main

▼ View Source

```python
import tkinter as tk
from tkinter import filedialog
import random
from logic import (
    generate_static_points,
    generate_grid_sensors_auto,
    add_sink,
    compute_neighbors,
    bfs_paths_to_sink,
    solve_ilp,
    transmit_data_along_path
)
from config import load_config, cfg, OFFSET_X, OFFSET_Y

class SensorGUI:
    def __init__(self, root):
        self.root = root
        self.root.title("Symulacja Sieci Sensorów")
        self.running = False
        self.cycle = 0
        self.packets_sent = 0
        self.packets_delivered = 0
        self.packets_lost = 0
        self.latencies = []
        self.low_coverage_cycles = 0

        self.canvas_width = 800
        self.canvas_height = 800
        self.scale = 1.0
        self.cfg = None
        self.entries = {}

        self.canvas = tk.Canvas(root, width=self.canvas_width, height=self.canvas
        self.canvas.pack()

        self.frame = tk.Frame(root)
        self.frame.pack()

        tk.Button(self.frame, text="Start", command=self.start_simulation).grid(r
        tk.Button(self.frame, text="Pauza", command=self.pause_simulation).grid(r
        tk.Button(self.frame, text="Reset", command=self.reset_simulation).grid(r
        tk.Button(self.frame, text="Załaduj konfigurację", command=self.load_conf
        tk.Button(self.frame, text="Ustawienia WSN", command=self.open_settings_w

        self.status_label = tk.Label(self.frame, text="Status: Załaduj konfigurac
        self.status_label.grid(row=2, column=0, columnspan=5)

        self.points = []
        self.sensors = []
        self.sink = None
        self.draw_scene()

    def load_config_gui(self):
```

```
54          filename = filedialog.askopenfilename(filetypes=[("INI files", "*.ini")])
55          if filename:
56              global cfg
57              self.cfg = load_config(filename)
58              cfg = self.cfg
59              max_canvas_size = 900
60              self.scale = max_canvas_size / max(self.cfg["FIELD_WIDTH"], self.cfg[
61              self.canvas_width = int(self.cfg["FIELD_WIDTH"] * self.scale + OFFSET
62              self.canvas_height = int(self.cfg["FIELD_HEIGHT"] * self.scale + OFFS
63
64              self.canvas.config(width=self.canvas_width, height=self.canvas_height
65
66              self.status_label.config(text="Status: Konfiguracja załadowana")
67              self.reset_simulation()
68
69      def open_settings_window(self):
70          if self.cfg is None:
71              self.status_label.config(text="Załaduj konfigurację przed edycją")
72              return
73
74          settings_window = tk.Toplevel(self.root)
75          settings_window.title("Ustawienia WSN")
76
77          self.entries = {}
78          row = 0
79          for key, value in self.cfg.items():
80              if key == "SEED":
81                  continue
82
83              label = tk.Label(settings_window, text=f"{key}:")
84              label.grid(row=row, column=0, padx=5, pady=2)
85
86              entry = tk.Entry(settings_window)
87              entry.insert(0, str(value))
88              entry.grid(row=row, column=1, padx=5, pady=2)
89              self.entries[key] = entry
90              row += 1
91
92          save_button = tk.Button(settings_window, text="Zapisz i Zastosuj", commar
93          save_button.grid(row=row, columnspan=2, pady=10)
94
95      def save_and_apply_settings(self, window):
96          new_cfg = self.cfg.copy()
97          try:
98              for key, entry in self.entries.items():
99                  value = entry.get()
100                 if isinstance(new_cfg[key], int):
101                     new_cfg[key] = int(value)
102                 elif isinstance(new_cfg[key], float):
103                     new_cfg[key] = float(value)
104                 else:
105                     new_cfg[key] = value
106
107             global cfg
108             cfg = new_cfg
109             self.cfg = cfg
110             self.status_label.config(text="Status: Ustawienia zapisane i zastosov
111             self.reset_simulation()
112             window.destroy()
```

```
113
114         except ValueError:
115             self.status_label.config(text="Błąd: Wartości muszą być numeryczne. S
116
117     def draw_scene(self, paths=None):
118         self.canvas.delete("all")
119
120         if not self.points or not self.sensors:
121             self.canvas.create_text(self.canvas_width // 2, self.canvas_height //
122                                     text="Brak załadowanej konfiguracji.\nZaładu
123                                     font=("Arial", 16), fill="gray")
124             return
125
126         for p in self.points:
127             x, y = p.x * self.scale + OFFSET_X, p.z * self.scale + OFFSET_Y
128             self.canvas.create_oval(x - 3, y - 3, x + 3, y + 3, fill="black")
129
130         for s in self.sensors:
131             x, y = s.x * self.scale + OFFSET_X, s.z * self.scale + OFFSET_Y
132             outline = "black"
133             if s.is_failed:
134                 fill = "gray"
135             elif s.id == "SINK":
136                 fill = "yellow"
137             elif s.energy <= 0:
138                 fill = "red"
139             elif s.is_on:
140                 fill = "green"
141             else:
142                 fill = "blue"
143
144             self.canvas.create_oval(x - 5, y - 5, x + 5, y + 5, fill=fill, outlir
145
146             if s.id != "SINK":
147                 self.canvas.create_oval(
148                     x - self.cfg["COVERAGE_RADIUS"] * self.scale,
149                     y - self.cfg["COVERAGE_RADIUS"] * self.scale,
150                     x + self.cfg["COVERAGE_RADIUS"] * self.scale,
151                     y + self.cfg["COVERAGE_RADIUS"] * self.scale,
152                     outline=outline, dash=(2, 2)
153                 )
154
155         if paths:
156             for s in self.sensors:
157                 if s.is_on and s.id != "SINK":
158                     path = paths.get(s)
159                     if path:
160                         for i in range(len(path) - 1):
161                             x1, y1 = path[i].x * self.scale + OFFSET_X, path[i].z
162                             x2, y2 = path[i + 1].x * self.scale + OFFSET_X, path[
