффициент понижения температуры:

- При отрицательных значениях коэффициента (-1 и 0) алгоритм выполняет всего одну итерацию и не может найти решение.
- При коэффициенте 0.99 наблюдается нормальная работа, что позволяет алгоритму медленно снижать температуру и достичь оптимального решения.
- Значения, превышающие 1 (например, 2), приводят к слишком быстрому охлаждению, что не позволяет алгоритму найти решение за отведенное время.

4. Количество ферзей:

- С увеличением количества ферзей время, необходимое для нахождения решения, возрастает. Однако алгоритм справляется с

нахождением оптимального решения для всех протестированных значений, от 8 до 100 ферзей.

- При 100 ферзях алгоритм находит решение за 260 секунд, что указывает на значительное увеличение времени при росте сложности задачи.

5. Количество шагов при постоянной температуре:

- При малом количестве шагов (1) алгоритм не находит решение.
- С увеличением количества шагов алгоритм находит решение быстрее. Например, при 50 шагах решение находится за 12 секунд.

Таким образом, для эффективной работы алгоритма отжига необходимо тщательно подбирать значения параметров. Оптимальная комбинация параметров зависит от размера задачи, но в общем случае наиболее удачными значениями для данной задачи являются: максимальная температура 1, минимальная температура 0, коэффициент понижения температуры 0.99.

Приложения

Приложение 1. Полный код программы.

```
Python 3.12
from PIL import Image, ImageDraw, ImageFont
import dearpygui.dearpygui as dpg

import random
import math
import time
import threading

process = None
is_running = False

# Data for algorithm
data_cb = {
    'num_queens': 8,
    'init_temp': 100.0,
    'fin_temp': 0.1,
    'alpha': 0.99,
    'steps_per_change': 100
}

# Data for plots, x is time when data captured in s, y is value of data
bad_solutions_y = []
bad_solutions_x = []
best_energy_y = []
best_energy_x = []
temperature_y = []
temperature_x = []

start_time = 0

class MemberType:
    def __init__(self):
        self.solution = list(range(data_cb["num_queens"]))
        self.energy = 0.0

def get_rand():
    return random.randint(0, data_cb["num_queens"] - 1)

def get_srand():
    return random.random()

def tweak_solution(member):
    x, y = random.sample(range(data_cb["num_queens"]), 2)
    member.solution[x], member.solution[y] \
        = member.solution[y], member.solution[x]

def initialize_solution(member):
    random.shuffle(member.solution)
    tweak_solution(member)

def emit_solution(member):
    board = [['.' for _ in range(data_cb["num_queens"])]
              for _ in range(data_cb["num_queens"])]
    for x in range(data_cb["num_queens"]):
        board[x][member.solution[x]] = 'Q'
    for row in board:
        print(' '.join(row))
    print("\n")
```

```

def compute_energy(member):
    conflicts = 0
    board = [['.' for _ in range(data_cb["num_queens"])]
              for _ in range(data_cb["num_queens"])]
    for i in range(data_cb["num_queens"]):
        board[i][member.solution[i]] = 'Q'
    dx = [-1, 1, -1, 1]
    dy = [-1, 1, 1, -1]
    for i in range(data_cb["num_queens"]):
        x, y = i, member.solution[i]
        for j in range(4):
            temp_x, temp_y = x, y
            while True:
                temp_x += dx[j]
                temp_y += dy[j]
                if temp_x < 0 or temp_x >= data_cb["num_queens"] \
                    or temp_y < 0 or temp_y >= data_cb["num_queens"]:
                    break
                if board[temp_x][temp_y] == 'Q':
                    conflicts += 1
    member.energy = float(conflicts)

def copy_solution(dest, src):
    dest.solution = src.solution[:]
    dest.energy = src.energy

def emit_solution_image(member):
    global data_cb
    cell_size = 30 # Size of each cell in pixels
    border_size = cell_size # Border size to accommodate letters and numbers
    board_size = data_cb["num_queens"] * cell_size
    # Adjusted image size to include the border
    image_size = board_size + 2 * border_size
    # Load the SVG file directly
    queen_image = Image.open('bQ.png').resize((cell_size, cell_size))

    # light wood color
    border_color = '#D2B48C'

    # Create a blank white image
    image = Image.new('RGB', (image_size, image_size), border_color)
    draw = ImageDraw.Draw(image)

    # Define colors for the checkerboard
    color1 = '#D3D3D3' # Light gray
    color2 = '#A9A9A9' # Dark gray

    # Draw the checkerboard pattern
    for x in range(data_cb["num_queens"]):
        for y in range(data_cb["num_queens"]):
            top_left = (y * cell_size + border_size,
                        x * cell_size + border_size)
            bottom_right = (top_left[0] + cell_size, top_left[1] + cell_size)
            fill_color = color1 if (x + y) % 2 == 0 else color2
            draw.rectangle([top_left, bottom_right], fill=fill_color)

    if member is not None:
        # Draw the queens
        for x in range(data_cb["num_queens"]):
            y = member.solution[x]
            # Calculate the top-left corner of the cell
            top_left = (y * cell_size + border_size,
                        x * cell_size + border_size)
            # Paste the queen image
            image.paste(queen_image, top_left, queen_image)

