

Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej

Symulacja dyskretna systemów złożonych

Stochastyczny model inteligentnego oświetlenia

Daniel Klarenbach

Gabriela Proszczuk

Małgorzata Śliwińska



SPIS TREŚCI

1. Wstęp
2. Wyliczanie natężenia światła
 - 2.1. Potrzebne pojęcia
 - 2.2. Założenia projektowe
 - 2.3. Obliczenia dla sensora
 - 2.4. Obliczenia dla okna
3. Implementacja
 - 3.1. Język i środowisko programowania
 - 3.2. Użytkowanie programu
 - 3.3. Elementy programu
 - 3.3.1. MainFrame
 - 3.3.2. Buttons
 - 3.3.3. Room
 - 3.3.4. LightSourceList oraz SensorList
 - 3.3.5. Menu
4. Przykład działania
5. Wnioski

1. Wstęp

Aplikacja umożliwia symulację rozkładu oświetlenia w pokoju w zależności od rodzaju, mocy i ustawienia oświetlenia wewnętrznego, a także od pory dnia i roku oraz od pogody. Użytkownik ma możliwość narysowania pokoju oraz rozmieszczenia w nim oświetlenia sufitowego. Dane o oświetleniu wewnętrznym zbierane są przez sensory na podłodze, o których rozmieszczeniu również decyduje klient. Wartości luminancji odczytanej przez poszczególne sensory oraz światła emitowanego przez poszczególne lampy są widoczne w pasku bocznym aplikacji. Światło jest ilustrowane w postaci heat mapy, która coraz bardziej intensywnemu oświetleniu przypisuje coraz cieplejsze kolory.

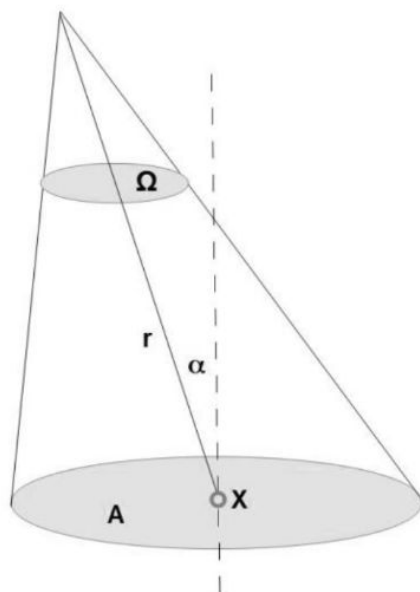
2. Wyliczanie natężenia światła

2.1. Potrzebne pojęcia

- Kąt bryłowy

Jest to część przestrzeni trójwymiarowej ograniczona przez powierzchnię stożkową, czyli wszystkie półproste wychodzące z pewnego ustalonego punktu, zwanego wierzchołkiem, przechodzące przez pewną ustaloną krzywą zamkniętą.

$$d\Omega = \frac{dA}{r^2}$$



Rys. 1. Kąt bryłowy. Oznaczenia: Ω - Kąt bryłowy, A - powierzchnia padania światła, X - punkt na powierzchni A , r - odległość wierzchołka od X , α - kąt padania światła.

- Strumień świetlny

Strumień świetlny to ta część promieniowania optycznego emitowanego przez źródło światła, którą widzi oko ludzkie w jednostce czasu. Wartość strumienia świetlnego wyznaczana jest na podstawie odpowiadającej mu wartości strumienia energetycznego równego ilości energii wysyłanej przez źródło we wszystkich kierunkach w jednostce czasu oraz skuteczności świetlnej, która zależy od rodzaju źródła światła. Jego jednostką jest lumen. Izotropowe źródło punktowe, którego światłość jest równa 1 kandeli, emituje strumień 4π lumenów.

$$\Phi_e = \frac{dW}{dt}$$

$$\Phi_v = K_m V(\lambda) \Phi_e$$

$$[\Phi_v] = cd \cdot sr = lm$$

- Natężenie światła (światłość)

Światłość I_v jest kątową (przestrzenną) gęstością strumienia świetlnego w danym kierunku, inaczej ilorazem strumienia świetlnego wysyłanego przez źródło w danym kierunku, w stożku o nieskończenie małym kącie rozwarcia obejmującym ten kierunek, do kąta bryłowego Ω tego stożka. Jednostką światłości jest kandela.

$$I_v = \frac{d\Phi_v}{d\Omega}$$

$$[I_v] = \frac{lm}{sr} = cd$$

- Natężenie oświetlenia

Charakteryzuje oświetlenie powierzchni, na którą pada strumień światła. Natężenie oświetlenia E_v elementarnej powierzchni dS jest ilorazem elementarnego strumienia świetlnego $d\Phi$ padającego na tę powierzchnię do jej wielkości. Jednostką natężenia oświetlenia jest luks.

$$E_v = \frac{d\Phi_v}{dS}$$

$$[E_v] = \frac{lm}{m^2} = lx$$

2.2. Założenia projektowe

- Źródła światła są punktowe.
- Nie bierzemy pod uwagę światła odbijającego się od powierzchni w pomieszczeniu (lustra, meble itp).
- Sensory są punktowe.

- Natężenie światła jest stałe w każdym punkcie stożka.

2.3. Obliczenia dla sensora

Wartość natężenia oświetlenia każdego z sensorów oblicza się na podstawie światłości i kąta bryłowego stożka światła, a także jego kąta nachylenia w płaszczyźnie XY oraz XZ. Program wylicza, czy dany sensor znajduje się w stożku światła danego źródła światła i jak daleko jest położony od linii wyznaczającej środek stożka. Do wyliczenia ostatecznego natężenia oświetlenia używany jest wzór:

$$E = \frac{I \cos \alpha}{r^2}$$

Oznaczenia jak na Rys. 1.