163                             self.canvas.create_line(x1, y1, x2, y2, fill="orange"
164
165     def save_logs(self):
166         if not self.sensors:
167             return
168         with open("logs.txt", "a") as f:
169             pdr = (self.packets_delivered / self.packets_sent * 100) if self.pack
170             avg_latency = sum(self.latencies) / len(self.latencies) if self.later
171             f.write(f"=== KONIEC | Cykl: {self.cycle} | PDR={pdr:.2f}% | Latency=
```

```
172                 for s in self.sensors:
173                     color = ""
174                     if s.is_failed:
175                         color = "gray"
176                     elif s.id == "SINK":
177                         color = "yellow"
178                     elif s.energy <= 0:
179                         color = "red"
180                     elif s.is_on:
181                         color = "green"
182                     else:
183                         color = "blue"
184                     f.write(f"Czujnik {s.id}: energia={s.energy:.2f}, kolor={color},
185                 f.write("\n")
186 
187         def start_simulation(self):
188             if self.cfg is None:
189                 self.status_label.config(text="Załaduj konfigurację przed startem")
190                 return
191             if not self.running:
192                 self.running = True
193                 self.simulate_step()
194 
195         def pause_simulation(self):
196             self.running = False
197 
198         def reset_simulation(self):
199             self.running = False
200             self.cycle = 0
201             self.packets_sent = 0
202             self.packets_delivered = 0
203             self.packets_lost = 0
204             self.latencies.clear()
205             self.low_coverage_cycles = 0
206 
207             if self.cfg is None:
208                 self.points = []
209                 self.sensors = []
210                 self.sink = None
211                 self.status_label.config(text="Status: Brak konfiguracji")
212             else:
213                 self.points = generate_static_points(self.cfg)
214                 self.sensors = generate_grid_sensors_auto(self.cfg["FIELD_WIDTH"], se
215                 self.sink = add_sink(self.sensors, self.cfg)
216                 comm_range = self.cfg["COVERAGE_RADIUS"] * 1.5
217                 compute_neighbors(self.sensors, comm_range)
218                 self.status_label.config(text="Status: Symulacja zresetowana")
219 
220             self.draw_scene()
221 
222         def simulate_step(self):
223             if not self.running or self.cfg is None:
224                 if self.cfg is None:
225                     self.status_label.config(text="Brak konfiguracji - przerwano symu
226                     self.running = False
227                 return
228 
229             self.cycle += 1
230 
```

```
231             for s in self.sensors:
232                 if s.id != "SINK" and not s.is_failed and random.random() < self.cfg[
233                     s.is_failed = True
234                     s.energy = 0
235 
236             selected = solve_ilp(self.points, self.sensors, self.cfg)
237 
238             covered_points = set()
239             active_this_cycle = set()
240 
241             sensors_with_energy = [s for s in self.sensors if s.energy > 0 and not s.
242             paths = bfs_paths_to_sink(self.sink, sensors_with_energy)
243 
244             for s in selected:
245                 if s.is_failed:
246                     continue
247 
248                 hops = paths.get(s, [])
249                 self.packets_sent += 1
250 
251                 if hops:
252                     delivered = transmit_data_along_path(hops, self.cfg)
253                     if delivered:
254                         self.packets_delivered += 1
255                         self.latencies.append(len(hops) - 1)
256                         for sensor_in_path in hops:
257                             if sensor_in_path.id != "SINK":
258                                 active_this_cycle.add(sensor_in_path)
259 
260                         covered_now = [i for i, p in enumerate(self.points) if s.cove
261                         covered_points.update(covered_now)
262                     else:
263                         self.packets_lost += 1
264 
265             for s in self.sensors:
266                 if s.id != "SINK" and not s.is_failed:
267                     if s in active_this_cycle:
268                         s.is_on = True
269                     else:
270                         s.is_on = False
271                         s.energy -= self.cfg["SLEEP_COST"]
272 
273             coverage = len(covered_points) / len(self.points) * 100 if self.points el
274             active = sum(1 for s in self.sensors if s.energy > 0 and s.id != "SINK")
275             on_count = sum(1 for s in self.sensors if s.is_on)
276             failed_count = sum(1 for s in self.sensors if s.is_failed)
277             required_coverage = self.cfg["MIN_COVERAGE_PERCENT"]
278 
279             if coverage < required_coverage:
280                 self.low_coverage_cycles += 1
281             else:
282                 self.low_coverage_cycles = 0
283 
284             if self.low_coverage_cycles >= 3:
285                 self.status_label.config(
286                     text=f"KONIEC SYMULACJI | Cykl: {self.cycle}",
287                     fg="black"
288                 )
289                 self.running = False
```

```
290             self.save_logs()
291         elif active == 0:
292             self.status_label.config(
293                 text=f"KONIEC SYMULACJI-wszystkie czujniki rozładowane | Cykl: {s
294                 fg="black"
295             )
296             self.running = False
297             self.save_logs()
298         else:
299             self.status_label.config(
300                 text=f"Cykl: {self.cycle} | Pokrycie: {coverage:.2f}% | ON: {on_c
301                 fg="black"
302             )
303             self.draw_scene(paths)
304             self.root.after(100, self.simulate_step)
305 
306 if __name__ == "__main__":
307     root = tk.Tk()
308     app = SensorGUI(root)
309     root.mainloop()
```