```

```

# Draw the border with letters and numbers
font_path = "arialbd.ttf" # Path to a bold TTF font file
font_size = 12 # Adjust the font size as needed
font = ImageFont.truetype(font_path, font_size)

def index_to_letters(index):
    letters = []
    while index >= 0:
        letters.append(chr(index % 26 + ord('A')))
        index = index // 26 - 1
    return ''.join(reversed(letters))

for i in range(data_cb["num_queens"]):
    # Draw letters on the top and bottom
    letter = index_to_letters(i)
    draw.text((border_size + i * cell_size + cell_size // 2,
               border_size // 2),
              letter, fill='black', anchor='mm', font=font)
    draw.text((border_size + i * cell_size + cell_size // 2,
               image_size - border_size // 2),
              letter, fill='black', anchor='mm', font=font)

    # Draw numbers on the left and right
    number = str(i + 1)
    draw.text((border_size // 2, border_size + i * cell_size +
               cell_size // 2), number, fill='black', anchor='mm', font=font)
    draw.text((image_size - border_size // 2, border_size + i * cell_size +
               cell_size // 2), number, fill='black', anchor='mm', font=font)

# resize image to 4k
image = image.resize((1000, 1000), Image.Resampling.BICUBIC)

image.save('solution.png')

def main():
    temperature_x.clear()
    temperature_y.clear()
    best_energy_x.clear()
    best_energy_y.clear()
    bad_solutions_x.clear()
    bad_solutions_y.clear()
    rejected = 0

    emit_solution_image(None)
    width, height, channels, data = dpq.load_image("solution.png")
    dpq.set_value("texture_tag", data)

    temperature = data_cb["init_temp"]
    current = MemberType()
    working = MemberType()
    best = MemberType()
    best.energy = 100.0

    start_time = time.time()

    initialize_solution(current)
    compute_energy(current)
    copy_solution(working, current)
    global is_running
    while (temperature > data_cb["fin_temp"]) and (best.energy > 0.0000000):
        if not is_running:
            break
        # print(f"Temperature : {temperature}")
        temperature_x.append(float(time.time() - start_time))

