**ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ**

**ЯМАЛО-НЕНЕЦКОГО АВТОНОМНОГО ОКРУГА**

**«МУРАВЛЕНКОВСКИЙ МНОГОПРОФИЛЬНЫЙ КОЛЛЕДЖ»**

**(ГБПОУ ЯНАО «Муравленковский многопрофильный колледж»)**

**Документация**

**Хакатон «Цифровые двойники в энергетике»**

Выполнили студенты **ГБПОУ ЯНАО «Муравленковский многопрофильный колледж» специальностей:**

**Сетевое и системное администрирование:**

Харитонов Данила Денисович (3 курс) СА-22д

Полищук Андрей Викторович (3 курс) СА-22д

**Техническая эксплуатация и обслуживание электрического и электромеханического оборудования (по отраслям):**

Михайлов Владислав Сергеевич (4 курс) ЭЛ-21д

Трегубов Алексей Владимирович (4 курс) ЭЛ-21д

**Теплоснабжение и теплотехническое оборудование:**

Валитов Данил Вадимович (4 курс) ТЭ-21д

**Наставник:**

Дьяченко Вера Михайловна

1. **Описание применяемых математических моделей и программных средств**

В нашем приложении, созданном на **Python** с использованием библиотеки **Tkinter**, реализуется модель электроснабжения и отопления с визуализацией объектов и их параметров. Приложение рассчитывает выработку и потребление энергии для различных объектов, таких как ГТЭС и ЦПС, основываясь на энергетическом балансе. Общая выработка энергии вычисляется как сумма выработки всех активных турбин, тогда как общее потребление определяется как сумма потребления всех подключенных объектов (кроме ГТЭС).

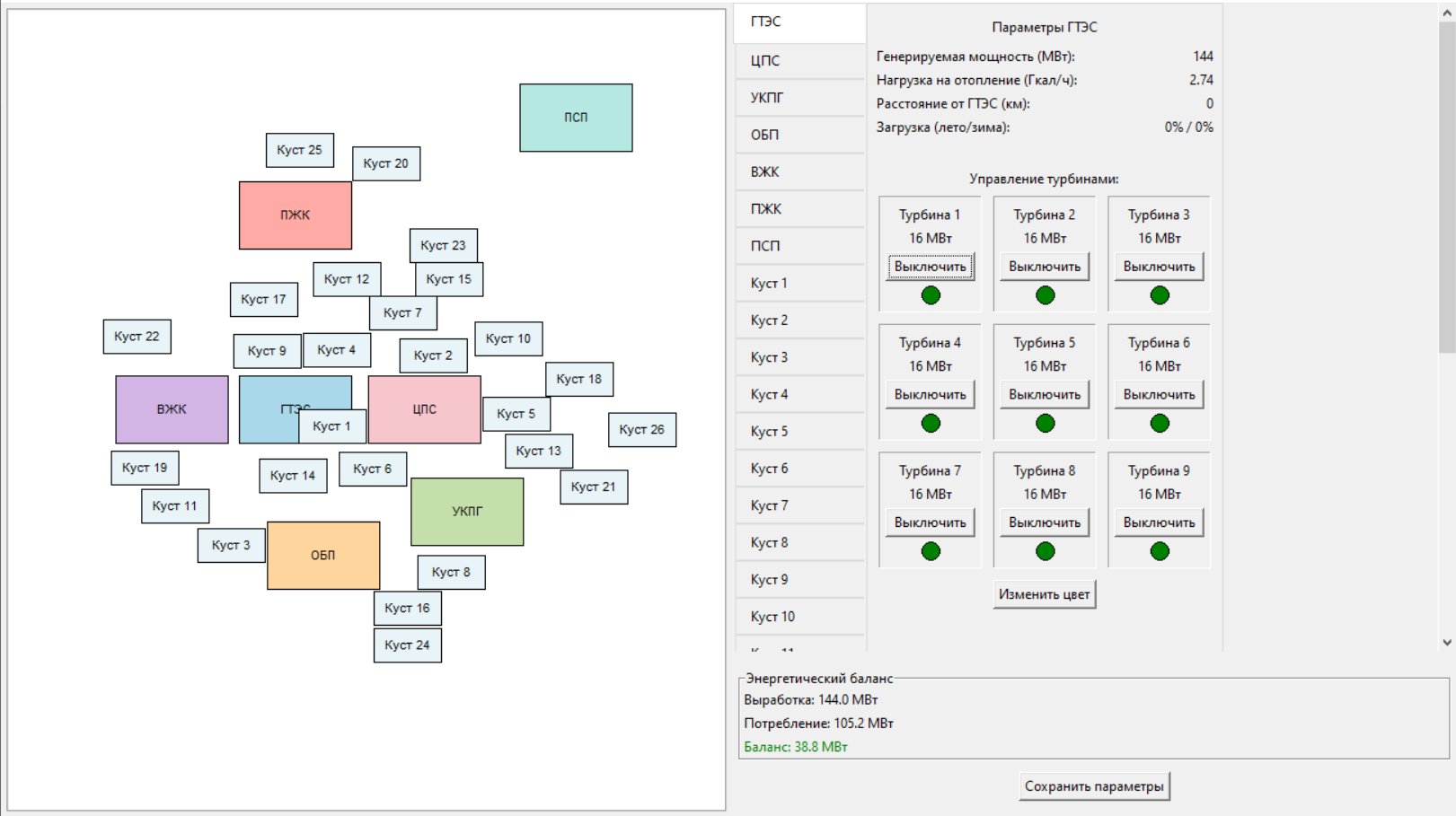
Объекты располагаются по расстоянию от ГТЭС, что может отражать физическую модель распределения объектов в системе. Расстояние влияет на потребление энергии, что можно учитывать в более сложных моделях. Геометрическое размещение объектов осуществляется с помощью тригонометрических функций (синус и косинус), которые используются для вычисления координат объектов на плоскости.