2.4. Obliczenia dla okna

- Kąt padania słońca:

$$\alpha = \arcsin[\sin \delta \sin \phi + \cos \delta \cos \phi \cos(HRA)],$$

gdzie:

δ - deklinacja słońca: $\delta = -23.45^\circ \times \cos[\frac{360}{365}(d + 10)]$

ϕ - długość geograficzna (zakładamy polską)

d - dzień roku

HRA - kąt godzinowy: $HRA = 15^\circ(LST - 12)$

LST - czas lokalny

- Strumień światła padającego na okno:

$$\Phi = E_0 S_0,$$

gdzie:

E_0 - natężenie oświetlenia na zewnątrz

S_0 - powierzchnia okna

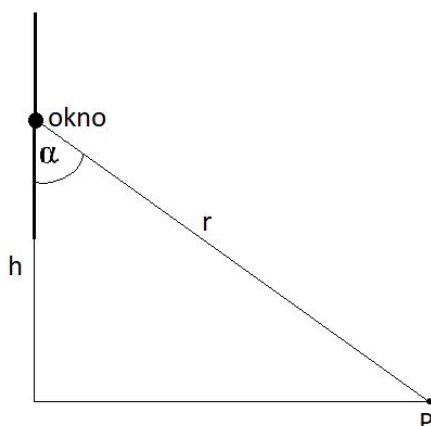
- Światłość:

$$I = \frac{\Phi}{\omega} = \frac{E_0 S_0}{\omega},$$

gdzie:

ω - kąt bryłowy nieba

Ostatecznie natężenie w danym punkcie:



Rys. 2. Schemat światła padającego z okna.

$$E = \frac{I \cos \alpha}{r^2}$$

3. Implementacja

3.1. Język i środowisko programowania

Model został zaimplementowany w języku Java. Jako IDE wybrano IntelliJ firmy JetBrains. GUI stworzono przy pomocy biblioteki graficznej Swing. Użyto także biblioteki AWT. By zwiększyć przejrzystość kodu oraz ułatwić implementację, skorzystano z biblioteki Lombok. Budowanie projektu usprawniono, używając narzędzia Apache Maven. Projekt przechowywano na repozytorium GitHub. Gdy któryś z członków projektu wprowadzał nową funkcjonalność, pushował ją na osobną gałąź. Po zatwierdzeniu przez pozostałych członków zespołu można było zmergować tę gałąź z gałęzią master.

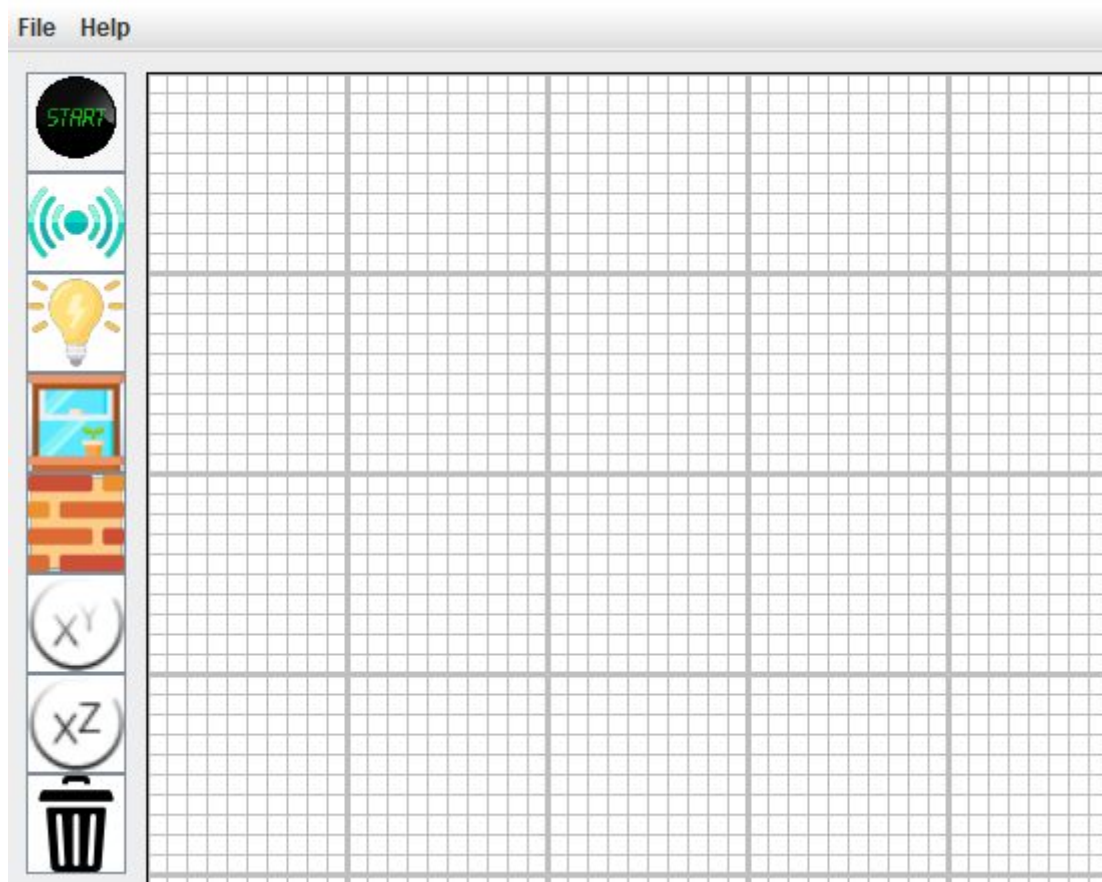
3.2. Użytkowanie programu

Na początku użytkownik naciska przycisk START i ustawia odpowiednią porę roku i dnia oraz natężenie oświetlenia na zewnątrz. Potem należy narysować ściany pokoju i dodać okna. Na podstawie światła zewnętrznego powstaje heat mapa. Następnie można dowolnie dodawać źródła światła i sensory. Po dodaniu źródła światła oraz sensoru pojawiają się okna konfiguracji. Po dodaniu źródła światła trzeba narysować kierunek jego świecenia w płaszczyźnie Xy oraz XZ, by dokończyć konfigurację. Wartości natężenia oświetlenia i nazwy sensorów oraz źródeł światła znajdują się na listach z boku.

3.3. Elementy programu

3.3.1. MainFrame

MainFrame to główna plansza GUI. Jak wcześniej wspomniano, do jej implementacji użyto biblioteki Swing. Zawiera ona przyciski pozwalające na dodanie i modyfikację elementów modelu, a także na ustawienie pory dnia i roku i natężenia oświetlenia na zewnątrz. W lewym górnym rogu znajduje się menu kontekstowe, a po dodaniu elementów ich opisy będą się pojawiały na listach znajdujących się z boku, które aktualizują się po każdym dodaniu nowego źródła światła lub modyfikacji istniejącego.