---

class SensorGUI:

▼ View Source

```
16 class SensorGUI:
17     def __init__(self, root):
18         self.root = root
19         self.root.title("Symulacja Sieci Sensorów")
20         self.running = False
21         self.cycle = 0
22         self.packets_sent = 0
23         self.packets_delivered = 0
24         self.packets_lost = 0
25         self.latencies = []
26         self.low_coverage_cycles = 0
27 
28         self.canvas_width = 800
29         self.canvas_height = 800
30         self.scale = 1.0
31         self.cfg = None
32         self.entries = {}
33 
34         self.canvas = tk.Canvas(root, width=self.canvas_width, height=self.canvas
35         self.canvas.pack()
36 
37         self.frame = tk.Frame(root)
38         self.frame.pack()
39 
40         tk.Button(self.frame, text="Start", command=self.start_simulation).grid(r
41         tk.Button(self.frame, text="Pauza", command=self.pause_simulation).grid(r
42         tk.Button(self.frame, text="Reset", command=self.reset_simulation).grid(r
43         tk.Button(self.frame, text="Załaduj konfigurację", command=self.load_conf
44         tk.Button(self.frame, text="Ustawienia WSN", command=self.open_settings_w
45 
46         self.status_label = tk.Label(self.frame, text="Status: Załaduj konfigurac
47         self.status_label.grid(row=2, column=0, columnspan=5)
48 
49         self.points = []
```

```
50          self.sensors = []
51          self.sink = None
52          self.draw_scene()
53
54      def load_config_gui(self):
55          filename = filedialog.askopenfilename(filetypes=[("INI files", "*.ini")])
56          if filename:
57              global cfg
58              self.cfg = load_config(filename)
59              cfg = self.cfg
60              max_canvas_size = 900
61              self.scale = max_canvas_size / max(self.cfg["FIELD_WIDTH"], self.cfg[
62              self.canvas_width = int(self.cfg["FIELD_WIDTH"] * self.scale + OFFSET
63              self.canvas_height = int(self.cfg["FIELD_HEIGHT"] * self.scale + OFFS
64
65              self.canvas.config(width=self.canvas_width, height=self.canvas_height
66
67              self.status_label.config(text="Status: Konfiguracja załadowana")
68              self.reset_simulation()
69
70      def open_settings_window(self):
71          if self.cfg is None:
72              self.status_label.config(text="Załaduj konfigurację przed edycją")
73              return
74
75          settings_window = tk.Toplevel(self.root)
76          settings_window.title("Ustawienia WSN")
77
78          self.entries = {}
79          row = 0
80          for key, value in self.cfg.items():
81              if key == "SEED":
82                  continue
83
84              label = tk.Label(settings_window, text=f"{key}:")
85              label.grid(row=row, column=0, padx=5, pady=2)
86
87              entry = tk.Entry(settings_window)
88              entry.insert(0, str(value))
89              entry.grid(row=row, column=1, padx=5, pady=2)
90              self.entries[key] = entry
91              row += 1
92
93          save_button = tk.Button(settings_window, text="Zapisz i Zastosuj", commar
94          save_button.grid(row=row, columnspan=2, pady=10)
95
96      def save_and_apply_settings(self, window):
97          new_cfg = self.cfg.copy()
98          try:
99              for key, entry in self.entries.items():
100                 value = entry.get()
101                 if isinstance(new_cfg[key], int):
102                     new_cfg[key] = int(value)
103                 elif isinstance(new_cfg[key], float):
104                     new_cfg[key] = float(value)
105                 else:
106                     new_cfg[key] = value
107
108             global cfg
```

```
109             cfg = new_cfg
110             self.cfg = cfg
111             self.status_label.config(text="Status: Ustawienia zapisane i zastosov
112             self.reset_simulation()
113             window.destroy()
114 
115         except ValueError:
116             self.status_label.config(text="Błąd: Wartości muszą być numeryczne. S
117 
118     def draw_scene(self, paths=None):
119         self.canvas.delete("all")
120 
121         if not self.points or not self.sensors:
122             self.canvas.create_text(self.canvas_width // 2, self.canvas_height //
123                                     text="Brak załadowanej konfiguracji.\nZaładu
124                                     font=("Arial", 16), fill="gray")
125             return
126 
127         for p in self.points:
128             x, y = p.x * self.scale + OFFSET_X, p.z * self.scale + OFFSET_Y
129             self.canvas.create_oval(x - 3, y - 3, x + 3, y + 3, fill="black")
130 
131         for s in self.sensors:
132             x, y = s.x * self.scale + OFFSET_X, s.z * self.scale + OFFSET_Y
133             outline = "black"
134             if s.is_failed:
135                 fill = "gray"
136             elif s.id == "SINK":
137                 fill = "yellow"
138             elif s.energy <= 0:
139                 fill = "red"
140             elif s.is_on:
141                 fill = "green"
142             else:
143                 fill = "blue"
144 
145             self.canvas.create_oval(x - 5, y - 5, x + 5, y + 5, fill=fill, outlir
146 
147             if s.id != "SINK":
148                 self.canvas.create_oval(
149                     x - self.cfg["COVERAGE_RADIUS"] * self.scale,
150                     y - self.cfg["COVERAGE_RADIUS"] * self.scale,
151                     x + self.cfg["COVERAGE_RADIUS"] * self.scale,
152                     y + self.cfg["COVERAGE_RADIUS"] * self.scale,
153                     outline=outline, dash=(2, 2)
154                 )
155 
156         if paths:
157             for s in self.sensors:
158                 if s.is_on and s.id != "SINK":