Каждый объект имеет свои параметры нагрузки, которые могут варьироваться в зависимости от времени года, что позволяет проводить более детальный анализ потребления и выработки энергии. Управление турбинами также является важной частью модели: пользователи могут включать и выключать турбины, что напрямую влияет на общую выработку энергии.

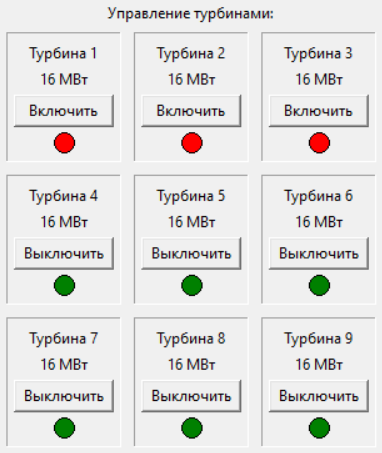
В приложении также применяются стандартные математические функции, такие как **math.cos()** и **math.sin()**, для вычисления координат объектов на основе углов и радиусов.

Модуль **colorchooser** используется для выбора цвета объектов, что добавляет интерактивности в приложение.

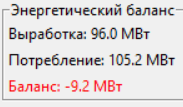
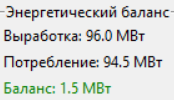
1. **Руководство пользователя**
2. Запустите код



1. Первоначально все источники питания включены (3 из них требуется выключить: 2 вывести на КР, 1 в резерв)



1. В связи с отключением трёх ГТУ мощность ГТЭС падает и нам требуется частично отключить потребителей.

1. **Листинг программы**

import tkinter as tk

from tkinter import ttk, colorchooser

import math

class VerticalNotebook(ttk.Notebook):

def \_\_init\_\_(self, master=None, \*\*kw):

ttk.Notebook.\_\_init\_\_(self, master, \*\*kw)

self.\_\_vertical\_init()

def \_\_vertical\_init(self):

style = ttk.Style()

style.configure("Vertical.TNotebook", tabposition="wn")

style.configure("Vertical.TNotebook.Tab", padding=[10, 5], width=15)

self.configure(style="Vertical.TNotebook")

class ScrollableNotebookFrame(tk.Frame):

def \_\_init\_\_(self, master, \*\*kwargs):

tk.Frame.\_\_init\_\_(self, master, \*\*kwargs)

self.canvas = tk.Canvas(self, highlightthickness=0)

self.scrollbar = ttk.Scrollbar(self, orient="vertical", command=self.canvas.yview)

self.scrollable\_frame = tk.Frame(self.canvas)

self.scrollable\_frame.bind(

"<Configure>",

lambda e: self.canvas.configure(

scrollregion=self.canvas.bbox("all")

)

)

self.canvas.create\_window((0, 0), window=self.scrollable\_frame, anchor="nw")

self.canvas.configure(yscrollcommand=self.scrollbar.set)

self.canvas.pack(side="left", fill="both", expand=True)

self.scrollbar.pack(side="right", fill="y")

class Building:

def \_\_init\_\_(self, canvas, name, x, y, width=100, height=60, color="lightgray"):

self.canvas = canvas

self.name = name

self.x = x

self.y = y

self.width = width

self.height = height

self.color = color

self.rect = self.canvas.create\_rectangle(

x, y, x + width, y + height, fill=color, outline="black"