Rys. 3.3.1. Fragment planszy GUI.

```

public class MainFrame extends JFrame {
    MainFrame(){
        // setting Main Frame attributes
        setTitle("Intelligent Light System Application");
        setPreferredSize(new Dimension( width: 1500+10+10+10+10, height: 800+50+10+10+10)); //50px - menu, 3*10px spaces between components
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setResizable(false);
        setVisible(true);
        setLayout(new BorderLayout( hgap: 0, vgap: 10));

        // menu
        MainMenu menu=new MainMenu();
        add(menu,BorderLayout.NORTH);

        // application panel
        JPanel applicationPanel=new JPanel();
        applicationPanel.setLayout(new BorderLayout( hgap: 10, vgap: 10));
        applicationPanel.setBorder(new EmptyBorder( top: 0, left: 10, bottom: 10, right: 10));

        // buttons
        Room room=new Room();
        applicationPanel.add(new Buttons(room),BorderLayout.WEST);

        // room
        applicationPanel.add(room,BorderLayout.CENTER);

        // list's panel
        JPanel lists=new JPanel();
        lists.setLayout(new BorderLayout( hgap: 10, vgap: 10));
        lists.setPreferredSize(new Dimension( width: 500, height: 800));

        // sensor List
        SensorList sensorList=new SensorList();
        JScrollPane sensorListScroller = new JScrollPane();
        sensorListScroller.setViewportViewView(sensorList);
        lists.add(sensorListScroller);
        lists.add(sensorList,BorderLayout.NORTH);

        // lightSource List
        LightSourceList lightSourceList=new LightSourceList();
        JScrollPane lightSourceListScroller = new JScrollPane();
        lightSourceListScroller.setViewportViewView(lightSourceList);
        add( lightSourceListScroller );
        lists.add(lightSourceList,BorderLayout.SOUTH);

        applicationPanel.add(lists,BorderLayout.EAST);

        add(applicationPanel);

        pack();
    }

    public static void main(String[] args) throws IOException {
        MainFrame mainFrame = new MainFrame();
    }
}

```

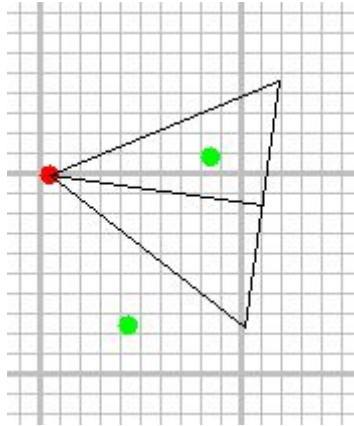
Rys. 3.3.2. Klasa MainFrame.

3.3.2. Buttons

Folder Buttons zawiera klasy przycisków widocznych na Rys. 3.:

- Buttons
- DeleteButton - usuwanie elementów
- LightSourceButton - dodawanie źródła światła

- ModeButton - ustalanie pory dnia, roku oraz oświetlenia na zewnątrz
- SensorButton - dodawanie sensora
- WallButton - dodawanie ściany
- WindowButton - dodawanie okna
- XZAxis Button - konfiguracja kąta świecenia źródła światła w płaszczyźnie XZ.
- XYAxisButton - konfiguracja kąta świecenia źródła światła w płaszczyźnie XY.



Rys. 3.3.3. Konfiguracja kąta świecenia źródła światła w płaszczyźnie XY.

Funkcjonalność pozostałych przycisków zostanie omówiona w punkcie 3.3.3.

Implementacja poszczególnych przycisków jest analogiczna, można zaprezentować ją na podstawie implementacji LightSourceButton:

```
class LightSourceButton extends JButton{
    LightSourceButton(Room room){
        super(new ImageIcon(LightSourceButton.class.getResource( name: "/LightSourceButton.png")));
        setPreferredSize(new Dimension( width: 50, height: 50));
        setBackground(Color.WHITE);
        addMouseListener( (MouseAdapter) mouseClicked(e) -> {
            room.setPaintLightSource(!room.isPaintLightSource());
            room.getLightSources().add(new LightSource( x: 200, y: 200, z: room.getRoomHeight()-50));
        });
    }
}
```

Rys. 3.3.4. Klasa LightSourceButton.

LightSource dodaje się w momencie i miejscu kliknięcia myszą.

Wszystkie przyciski zebrane są w klasie Buttons:

```
public class Buttons extends JPanel {  
    public Buttons(Room room){  
        setLayout(new FlowLayout( align: 0, hgap: 0, vgap: 0));  
        setPreferredSize(new Dimension( width: 50, height: 800));  
  
        ModeButton modeButton = new ModeButton(room);  
        add(modeButton);  
  
        SensorButton sensorButton=new SensorButton(room);  
        add(sensorButton);  
  
        LightSourceButton lightSourceButton=new LightSourceButton(room);  
        add(lightSourceButton);  
    }  
}
```

Rys.3.3.5. Fragment klasy Buttons.

3.3.3. Room

Folder Room zawiera następujące klasy:

- Sun

Zaimplementowano wyliczanie kąta padania słońca, tak jak było to podane w punkcie 2.3. Odpowiednie parametry podaje się w oknie konfiguracyjnym pojawiającym się po naciśnięciu przycisku START.

Lux outside:	20000
Day of year:	123
Time of day:	12
Apply	

Rys. 3.3.6. Okno konfiguracyjne.

```

@Setter
@Getter
@ToString
public class Sun {
    private double angle;
    private int day;
    private int illumination;
    private int time;

    public void setAngle(double latitude) {
        double HRA = (15)*(time-12);
        double incos = (double)360*(day+10)/365;
        double declinationAngle = (-23.35)*Math.cos(incos);
        double sin = Math.sin(Math.toRadians(declinationAngle))*Math.sin(Math.toRadians(latitude));
        double cos = Math.cos(Math.toRadians(declinationAngle))*Math.cos(Math.toRadians(latitude))*Math.cos(Math.toRadians(HRA));
        double elevationAngle = Math.toDegrees(Math.asin(sin+cos));
        this.angle=elevationAngle;
    }

    public double getAngle() { return angle; }
    public int getDay(){ return day;}
    public int getTime(){ return time;}
    public int getIllumination(){ return illumination;}

    public void setDay(int day){ this.day=day; }

    public void setIllumination(int illumination) { this.illumination=illumination; }

    public void setTime(int time) { this.time=time; }
}

```

Rys. 3.3.7. Klasa Sun.