159                     path = paths.get(s)
160                     if path:
161                         for i in range(len(path) - 1):
162                             x1, y1 = path[i].x * self.scale + OFFSET_X, path[i].z
163                             x2, y2 = path[i + 1].x * self.scale + OFFSET_X, path[
164                             self.canvas.create_line(x1, y1, x2, y2, fill="orange'
165 
166     def save_logs(self):
167         if not self.sensors:
```

```
168             return
169         with open("logs.txt", "a") as f:
170             pdr = (self.packets_delivered / self.packets_sent * 100) if self.pack
171             avg_latency = sum(self.latencies) / len(self.latencies) if self.later
172             f.write(f"=== KONIEC | Cykl: {self.cycle} | PDR={pdr:.2f}% | Latency=
173             for s in self.sensors:
174                 color = ""
175                 if s.is_failed:
176                     color = "gray"
177                 elif s.id == "SINK":
178                     color = "yellow"
179                 elif s.energy <= 0:
180                     color = "red"
181                 elif s.is_on:
182                     color = "green"
183                 else:
184                     color = "blue"
185                 f.write(f"Czujnik {s.id}: energia={s.energy:.2f}, kolor={color},
186             f.write("\n")
187 
188     def start_simulation(self):
189         if self.cfg is None:
190             self.status_label.config(text="Załaduj konfigurację przed startem")
191             return
192         if not self.running:
193             self.running = True
194             self.simulate_step()
195 
196     def pause_simulation(self):
197         self.running = False
198 
199     def reset_simulation(self):
200         self.running = False
201         self.cycle = 0
202         self.packets_sent = 0
203         self.packets_delivered = 0
204         self.packets_lost = 0
205         self.latencies.clear()
206         self.low_coverage_cycles = 0
207 
208         if self.cfg is None:
209             self.points = []
210             self.sensors = []
211             self.sink = None
212             self.status_label.config(text="Status: Brak konfiguracji")
213         else:
214             self.points = generate_static_points(self.cfg)
215             self.sensors = generate_grid_sensors_auto(self.cfg["FIELD_WIDTH"], se
216             self.sink = add_sink(self.sensors, self.cfg)
217             comm_range = self.cfg["COVERAGE_RADIUS"] * 1.5
218             compute_neighbors(self.sensors, comm_range)
219             self.status_label.config(text="Status: Symulacja zresetowana")
220 
221         self.draw_scene()
222 
223     def simulate_step(self):
224         if not self.running or self.cfg is None:
225             if self.cfg is None:
226                 self.status_label.config(text="Brak konfiguracji - przerwano symu
```

```
                self.running = False
            return

        self.cycle += 1

        for s in self.sensors:
            if s.id != "SINK" and not s.is_failed and random.random() < self.cfg[
                s.is_failed = True
                s.energy = 0

        selected = solve_ilp(self.points, self.sensors, self.cfg)

        covered_points = set()
        active_this_cycle = set()

        sensors_with_energy = [s for s in self.sensors if s.energy > 0 and not s.
        paths = bfs_paths_to_sink(self.sink, sensors_with_energy)

        for s in selected:
            if s.is_failed:
                continue

            hops = paths.get(s, [])
            self.packets_sent += 1

            if hops:
                delivered = transmit_data_along_path(hops, self.cfg)
                if delivered:
                    self.packets_delivered += 1
                    self.latencies.append(len(hops) - 1)
                    for sensor_in_path in hops:
                        if sensor_in_path.id != "SINK":
                            active_this_cycle.add(sensor_in_path)

                    covered_now = [i for i, p in enumerate(self.points) if s.cove
                    covered_points.update(covered_now)
                else:
                    self.packets_lost += 1

        for s in self.sensors:
            if s.id != "SINK" and not s.is_failed:
                if s in active_this_cycle:
                    s.is_on = True
                else:
                    s.is_on = False
                    s.energy -= self.cfg["SLEEP_COST"]

        coverage = len(covered_points) / len(self.points) * 100 if self.points el
        active = sum(1 for s in self.sensors if s.energy > 0 and s.id != "SINK")
        on_count = sum(1 for s in self.sensors if s.is_on)
        failed_count = sum(1 for s in self.sensors if s.is_failed)
        required_coverage = self.cfg["MIN_COVERAGE_PERCENT"]

        if coverage < required_coverage:
            self.low_coverage_cycles += 1
        else:
            self.low_coverage_cycles = 0

        if self.low_coverage_cycles >= 3:
```

```
286             self.status_label.config(
287                 text=f"KONIEC SYMULACJI | Cykl: {self.cycle}",
288                 fg="black"
289             )
290             self.running = False
291             self.save_logs()
292         elif active == 0:
293             self.status_label.config(
294                 text=f"KONIEC SYMULACJI-wszystkie czujniki rozładowane | Cykl: {s
295                 fg="black"
296             )
297             self.running = False
298             self.save_logs()
299         else:
300             self.status_label.config(
301                 text=f"Cykl: {self.cycle} | Pokrycie: {coverage:.2f}% | ON: {on_c
302                 fg="black"
303             )
304             self.draw_scene(paths)
305             self.root.after(100, self.simulate_step)
```