)

self.text = self.canvas.create\_text(

x + width / 2, y + height / 2, text=name, fill="black", font=("Arial", 8)

)

self.power\_consumption = 0

self.heat\_consumption = 0

self.power\_generation = 0

self.connected = True

self.turbines = []

def update(self, x=None, y=None, width=None, height=None, color=None):

if x is not None:

self.x = x

if y is not None:

self.y = y

if width is not None:

self.width = width

if height is not None:

self.height = height

if color is not None:

self.color = color

self.canvas.coords(

self.rect, self.x, self.y, self.x + self.width, self.y + self.height

)

self.canvas.itemconfig(self.rect, fill=self.color)

self.canvas.coords(

self.text, self.x + self.width / 2, self.y + self.height / 2

)

class Turbine:

def \_\_init\_\_(self, id, power=16):

self.id = id

self.power = power

self.active = True

self.runtime = 0

self.last\_maintenance = 0

class Application(tk.Tk):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.title("Расчет параметров электроснабжения и отопления")

self.geometry("1400x800")

# Данные из таблицы

self.objects\_data = {

"ГТЭС": {"power": 114, "heat": 2.74, "distance": 0, "summer\_load": 0, "winter\_load": 0},

"ЦПС": {"power": 22, "heat": 0, "distance": 0.3, "summer\_load": 70, "winter\_load": 55},

"УКПГ": {"power": 30.0, "heat": 3.04, "distance": 0.50, "summer\_load": 90, "winter\_load": 95},

"ОБП": {"power": 9.0, "heat": 0.64, "distance": 3.0, "summer\_load": 80, "winter\_load": 90},

"ВЖК": {"power": 2.0, "heat": 0.35, "distance": 3.50, "summer\_load": 100, "winter\_load": 100},

"ПЖК": {"power": 3.0, "heat": 2.58, "distance": 4.0, "summer\_load": 100, "winter\_load": 100},

"ПСП": {"power": 10.0, "heat": 1.91, "distance": 100.0, "summer\_load": 100, "winter\_load": 100},

}

# Добавляем кусты

for i in range(1, 27):

self.objects\_data[f"Куст {i}"] = {

"power": 0.7 + (i % 10) \* 0.1 if i > 1 else 0.7,

"heat": 0,

"distance": 1.0 + (i-1) \* 0.5 if i <= 19 else 17 + (i-20) \* 1.0,

"summer\_load": 50,

"winter\_load": 100

}

# Левая часть - схема (в рамке)

self.scheme\_frame = tk.Frame(self, width=700, height=800, bg="white", bd=2, relief=tk.GROOVE)

self.scheme\_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=5, pady=5)

self.canvas = tk.Canvas(self.scheme\_frame, bg="white", highlightthickness=0)

self.canvas.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

# Основные здания

self.buildings = []

# Центр (ГТЭС)

center\_x, center\_y = 350, 350

gtes = Building(self.canvas, "ГТЭС", center\_x-150, center\_y-30, width=100, height=60, color="#a8d8ea")

gtes.power\_generation = 114

gtes.power\_consumption = 0

gtes.heat\_consumption = 2.74

# Добавляем турбины для ГТЭС (без визуализации на схеме)

self.turbines = []

for i in range(9):

turbine = Turbine(i+1)

self.turbines.append(turbine)

gtes.turbines.append(turbine)

gtes.power\_generation = sum(t.power for t in gtes.turbines if t.active)

self.buildings.append(gtes)

# Расположим объекты по расстоянию от ГТЭС с проверкой на пересечения

objects\_to\_place = [

("ЦПС", 0.3, "#f7c5cc"),

("УКПГ", 0.5, "#c4dfaa"),

("ОБП", 3.0, "#ffd59e"),

("ВЖК", 4.2, "#d3b5e5"),

("ПЖК", 4.0, "#ffaaa5"),

("ПСП", 100.0, "#b5e8e0")

]