```

```

temperature_y.append(float(temperature))
accepted = 0
for _ in range(data_cb["steps_per_change"]):
    tweak_solution(working)
    compute_energy(working)
    delta = working.energy - current.energy
    if delta <= 0 or math.exp(-delta / temperature) > get_srand():
        accepted += 1 if delta > 0 else 0
        copy_solution(current, working)
        if current.energy < best.energy:
            copy_solution(best, current)
        else:
            rejected += 1
            if rejected % 10 == 0:
                bad_solutions_x.append(float(time.time() - start_time))
                bad_solutions_y.append(rejected)
    else:
        copy_solution(working, current)
# print(f"Best energy = {best.energy}")
best_energy_x.append(float(time.time() - start_time))
best_energy_y.append(float(best.energy))
temperature *= data_cb["alpha"]

# Fit the axis data
if len(temperature_x) > 0:
    dpd.set_value("series_tag_temp", [temperature_x, temperature_y])
    # dpd.fit_axis_data("y_axis_temp")
    dpd.fit_axis_data("x_axis_temp")
    dpd.set_value("temp_text", f"{temperature:.2f}")
if len(best_energy_x) > 0:
    dpd.set_value("series_tag_energy", [best_energy_x, best_energy_y])
    # dpd.fit_axis_data("y_axis_energy")
    dpd.set_axis_limits("y_axis_energy", -0.5,
                        max(best_energy_y) + 0.5)
    dpd.fit_axis_data("x_axis_energy")
    dpd.set_value("energy_text", f"{best.energy:.2f}")
if len(bad_solutions_x) > 0:
    dpd.set_value("series_tag_bad", [bad_solutions_x, bad_solutions_y])
    dpd.fit_axis_data("x_axis_bad")
    dpd.set_axis_limits("y_axis_bad", -0.5, max(bad_solutions_y) + 0.5)
    dpd.set_value("bads_text", f"{bad_solutions_y[-1]}")
if len(temperature_x) > 0:
    dpd.set_value("time_text", f"{temperature_x[-1]:.2f}s")

# emit_solution(best)
emit_solution_image(best)
# update texture
width, height, channels, data = dpd.load_image("solution.png")
dpd.set_value("texture_tag", data)
stop()

def run():
    global is_running
    global process
    if process is None:
        is_running = True
        process = threading.Thread(target=main)
        process.start()

def stop():
    global is_running
    global process
    is_running = False
    if process is not None:
        try:

```

```

        process.join()
    except:
        pass
process = None

def update_layout():
    port_width = dpq.get_viewport_client_width()
    port_height = dpq.get_viewport_client_height()

    inputs_height = 26

    board_size = min(port_width, port_height - inputs_height * 6)

    plots_width = port_width - board_size / 0.97
    plots_height = board_size / 3 - 3 # / 3.031

    inputs_width = min(max(port_width * 0.15, 100), 500)
    outputs_width = min(max(port_width * 0.15, 100), 500)

    if dpq.does_item_exist("board"):
        dpq.configure_item("board", width=board_size, height=board_size)

    if dpq.does_item_exist("max_temp"):
        dpq.configure_item("max_temp", width=inputs_width)
    if dpq.does_item_exist("min_temp"):
        dpq.configure_item("min_temp", width=inputs_width)
    if dpq.does_item_exist("alpha"):
        dpq.configure_item("alpha", width=inputs_width)
    if dpq.does_item_exist("num_queens"):
        dpq.configure_item("num_queens", width=inputs_width)
    if dpq.does_item_exist("steps"):
        dpq.configure_item("steps", width=inputs_width)

    if dpq.does_item_exist("temp_text"):
        dpq.configure_item("temp_text", width=outputs_width)
    if dpq.does_item_exist("energy_text"):
        dpq.configure_item("energy_text", width=outputs_width)
    if dpq.does_item_exist("bads_text"):
        dpq.configure_item("bads_text", width=outputs_width)
    if dpq.does_item_exist("time_text"):
        dpq.configure_item("time_text", width=outputs_width)

    if dpq.does_item_exist("plot_temp"):
        dpq.configure_item("plot_temp", width=plots_width, height=plots_height)
    if dpq.does_item_exist("plot_energy"):
        dpq.configure_item("plot_energy", width=plots_width,
                           height=plots_height)
    if dpq.does_item_exist("plot_bads"):
        dpq.configure_item("plot_bads", width=plots_width, height=plots_height)

    if dpq.does_item_exist("y_axis_temp"):
        dpq.set_axis_limits("y_axis_temp", 0, data_cb["init_temp"])

if __name__ == "__main__":
    dpq.create_context()
    dpq.create_viewport(title='N-Queen Task', width=800, height=600)
    dpq.set_viewport_resize_callback(update_layout)
    # print(dpq.get_viewport_client_width())
    # print(dpq.get_viewport_client_height())

    width, height, channels, data = dpq.load_image("solution.png")
    with dpq.texture_registry(show=False):
        dpq.add_dynamic_texture(
            width=width, height=height, default_value=data, tag="texture_tag")