SensorGUI(root) ▼ View Source

```
17     def __init__(self, root):
18         self.root = root
19         self.root.title("Symulacja Sieci Sensorów")
20         self.running = False
21         self.cycle = 0
22         self.packets_sent = 0
23         self.packets_delivered = 0
24         self.packets_lost = 0
25         self.latencies = []
26         self.low_coverage_cycles = 0
27 
28         self.canvas_width = 800
29         self.canvas_height = 800
30         self.scale = 1.0
31         self.cfg = None
32         self.entries = {}
33 
34         self.canvas = tk.Canvas(root, width=self.canvas_width, height=self.ca
35         self.canvas.pack()
36 
37         self.frame = tk.Frame(root)
38         self.frame.pack()
39 
40         tk.Button(self.frame, text="Start", command=self.start_simulation).gr
41         tk.Button(self.frame, text="Pauza", command=self.pause_simulation).gr
42         tk.Button(self.frame, text="Reset", command=self.reset_simulation).gr
43         tk.Button(self.frame, text="Załaduj konfigurację", command=self.load_
44         tk.Button(self.frame, text="Ustawienia WSN", command=self.open_setting
45 
46         self.status_label = tk.Label(self.frame, text="Status: Załaduj konfigu
47         self.status_label.grid(row=2, column=0, columnspan=5)
48 
49         self.points = []
50         self.sensors = []
```

```
51            self.sink = None
52            self.draw_scene()
```