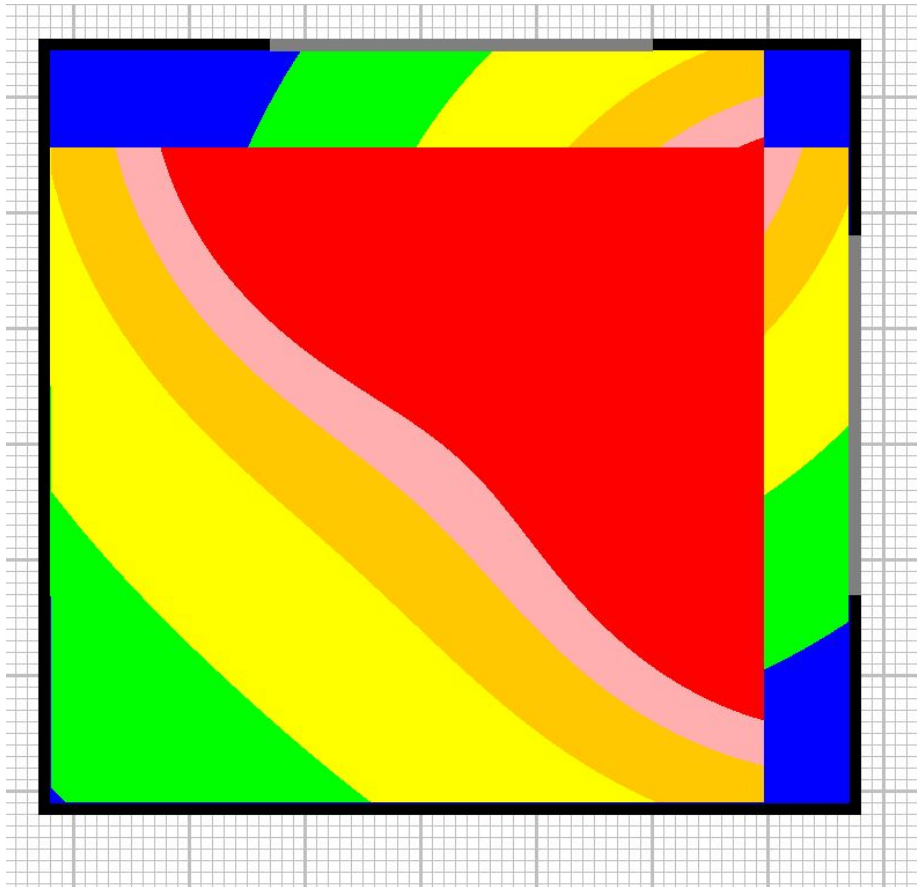
- Wall

Należy narysować ściany po kliknięciu odpowiedniego przycisku. Można stworzyć pokój o nawet nietypowym kształcie.

Klasa Wall zawiera jedynie współrzędne dwóch końców ściany.

- Window

Na ścianach jest możliwość narysowania okna. Jest ono podstawą do powstania heat mapy.



Rys. 3.3.8. Model z heat mapą po dodaniu dwóch okien (zaznaczone szarym kolorem).

Heat mapa uwzględnia to, że okno znajduje się na pewnej wysokości oraz to, że im dalej od okna, tym mniej światła dociera do danego miejsca. Na załączonym przykładzie najmocniej oświetlonym miejscem jest (w przybliżeniu) środek pokoju.

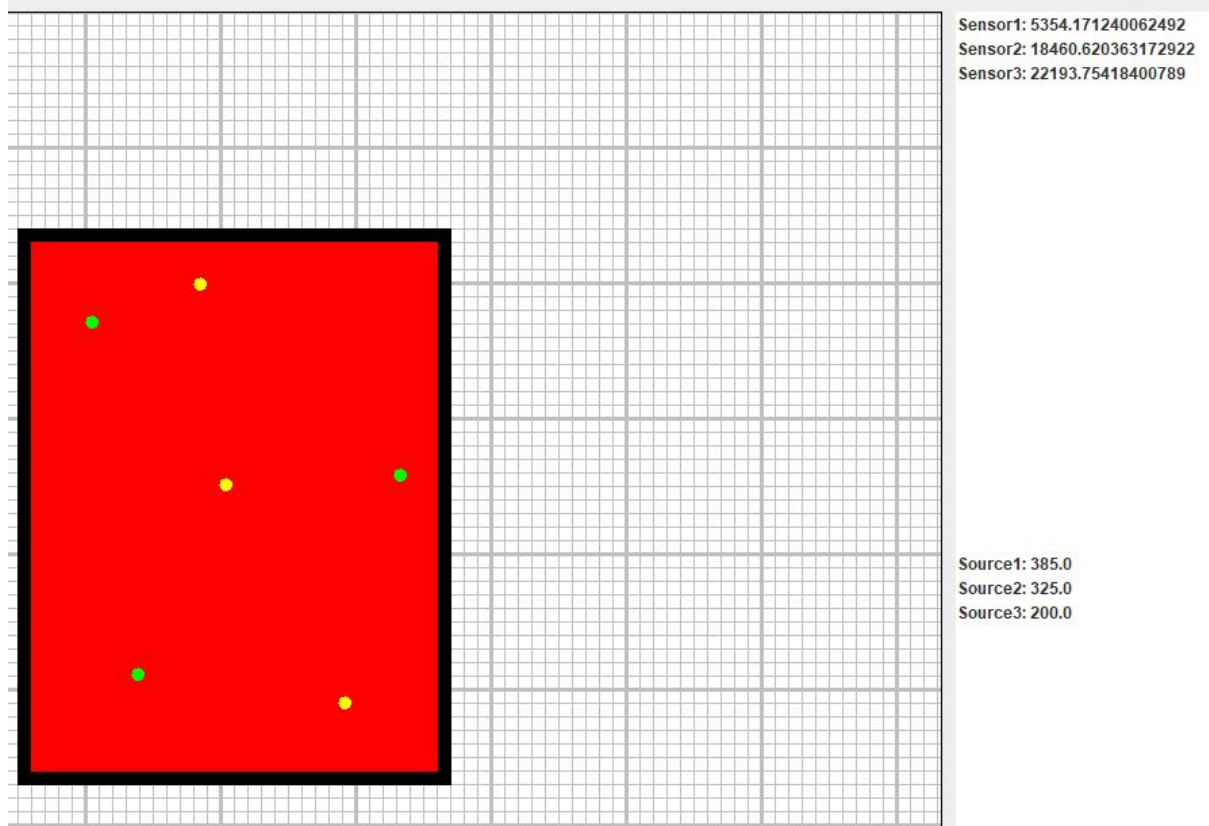
```
@Setter
@Getter
@String
class Window {
    private int x1;
    private int y1;
    private int x2;
    private int y2;
    private double shadow;
    private double height=1.5;

    public double getShadow(double angle){
        this.shadow = 0.8/Math.tan(Math.toRadians(angle));
        return shadow;
    }
}
```