```

```

with dpq.window(label="Example Window", tag="fullscreen"):

    with dpq.theme(tag="plot_theme_temp"):
        with dpq.theme_component(dpq.mvLineSeries):
            dpq.add_theme_color(
                dpq.mvPlotCol_Line, (255, 99, 99),
                category=dpq.mvThemeCat_Plots)
            dpq.add_theme_style(
                dpq.mvPlotStyleVar_Marker, dpq.mvPlotMarker_None,
                category=dpq.mvThemeCat_Plots)
            dpq.add_theme_style(
                dpq.mvPlotStyleVar_MarkerSize, 0,
                category=dpq.mvThemeCat_Plots)
            dpq.add_theme_style(
                dpq.mvPlotStyleVar_LineWeight, 3.0,
                category=dpq.mvThemeCat_Plots)
        with dpq.theme(tag="plot_theme_energy"):
            with dpq.theme_component(dpq.mvLineSeries):
                dpq.add_theme_color(
                    dpq.mvPlotCol_Line, (22, 90, 255),
                    category=dpq.mvThemeCat_Plots)
                dpq.add_theme_style(
                    dpq.mvPlotStyleVar_Marker, dpq.mvPlotMarker_None,
                    category=dpq.mvThemeCat_Plots)
                dpq.add_theme_style(
                    dpq.mvPlotStyleVar_MarkerSize, 0,
                    category=dpq.mvThemeCat_Plots)
                dpq.add_theme_style(
                    dpq.mvPlotStyleVar_LineWeight, 3.0,
                    category=dpq.mvThemeCat_Plots)
        with dpq.theme(tag="plot_theme_bads"):
            with dpq.theme_component(dpq.mvLineSeries):
                dpq.add_theme_color(
                    dpq.mvPlotCol_Line, (123, 123, 123),
                    category=dpq.mvThemeCat_Plots)
                dpq.add_theme_style(
                    dpq.mvPlotStyleVar_Marker, dpq.mvPlotMarker_None,
                    category=dpq.mvThemeCat_Plots)
                dpq.add_theme_style(
                    dpq.mvPlotStyleVar_MarkerSize, 0,
                    category=dpq.mvThemeCat_Plots)
                dpq.add_theme_style(
                    dpq.mvPlotStyleVar_LineWeight, 3.0,
                    category=dpq.mvThemeCat_Plots)

    def on_value_change_max_temp(sender, app_data, user_data):
        stop()
        user_data['init_temp'] = float(app_data)
        dpq.set_axis_limits("y_axis_temp", 0, data_cb["init_temp"])

    def on_value_change_min_temp(sender, app_data, user_data):
        stop()
        user_data['fin_temp'] = float(app_data)

    def on_value_change_alpha(sender, app_data, user_data):
        stop()
        user_data['alpha'] = float(app_data)

    def on_value_change_steps(sender, app_data, user_data):
        stop()
        user_data['steps_per_change'] = int(app_data)

    def on_value_change_num_queens(sender, app_data, user_data):
        stop()