# Распределим углы с учетом предотвращения наложений

angles = [0, 60, 120, 180, 240, 300]

radius\_step = 80

placed\_buildings = []

for i, (name, distance, color) in enumerate(objects\_to\_place):

base\_radius = min(distance \* 50, 250) if distance < 50 else 300

radius = base\_radius

angle = angles[i]

while True:

x = center\_x + radius \* math.cos(math.radians(angle)) - 50

y = center\_y + radius \* math.sin(math.radians(angle)) - 30

intersects = False

for (bx, by, bw, bh) in placed\_buildings:

if (x < bx + bw and x + 100 > bx and

y < by + bh and y + 60 > by):

intersects = True

break

if not intersects:

break

radius += radius\_step

building = Building(self.canvas, name, x, y, width=100, height=60, color=color)

building.power\_consumption = self.objects\_data[name]["power"]

building.heat\_consumption = self.objects\_data[name]["heat"]

self.buildings.append(building)

placed\_buildings.append((x, y, 100, 60))

# Кусты - расположим по расстоянию от ГТЭС с проверкой на пересечения

self.wells = []

placed\_wells = []

for i in range(1, 27):

distance = self.objects\_data[f"Куст {i}"]["distance"]

angle = (i \* 137.5) % 360

base\_radius = min(distance \* 9, 300) if distance < 15 else 150

radius = base\_radius

while True:

x = center\_x + radius \* math.cos(math.radians(angle)) - 30

y = center\_y + radius \* math.sin(math.radians(angle)) - 15

intersects = False

for (bx, by, bw, bh) in placed\_buildings + placed\_wells:

if (x < bx + bw and x + 60 > bx and

y < by + bh and y + 30 > by):

intersects = True

break

if not intersects:

break

radius += 10

angle = (angle + 5) % 360

well = Building(self.canvas, f"Куст {i}", x, y, width=60, height=30, color="#e8f4f8")

well.power\_consumption = self.objects\_data[f"Куст {i}"]["power"]

self.wells.append(well)

placed\_wells.append((x, y, 60, 30))

# Правая часть - параметры

self.params\_frame = tk.Frame(self, width=500, height=800, bg="#f0f0f0")

self.params\_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)

self.main\_container = ScrollableNotebookFrame(self.params\_frame)

self.main\_container.pack(fill=tk.BOTH, expand=True)

self.notebook = VerticalNotebook(self.main\_container.scrollable\_frame)

self.notebook.pack(fill=tk.BOTH, expand=True)

self.tabs = {}

for building in self.buildings + self.wells:

tab = tk.Frame(self.notebook)

self.tabs[building.name] = tab

self.notebook.add(tab, text=building.name)

self.create\_building\_tab(tab, building)

# Панель управления

self.control\_frame = tk.Frame(self.params\_frame, bg="#f0f0f0")

self.control\_frame.pack(fill=tk.X, pady=10)

self.stats\_frame = tk.LabelFrame(self.control\_frame, text="Энергетический баланс", bg="#f0f0f0")

self.stats\_frame.pack(fill=tk.X, padx=5, pady=5)

self.generation\_label = tk.Label(self.stats\_frame, text="Выработка: 0 МВт", bg="#f0f0f0")

self.generation\_label.pack(anchor="w")

self.consumption\_label = tk.Label(self.stats\_frame, text="Потребление: 0 МВт", bg="#f0f0f0")

self.consumption\_label.pack(anchor="w")

self.balance\_label = tk.Label(self.stats\_frame, text="Баланс: 0 МВт", bg="#f0f0f0")

self.balance\_label.pack(anchor="w")

btn\_save = tk.Button(

self.control\_frame,

text="Сохранить параметры",

command=self.save\_parameters,

)

btn\_save.pack(pady=5)

if self.buildings:

self.notebook.select(0)

self.main\_container.canvas.bind\_all("<MouseWheel>",

lambda e: self.main\_container.canvas.yview\_scroll(int(-1\*(e.delta/120)), "units"))

self.update\_stats()

def create\_building\_tab(self, tab, building):

main\_frame = tk.Frame(tab)

main\_frame.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

tk.Label(main\_frame, text=f"Параметры {building.name}").grid(row=0, column=0, columnspan=2, pady=5)

if building.name == "ГТЭС":

tk.Label(main\_frame, text="Генерируемая мощность (МВт):").grid(row=1, column=0, sticky="w")

power\_label = tk.Label(main\_frame, text=f"{building.power\_generation}")

power\_label.grid(row=1, column=1, sticky="e")

else:

tk.Label(main\_frame, text="Потребляемая мощность (МВт):").grid(row=1, column=0, sticky="w")

power\_label = tk.Label(main\_frame, text=f"{building.power\_consumption}")

power\_label.grid(row=1, column=1, sticky="e")

if building.heat\_consumption > 0:

tk.Label(main\_frame, text="Нагрузка на отопление (Гкал/ч):").grid(row=2, column=0, sticky="w")