Rys. 3.3.9. Klasa Window.

- LightSource

Dodane źródło światła odznacza się kolorem żółtym. Najnowsze skonfigurowane źródło światła jest czerwone. Po jego dodaniu na liście z prawej strony pojawia się komunikat, że konieczne jest dokończenie konfiguracji źródła. W tym celu należy zaznaczyć kąt jego padania w płaszczyźnie XY oraz XZ. Dopiero po ukończeniu konfiguracji na liście widoczna jest nazwa i strumień świetlny.



Rys. 3.3.10. Przykładowe rozmieszczenie źródeł (żółte) i sensorów (zielone).

```

public LightSource(int x, int y, int z){
    this.x=x;
    this.y=y;
    this.z=z;
}

public void draw(Graphics2D g2d){
    if(isPlaced()==false){
        color=new Color( r: 255, g: 255, b: 153);
        g2d.setColor(color);
        g2d.fillOval( r: x-radius, i1: y-radius, i2: 2*radius, i3: 2*radius);
    }
    else{
        g2d.setColor(color);
        g2d.fillOval( r: x-radius, i1: y-radius, i2: 2*radius, i3: 2*radius);
    }
}

static LightSource getAtLightSource(int x, int y, ArrayList<LightSource> lightSources) {
    LightSource current;
    for (int i = 0; i < lightSources.size(); i++) {
        current=lightSources.get(i);
        if (Math.pow((x - current.getX()),2) + Math.pow((y - current.getY()),2) <= Math.pow((current.getRadius()+10),2)) return current;
    }
    return null;
}

// counts the distance in pixels(1px-1cm) between the center of the lightSource and the given point located on the opticAxis
double countConeHeightWithGivenPoint(int[] point){
    return Math.sqrt(Math.pow((getX()-point[0]),2) + Math.pow((getY()-point[1]),2) + Math.pow((getZ()-point[2]),2));
}

// counts radius in pixels(1px-1cm) of the cone for given height
double countConeRadiusWithGivenPoint(int[] point){
    double radians=Math.toRadians(angle/2);
    double currentHeight=countConeHeightWithGivenPoint(point);
    return Math.tan(radians)*currentHeight;
}

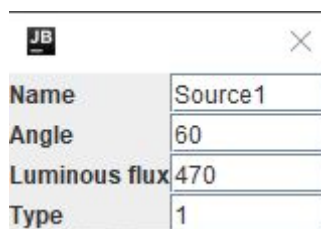
```

Rys. 3.3.11. Konstruktor i funkcje klasy LightSource.

- LightSourceConfigurationPopup

Jest to okno konfiguracji źródła światła. Po jego zamknięciu ustawiają się nazwa, kąt świecenia i strumień świetlny. Można do niego wrócić w dowolnym momencie tworzenia modelu.

W implementacji dodano listener wykrywający zamykanie okna, które skutkuje zapisaniem wprowadzonych wartości.



JB		X	
Name	Source1		
Angle	60		
Luminous flux	470		
Type	1		

Rys. 3.3.11. Okno konfiguracyjne.


```

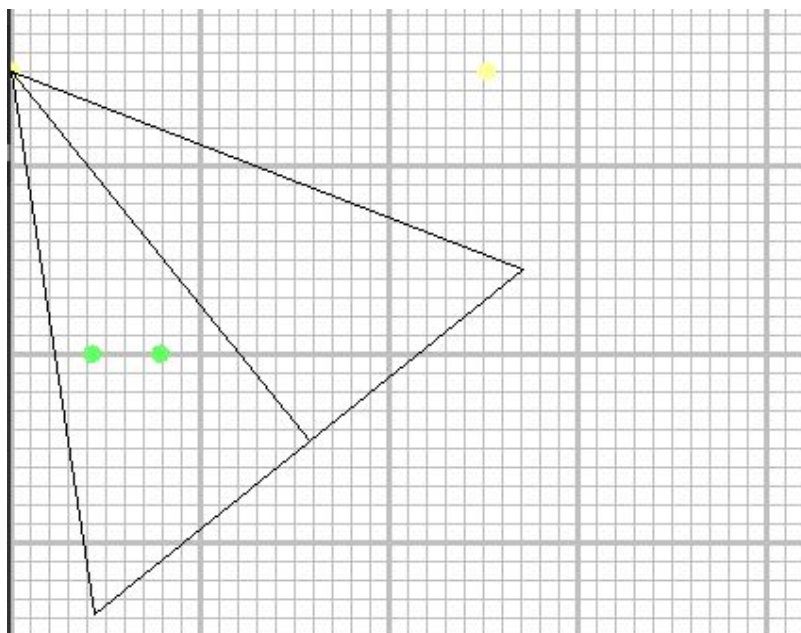
public class LightSourceConfigurationPopup extends JDialog {
    LightSourceConfigurationPopup(LightSource lightSource) {
        setSize( width: 400, height: 400);
        setVisible(true);
        setLayout(new GridLayout( rows: 4, cols: 2));
        JTextField name=new JTextField();
        add(new JLabel( text: "Name"));
        add(name);
        add(new JLabel( text: "Angle"));
        JTextField angle=new JTextField();
        add(angle);
        add(new JLabel( text: "Luminous flux"));
        JTextField luminousFlux=new JTextField();
        add(luminousFlux);
        add(new JLabel( text: "Type"));
        JTextField type=new JTextField();
        add(type);
        addWindowListener((WindowAdapter) windowClosing(e) -> {
            lightSource.setName(name.getText());
            lightSource.setAngle(Double.parseDouble(angle.getText()));
            lightSource.setLuminousFlux(Double.parseDouble(luminousFlux.getText()));
            lightSource.setType(type.getText());
            LightSourceList.updateAllItems();
        });
        pack();
    }
}

```

Rys. 3.3.13. Klasa *LightSourceConfigurationPopup*.