---

root

running

cycle

packets_sent

packets_delivered

packets_lost

latencies

low_coverage_cycles

canvas_width

canvas_height

scale

cfg

entries

canvas

frame

status_label

points

sensors

sink

def load_config_gui(self): ▼ View Source

```
54        def load_config_gui(self):
55            filename = filedialog.askopenfilename(filetypes=[("INI files", "*.ini'
56            if filename:
57                global cfg
```

```
58              self.cfg = load_config(filename)
59              cfg = self.cfg
60              max_canvas_size = 900
61              self.scale = max_canvas_size / max(self.cfg["FIELD_WIDTH"], self.
62              self.canvas_width = int(self.cfg["FIELD_WIDTH"] * self.scale + OFI
63              self.canvas_height = int(self.cfg["FIELD_HEIGHT"] * self.scale + (
64
65              self.canvas.config(width=self.canvas_width, height=self.canvas_he:
66
67              self.status_label.config(text="Status: Konfiguracja załadowana")
68              self.reset_simulation()
```

def open_settings_window(self): ▼ View Source

```
70      def open_settings_window(self):
71          if self.cfg is None:
72              self.status_label.config(text="Załaduj konfigurację przed edycją"
73              return
74
75          settings_window = tk.Toplevel(self.root)
76          settings_window.title("Ustawienia WSN")
77
78          self.entries = {}
79          row = 0
80          for key, value in self.cfg.items():
81              if key == "SEED":
82                  continue
83
84              label = tk.Label(settings_window, text=f"{key}:")
85              label.grid(row=row, column=0, padx=5, pady=2)
86
87              entry = tk.Entry(settings_window)
88              entry.insert(0, str(value))
89              entry.grid(row=row, column=1, padx=5, pady=2)
90              self.entries[key] = entry
91              row += 1
92
93          save_button = tk.Button(settings_window, text="Zapisz i Zastosuj", cor
94          save_button.grid(row=row, columnspan=2, pady=10)
```