heat\_label = tk.Label(main\_frame, text=f"{building.heat\_consumption}")

heat\_label.grid(row=2, column=1, sticky="e")

tk.Label(main\_frame, text="Расстояние от ГТЭС (км):").grid(row=3, column=0, sticky="w")

distance = self.objects\_data[building.name]["distance"] if building.name in self.objects\_data else 0

distance\_label = tk.Label(main\_frame, text=f"{distance}")

distance\_label.grid(row=3, column=1, sticky="e")

if building.name in self.objects\_data:

tk.Label(main\_frame, text="Загрузка (лето/зима):").grid(row=4, column=0, sticky="w")

load\_label = tk.Label(main\_frame,

text=f"{self.objects\_data[building.name]['summer\_load']}% / {self.objects\_data[building.name]['winter\_load']}%")

load\_label.grid(row=4, column=1, sticky="e")

if building.name == "ГТЭС":

tk.Label(main\_frame, text="\nУправление турбинами:").grid(row=5, column=0, columnspan=2, pady=(10,0))

turbines\_frame = tk.Frame(main\_frame)

turbines\_frame.grid(row=6, column=0, columnspan=2, sticky="ew")

for i, turbine in enumerate(building.turbines):

row = i // 3

col = i % 3

turbine\_frame = tk.Frame(turbines\_frame, bd=1, relief=tk.GROOVE, padx=5, pady=5)

turbine\_frame.grid(row=row, column=col, padx=5, pady=5, sticky="nsew")

tk.Label(turbine\_frame, text=f"Турбина {turbine.id}").pack()

tk.Label(turbine\_frame, text=f"{turbine.power} МВт").pack()

btn\_state = tk.Button(

turbine\_frame,

text="Выключить" if turbine.active else "Включить",

command=lambda t=turbine: self.toggle\_turbine(t),

width=10

)

btn\_state.pack(pady=2)

status\_indicator = tk.Canvas(turbine\_frame, width=20, height=20, highlightthickness=0)

status\_indicator.create\_oval(2, 2, 18, 18, fill="green" if turbine.active else "red", outline="black")

status\_indicator.pack()

if building.name != "ГТЭС":

tk.Label(main\_frame, text="\nСостояние объекта:").grid(row=7, column=0, columnspan=2, pady=(10,0))

self.connection\_var = tk.BooleanVar(value=building.connected)

connection\_btn = tk.Checkbutton(

main\_frame,

text="Подключен к сети",

variable=self.connection\_var,

command=lambda: self.toggle\_building\_connection(building),

indicatoron=False,

selectcolor="lightgreen" if building.connected else "lightcoral",

width=15

)

connection\_btn.grid(row=8, column=0, columnspan=2, pady=5)

btn\_color = tk.Button(

main\_frame,

text="Изменить цвет",

command=lambda: self.change\_building\_color(building),

)

btn\_color.grid(row=9, column=0, columnspan=2, pady=5)

def toggle\_turbine(self, turbine):

turbine.active = not turbine.active

gtes = next(b for b in self.buildings if b.name == "ГТЭС")

gtes.power\_generation = sum(t.power for t in gtes.turbines if t.active)

self.update\_stats()

self.update\_building\_tab("ГТЭС")

def toggle\_building\_connection(self, building):

building.connected = not building.connected

self.update\_stats()

self.update\_building\_tab(building.name)

def update\_building\_tab(self, building\_name):

tab = self.tabs[building\_name]

for widget in tab.winfo\_children():

widget.destroy()

building = next((b for b in self.buildings + self.wells if b.name == building\_name), None)

if building:

self.create\_building\_tab(tab, building)

def change\_building\_color(self, building):

color = colorchooser.askcolor()[1]

if color:

building.update(color=color)

def update\_stats(self):

total\_generation = sum(b.power\_generation for b in self.buildings if hasattr(b, 'power\_generation'))

total\_consumption = sum(b.power\_consumption for b in self.buildings + self.wells if b.connected and b.name != "ГТЭС")

balance = total\_generation - total\_consumption

self.generation\_label.config(text=f"Выработка: {total\_generation:.1f} МВт")

self.consumption\_label.config(text=f"Потребление: {total\_consumption:.1f} МВт")

self.balance\_label.config(text=f"Баланс: {balance:.1f} МВт",

fg="green" if balance >= 0 else "red")

def save\_parameters(self):

print("Параметры сохранены")

if \_\_name\_\_ == "\_\_main\_\_":

app = Application()

app.mainloop()