- **XZAxisPopup**

Jest to okno pozwalające na ustalenie kąta świecenia źródła światła w płaszczyźnie XZ. Widać na nim, że wszystkie lampy są umieszczone na suficie, a czujniki na podłodze.



Rys. 3.3.14. Okno konfiguracyjne.

```

private void drawAxis(int x, int y, Graphics2D g2d) {
    g2d.setColor(Color.black);
    g2d.setStroke(new BasicStroke( width: 1));
    LightSource current=room.getCurrentLightSource();
    int newX=evaluateX(current.getX(),current.getY());

    g2d.drawLine(newX, current.getZ(), x, y);

    // optic axis and line perpendicular to optic axis and containing point (x,y)
    double a1, a2, b2;
    if (newX == x) {
        a2 = 1;
        b2 = -x;
    } else {
        a1 = (double) (current.getZ() - y) / (double) (newX - x);
        a2 = (-1) / a1;
        b2 = y - a2 * x;
    }
}

// equation of circle with radius equal to r=tan(alfa/2)*distance between (x,y) and (currentLightSource.x,currentLightSource.y)
double height = Math.sqrt(Math.pow((newX - x), 2) + Math.pow((current.getZ() - y), 2));
double radius = Math.tan(Math.toRadians(current.getAngle() / 2)) * height;

double a, b, c;
double delta;
a = 1 + Math.pow(a2, 2);
b = (-2) * x + 2 * a2 * b2 - 2 * a2 * y;
c = Math.pow(x, 2) + Math.pow(b2, 2) - 2 * b2 * y + Math.pow(y, 2) - Math.pow(radius, 2);
delta = Math.pow(b, 2) - 4 * a * c;

double x1, x2, y1, y2;
x1 = ((-1) * b - Math.sqrt(delta)) / (2 * a);
x2 = ((-1) * b + Math.sqrt(delta)) / (2 * a);
y1 = a2 * x1 + b2;
y2 = a2 * x2 + b2;

g2d.drawPolygon(new int[]{newX, (int) x1, (int) x2}, new int[]{current.getZ(), (int) y1, (int) y2}, 3);
}

// evaluating point's x coordinate in new coordinate system - OX is now the optic axis of cone and OY is OZ
private int evaluateX(int x,int y){
    int newX;
    LightSource current=room.getCurrentLightSource();
    double x1=current.getX();
    double x2=current.getAxisX();
    double y1=current.getY();
    double y2=current.getAxisY();
    double A=(-1)/((y1-y2)/(x1-x2));
    double B=1;
    double C=(-1)*(y1-A*x1);
    A=-A;
    newX= (int) (Math.abs(A*x+B*y+C)/Math.sqrt(Math.pow(A,2)+Math.pow(B,2)));
    return newX;
}

```

Rys. 3.3.14. Fragment klasy XZAxisPopup.

- Sensor

Dodany sensor odznacza się kolorem zielonym. Po jego dodaniu na liście z lewej strony pojawia się komunikat o konieczności dokończenia konfiguracji (w tym przypadku jest to tylko nadanie nazwy). Wykrywane natężenie oświetlenia każdego sensora jest odświeżane po każdym dodaniu nowego źródła światła albo zmodyfikowaniu któregoś z istniejących. Graficzne przedstawienie sensorów jest widoczne na rysunku 3.3.10. Klasa realizuje przedstawione wcześniej obliczenia.


```

static Sensor getAtSensor(int x, int y, ArrayList<Sensor> sensors) {
    Sensor current;
    for (int i = 0; i < sensors.size(); i++) {
        current=sensors.get(i);
        if (Math.pow((x - current.getX()),2) + Math.pow((y - current.getY()),2) <= Math.pow((current.getRadius()+10),2)) return sensors.get(i);
    }
    return null;
}

boolean isInsideTheCone(LightSource lightSource){
    int[] point=new int[]{getX(),getY(),getZ()};
    int[] lineStart=new int[]{lightSource.getX(),lightSource.getY(),lightSource.getZ()};
    int[] lineEnd=new int[]{lightSource.getAxisX(),lightSource.getAxisY(),lightSource.getAxisZ()};
    double distance= distanceBetweenPointAndLine(point,lineStart,lineEnd);
    int[] pointOnLine=crossPoint(point,lineStart,lineEnd);
    double radius=lightSource.countConeRadiusWithGivenPoint(pointOnLine);
    //System.out.println("d r; "+distance+" "+radius);
    return (distance<=radius && distance>=0);
}

void countIlluminance(ArrayList<LightSource> lightSources){
    double tempIlluminance=0;
    double r=0;
    double I=0;
    double cos=0;
    for(int i=0;i<lightSources.size();i++){
        LightSource temp=lightSources.get(i);
        if(isInsideTheCone(temp)) {
            System.out.println(getName() + " jest");
            // illuminance = (I/r^2)*cos(alfa)
            r = (Math.sqrt(Math.pow((getX() - temp.getX()), 2) + Math.pow((getY() - temp.getY()), 2) + Math.pow((getZ() - temp.getZ()), 2))) / 100;
            I = (temp.getLuminousFlux() / (2 * Math.PI * (1 - Math.cos(temp.getAngle() / 2))));
            cos = Math.abs((temp.getZ() - getZ())) / r;
            tempIlluminance += (I / Math.pow(r, 2) * cos);
            double tempI=(I / Math.pow(r, 2) * cos);
            System.out.println("r I cos illuminance: " + r + " " + I + " " + cos + " " + tempI);
        }
    }
    setIlluminance(tempIlluminance);
    sensorConfigurationPopup.getIlluminance().setText(String.valueOf(illuminance));
    System.out.println(illuminance);
    System.out.println("=====");
}

public static double distanceBetweenPointAndLine(int[] point, int[] lineStart, int[] lineEnd){
    int[] vector1 = new int[3];
    int[] vector2 = new int[3];
    int[] TotalThing = new int[3];

    vector1[0] = lineEnd[0]-lineStart[0];
    vector1[1] = lineEnd[1]-lineStart[1];
    vector1[2] = lineEnd[2]-lineStart[2];

    vector2[0] = point[0]-lineStart[0];
    vector2[1] = point[1]-lineStart[1];
    vector2[2] = point[2]-lineStart[2];

    TotalThing[0] = (vector1[1]*vector2[2] - vector1[2]*vector2[1]);
    TotalThing[1] = (vector1[2]*vector2[0] - vector1[0]*vector2[2]);
    TotalThing[2] = (vector1[0]*vector2[1] - vector1[1]*vector2[0]);

    double distance = (double) ((Math.sqrt(Math.pow(TotalThing[0],2)+Math.pow(TotalThing[1],2)+Math.pow(TotalThing[2],2))) /
        Math.sqrt(Math.pow(vector1[0],2)+Math.pow(vector1[1],2)+Math.pow(vector1[2],2)));
    return distance;
}

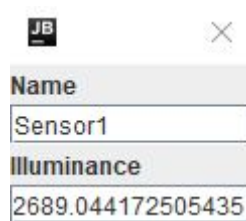
// returns coordinates of a crosspoint between optic axis and the line perpendicular to the optic axis and containing given point (double[] point)
public int[] crossPoint(int[] point,int [] lineStart,int [] lineEnd){
    int A,B,C;
    double x;
    A=lineEnd[0]-lineStart[0];
    B=lineEnd[1]-lineStart[1];
    C=lineEnd[2]-lineStart[2];
    x= (point[2]*C-lineStart[0]*A+A*point[0]-lineStart[1]*B+point[1]*B-lineStart[2]*C)/(Math.pow(A,2)+Math.pow(B,2)+Math.pow(C,2));
    return new int[]{(lineStart[0]+(int)(A*x)),(lineStart[1]+(int)(B*x)),(lineStart[2]+(int)(C*x))};
}