```



```

user_data['num_queens'] = int(app_data)
emit_solution_image(None)
width, height, channels, data = dpg.load_image("solution.png")
dpg.set_value("texture_tag", data)

# Максимальная температура
with dpg.group(horizontal=True):
    with dpg.group():
        dpg.add_input_float(label="Max temperature",
                             default_value=100,
                             width=200,
                             callback=on_value_change_max_temp,
                             user_data=data_cb,
                             min_value=0.0,
                             step=1.0,
                             tag="max_temp")

    # Минимальная температура
    dpg.add_input_float(label="Min temperature",
                         default_value=0.01,
                         width=200,
                         callback=on_value_change_min_temp,
                         user_data=data_cb,
                         min_value=0.0,
                         step=0.01,
                         tag="min_temp")

    # Коэффициент понижения температуры
    dpg.add_input_float(label="Temperature reduction coefficient",
                         default_value=0.99,
                         width=200,
                         callback=on_value_change_alpha,
                         user_data=data_cb,
                         min_value=0.0,
                         max_value=1.0,
                         step=0.01,
                         tag="alpha")

    # Количество ферзей
    dpg.add_input_int(label="Number of queens",
                      default_value=8,
                      width=200,
                      callback=on_value_change_num_queens,
                      user_data=data_cb,
                      min_value=1, max_value=30,
                      tag="num_queens")

    # Количество шагов при постоянном значении температуры
    dpg.add_input_int(label="Number of steps at constant "
                           + "temperature",
                      default_value=10,
                      width=200,
                      callback=on_value_change_steps,
                      user_data=data_cb,
                      tag="steps")

with dpg.group():
    # text output
    dpg.add_input_text(default_value="...",
                       readonly=True,
                       tag="temp_text",
                       width=50)

    dpg.add_input_text(default_value="...",
                       readonly=True,
                       tag="energy_text",
                       width=50)

    dpg.add_input_text(default_value="...",
                       readonly=True,
                       tag="bads_text",
                       width=50)

```

```

dpg.add_input_text(default_value="...",
                    readonly=True,
                    tag="time_text",
                    width=50)

with dpg.group():
    dpg.add_text("Temperature")
    dpg.add_text("Best Energy")
    dpg.add_text("Bad Solutions")
    dpg.add_text("Time")

with dpg.group(horizontal=True):
    dpg.add_button(label="run", callback=run)
    dpg.add_button(label="stop", callback=stop)

text_width = dpg.get_viewport_client_width() * 5 / 10
text_height = dpg.get_viewport_client_width() * 5 / 10
with dpg.group(horizontal=True):
    dpg.add_image("texture_tag",
                  width=text_width,
                  height=text_height, tag="board")

emit_solution_image(None)
width, height, channels, data = dpg.load_image("solution.png")
dpg.set_value("texture_tag", data)

plots_width = text_width / 2
plots_height = text_height / 3.031 # 4.548
with dpg.group():
    with dpg.plot(label="Temperature",
                  width=plots_width,
                  height=plots_height,
                  tag="plot_temp"):
        dpg.add_plot_legend()
        dpg.add_plot_axis(
            dpg.mvXAxis, label="time, s", tag="x_axis_temp")
        dpg.add_plot_axis(
            dpg.mvYAxis, label="temperature", tag="y_axis_temp")
        dpg.set_axis_limits("y_axis_temp", 0, data_cb["init_temp"])
        dpg.add_line_series(
            temperature_x, temperature_y, parent="y_axis_temp",
            tag="series_tag_temp")
        dpg.bind_item_theme("series_tag_temp", "plot_theme_temp")
    with dpg.plot(label="Best Energy",
                  width=plots_width,
                  height=plots_height,
                  tag="plot_energy"):
        dpg.add_plot_legend()
        dpg.add_plot_axis(
            dpg.mvXAxis, label="time, s", tag="x_axis_energy")
        dpg.add_plot_axis(
            dpg.mvYAxis, label="energy", tag="y_axis_energy")
        dpg.add_line_series(
            best_energy_x, best_energy_y, parent="y_axis_energy",
            tag="series_tag_energy")
        dpg.bind_item_theme("series_tag_energy", "plot_theme_energy")
    with dpg.plot(label="Bad Solutions",
                  width=plots_width,
                  height=plots_height,
                  tag="plot_bads"):
        dpg.add_plot_legend()
        dpg.add_plot_axis(
            dpg.mvXAxis, label="time, s", tag="x_axis_bad")
        dpg.add_plot_axis(
            dpg.mvYAxis, label="bad solutions", tag="y_axis_bad")

```

```
        dpg.add_line_series(
            bad_solutions_x, bad_solutions_y, parent="y_axis_bad",
            tag="series_tag_bad")
        dpg.bind_item_theme("series_tag_bad", "plot_theme_bads")

    update_layout()

    dpg.setup_dearpygui()
    dpg.set_primary_window("fullscreen", True)
    dpg.show_viewport()
    dpg.start_dearpygui()
    dpg.destroy_context()
```