def save_and_apply_settings(self, window): ▼ View Source

```
96      def save_and_apply_settings(self, window):
97          new_cfg = self.cfg.copy()
98          try:
99              for key, entry in self.entries.items():
100                 value = entry.get()
101                 if isinstance(new_cfg[key], int):
102                     new_cfg[key] = int(value)
103                 elif isinstance(new_cfg[key], float):
104                     new_cfg[key] = float(value)
105                 else:
106                     new_cfg[key] = value
107
108             global cfg
```

```
109             cfg = new_cfg
110             self.cfg = cfg
111             self.status_label.config(text="Status: Ustawienia zapisane i zast
112             self.reset_simulation()
113             window.destroy()
114 
115         except ValueError:
116             self.status_label.config(text="Błąd: Wartości muszą być numeryczn
```

---

def draw_scene(self, paths=**None**): ▼ View Source

```
118     def draw_scene(self, paths=None):
119         self.canvas.delete("all")
120 
121         if not self.points or not self.sensors:
122             self.canvas.create_text(self.canvas_width // 2, self.canvas_heigh
123                                     text="Brak załadowanej konfiguracji.\nZa
124                                     font=("Arial", 16), fill="gray")
125             return
126 
127         for p in self.points:
128             x, y = p.x * self.scale + OFFSET_X, p.z * self.scale + OFFSET_Y
129             self.canvas.create_oval(x - 3, y - 3, x + 3, y + 3, fill="black"
130 
131         for s in self.sensors:
132             x, y = s.x * self.scale + OFFSET_X, s.z * self.scale + OFFSET_Y
133             outline = "black"
134             if s.is_failed:
135                 fill = "gray"
136             elif s.id == "SINK":
137                 fill = "yellow"
138             elif s.energy <= 0:
139                 fill = "red"
140             elif s.is_on:
141                 fill = "green"
142             else:
143                 fill = "blue"
144 
145             self.canvas.create_oval(x - 5, y - 5, x + 5, y + 5, fill=fill, ou
146 
147             if s.id != "SINK":
148                 self.canvas.create_oval(
149                     x - self.cfg["COVERAGE_RADIUS"] * self.scale,
150                     y - self.cfg["COVERAGE_RADIUS"] * self.scale,
151                     x + self.cfg["COVERAGE_RADIUS"] * self.scale,
152                     y + self.cfg["COVERAGE_RADIUS"] * self.scale,
153                     outline=outline, dash=(2, 2)
154                 )
155 
156         if paths:
157             for s in self.sensors:
158                 if s.is_on and s.id != "SINK":
159                     path = paths.get(s)
160                     if path:
161                         for i in range(len(path) - 1):
162                             x1, y1 = path[i].x * self.scale + OFFSET_X, path
163                             x2, y2 = path[i + 1].x * self.scale + OFFSET_X,
```

```
164                 self.canvas.create_line(x1, y1, x2, y2, fill="ora
```

def save_logs(self):

▼ View Source

```
166     def save_logs(self):
167         if not self.sensors:
168             return
169         with open("logs.txt", "a") as f:
170             pdr = (self.packets_delivered / self.packets_sent * 100) if self
171             avg_latency = sum(self.latencies) / len(self.latencies) if self.
172             f.write(f"=== KONIEC | Cykl: {self.cycle} | PDR={pdr:.2f}% | Late
173             for s in self.sensors:
174                 color = ""
175                 if s.is_failed:
176                     color = "gray"
177                 elif s.id == "SINK":
178                     color = "yellow"
179                 elif s.energy <= 0:
180                     color = "red"
181                 elif s.is_on:
182                     color = "green"
183                 else:
184                     color = "blue"
185                 f.write(f"Czujnik {s.id}: energia={s.energy:.2f}, kolor={colo
186             f.write("\n")
```

def start_simulation(self):