```

Rys. 3.3.15. Fragment klasy Sensor.

- **SensorConfigurationPopup**

Jest to okno konfiguracji źródła światła. Po jego zamknięciu ustawiają się nazwa, kąt świecenia i strumień świetlny. Można do niego wrócić w dowolnym momencie tworzenia modelu.



Rys. 3.3.16. Okno konfiguracyjne.

Implementacja tej klasy jest analogiczna do implementacji klasy LightSourceConfigurationPopup.

- **Room**

Jest to klasa spajająca funkcjonalność wszystkich klas w folderze. Umożliwia ona narysowanie poszczególnych elementów i usprawniająca, a w większości przypadków umożliwiającą przekazywanie wartości między klasami w całym projekcie. Room odpowiada za pojawienie się odpowiednich okien i ustawienie odpowiednich wartości w odpowiednim czasie. Są w nim listenery wykrywające ruch i klikanie myszy.

Tutaj jest także zaimplementowane tworzenie heat mapy. Ma ona dostępne rozróżnienie 6 poziomów natężenia oświetlenia. Większa liczba poziomów zaburzałaby czytelność wizualizacji, a większa dokładność informacji nie jest potrzebna użytkownikowi. Heat mapa uwzględnia to, że okno nie jest zamieszczone przy samej podłodze, ale na pewnej wysokości. Dlatego też pasek podłogi tuż przy ścianie nie będzie najmocniej oświetlony. Im dalej od okna, tym natężenie oświetlenia mniejsze, czemu odpowiadają coraz zimniejsze kolory. Można wizualizować światło dochodzące z wielu okien jednocześnie.

```

public Room() {
    setPreferredSize(new Dimension( width: 950, height: 800));
    setBorder(BorderFactory.createLineBorder(Color.black));
    setBackground(Color.WHITE);
    setLayout(null);

    addMouseListener((MouseAdapter) mousePressed(e) + {
        if (!roomIsPainted) {
            if (paintWall) {
                currentWall = addWall(e);
                if (walls.size() > 0 && currentWall.getX() == walls.get(0).getX() && currentWall.getY() == walls.get(0).getY()) {
                    roomIsPainted = true;
                    currentWall = null;
                    paintWall = false;
                } else walls.add(currentWall);
                repaint();
            }
        }
        if (paintWindow) { // create new or take the last one from table
            if (windowEnd == false) {
                Window window = new Window();
                currentWindow = setWindow(e, window, windowEnd: false);
                windowEnd = true;
                windows.add(currentWindow);
            }
            else {
                currentWindow = setWindow(e, windows.get(windows.size() - 1), windowEnd: true);
                windowEnd = false;
                paintWindow = false;
                windows.add(currentWindow);
            }
            //currentWindow = null;
            repaint();
        }

        if (paintSensor) {
            sensors.get(sensors.size() - 1).setPlaced(true);
            sensors.get(sensors.size() - 1).setSensorConfigurationPopup(new SensorConfigurationPopup(sensors.get(sensors.size() - 1)));
            paintSensor = false;
            sensors.get(sensors.size() - 1).countIlluminance(lightSources);
            repaint();
            SensorList.addItem(sensors.get(sensors.size() - 1).getName() + ": " + sensors.get(sensors.size() - 1).getIlluminance());
        }
    }
}