▼ View Source

```
188     def start_simulation(self):
189         if self.cfg is None:
190             self.status_label.config(text="Załaduj konfigurację przed starter
191             return
192         if not self.running:
193             self.running = True
194             self.simulate_step()
```

def pause_simulation(self):

▼ View Source

```
196     def pause_simulation(self):
197         self.running = False
```

def reset_simulation(self):

▼ View Source

```
199     def reset_simulation(self):
200         self.running = False
201         self.cycle = 0
202         self.packets_sent = 0
203         self.packets_delivered = 0
204         self.packets_lost = 0
205         self.latencies.clear()
206         self.low_coverage_cycles = 0
207
```

```
208         if self.cfg is None:
209             self.points = []
210             self.sensors = []
211             self.sink = None
212             self.status_label.config(text="Status: Brak konfiguracji")
213         else:
214             self.points = generate_static_points(self.cfg)
215             self.sensors = generate_grid_sensors_auto(self.cfg["FIELD_WIDTH"]
216             self.sink = add_sink(self.sensors, self.cfg)
217             comm_range = self.cfg["COVERAGE_RADIUS"] * 1.5
218             compute_neighbors(self.sensors, comm_range)
219             self.status_label.config(text="Status: Symulacja zresetowana")
220 
221         self.draw_scene()
```

---

def simulate_step(self): ▼ View Source

```
223     def simulate_step(self):
224         if not self.running or self.cfg is None:
225             if self.cfg is None:
226                 self.status_label.config(text="Brak konfiguracji - przerwano
227                 self.running = False
228             return
229 
230         self.cycle += 1
231 
232         for s in self.sensors:
233             if s.id != "SINK" and not s.is_failed and random.random() < self
234                 s.is_failed = True
235                 s.energy = 0
236 
237         selected = solve_ilp(self.points, self.sensors, self.cfg)
238 
239         covered_points = set()
240         active_this_cycle = set()
241 
242         sensors_with_energy = [s for s in self.sensors if s.energy > 0 and n
243         paths = bfs_paths_to_sink(self.sink, sensors_with_energy)
244 
245         for s in selected:
246             if s.is_failed:
247                 continue
248 
249             hops = paths.get(s, [])
250             self.packets_sent += 1
251 
252             if hops:
253                 delivered = transmit_data_along_path(hops, self.cfg)
254                 if delivered:
255                     self.packets_delivered += 1
256                     self.latencies.append(len(hops) - 1)
257                     for sensor_in_path in hops:
258                         if sensor_in_path.id != "SINK":
259                             active_this_cycle.add(sensor_in_path)
260 
261                     covered_now = [i for i, p in enumerate(self.points) if s
262                     covered_points.update(covered_now)
```

```
                else:
                    self.packets_lost += 1

        for s in self.sensors:
            if s.id != "SINK" and not s.is_failed:
                if s in active_this_cycle:
                    s.is_on = True
                else:
                    s.is_on = False
                    s.energy -= self.cfg["SLEEP_COST"]

        coverage = len(covered_points) / len(self.points) * 100 if self.poin
        active = sum(1 for s in self.sensors if s.energy > 0 and s.id != "SI
        on_count = sum(1 for s in self.sensors if s.is_on)
        failed_count = sum(1 for s in self.sensors if s.is_failed)
        required_coverage = self.cfg["MIN_COVERAGE_PERCENT"]

        if coverage < required_coverage:
            self.low_coverage_cycles += 1
        else:
            self.low_coverage_cycles = 0

        if self.low_coverage_cycles >= 3:
            self.status_label.config(
                text=f"KONIEC SYMULACJI | Cykl: {self.cycle}",
                fg="black"
            )
            self.running = False
            self.save_logs()
        elif active == 0:
            self.status_label.config(
                text=f"KONIEC SYMULACJI-wszystkie czujniki rozładowane | Cyk
                fg="black"
            )
            self.running = False
            self.save_logs()
        else:
            self.status_label.config(
                text=f"Cykl: {self.cycle} | Pokrycie: {coverage:.2f}% | ON:
                fg="black"
            )
            self.draw_scene(paths)
            self.root.after(100, self.simulate_step)
```