```

```

if (paintXYAxis) {
    if (currentLightSource == null) {
        currentLightSource = getAtLightSource(e.getX(), e.getY(), lightSources);
        if (currentLightSource != null) currentLightSource.setColor(Color.RED);
        else paintXYAxis = false; // no lightSource detected in the area
    } else {
        currentLightSource.setAxisX(e.getX());
        currentLightSource.setAxisY(e.getY());
        currentLightSource.setColor(Color.YELLOW);
        paintXYAxis = false;
        currentLightSource = null;
    }

    repaint();
}

if (paintXZAxis) {
    currentLightSource = getAtLightSource(e.getX(), e.getY(), lightSources);
    if (currentLightSource != null) {
        currentLightSource.setColor(Color.RED);
        XZAxisPopup popup = new XZAxisPopup( room: Room.this);
        repaint();
        LightSourceList.updateAllItems();
        SensorList.updateAllItems();
    }
}

if (e.getClickCount() == 2) {
    currentLightSource = getAtLightSource(e.getX(), e.getY(), lightSources);
    if (currentLightSource != null) {
        currentLightSource.getLightSourceConfigurationPopup().setVisible(true);
        currentLightSource = null;
    } else {
        currentSensor = getAtSensor(e.getX(), e.getY(), sensors);
        if (currentSensor != null) {
            currentSensor.getSensorConfigurationPopup().setVisible(true);
            currentSensor = null;
        }
    }
}

if (e.getClickCount() == 1) {
    for (int i = 0; i < sensors.size(); i++) sensors.get(i).countIlluminance(lightSources);
    System.out.println("*****");
}

```

```

addMouseMotionListener((MouseAdapter) mouseMoved(e) → {
    if (paintWall && currentWall != null) {
        currentWall = addWall(e);
        repaint();
    }
    if(paintWindow && currentWindow!=null){
        if(windowEnd==true) {
            currentWindow = addWindow(e, windowEnd); // do rysowania na bieżąco
        }
        repaint();
    }

    if (paintSensor) {
        sensors.get(sensors.size() - 1).setX(e.getX() - 5); // -5 so that sensor would be visible
        sensors.get(sensors.size() - 1).setY(e.getY() - 5);
        repaint();
    }

    if (paintLightSource) {
        lightSources.get(lightSources.size() - 1).setX(e.getX() - 5);
        lightSources.get(lightSources.size() - 1).setY(e.getY() - 5);
        repaint();
    }

    if (paintXYAxis && currentLightSource != null) {
        currentLightSource.setAxisX(e.getX());
        currentLightSource.setAxisY(e.getY());
        repaint();
    }
});

```

Rys. 3.3.17. Fragmenty klasy Room.

3.3.4. LightSourceList oraz SensorList

Klasy te mają analogiczną implementację, dlatego zostaną omówione razem. Umożliwiają one stworzenie list elementów wyświetlanych na bieżąco podczas tworzenia i modyfikowania modelu. Sprawdzają też, czy konfiguracja elementu jest kompletna.

Implementacja na przykładzie LightSourceList:

```
public class LightSourceList extends JList{
    static DefaultListModel data = new DefaultListModel();

    public LightSourceList(){
        super(data);
        setPreferredSize(new Dimension( width: 500, height: 400));
    }

    public static void addItem(String Item) {
        if (Item.contains("null")) {data.addElement("Incomplete Configuration"); }
        else {data.addElement(Item); }
    }

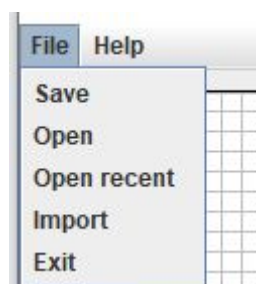
    public static void updateAllItems() {
        ArrayList<LightSource> sources = Room.getLightSources();
        for (int i=0; i<sources.size(); i++) {
            LightSource source = sources.get(i);
            if (source.getName() == null) {data.set(i, "LightSource " + i); }
            else {data.set(i, source.getName() + " : " + source.getLuminousFlux()); }
        }
    }
}
```

Rys. 3.3.18. Klasa LightSourceList.

3.3.5. Menu

Ten folder zawiera menu składające się z:

- File Menu
- Help Menu

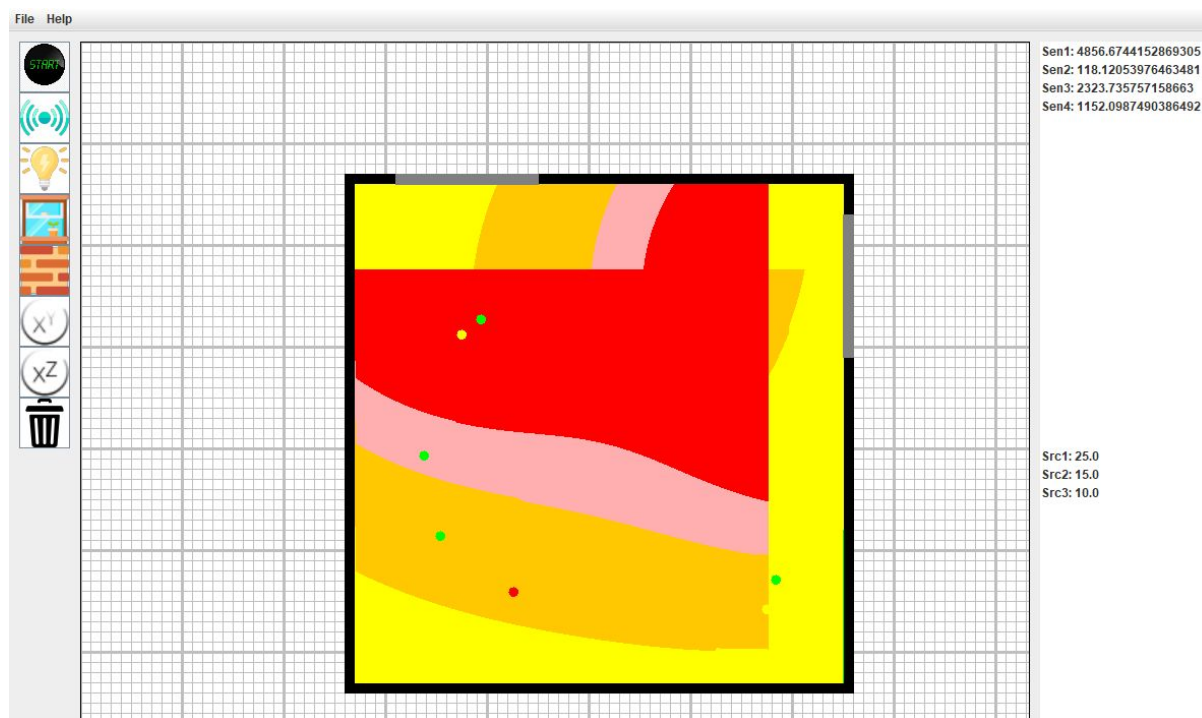


Rys.3.3.19. Rozwinięte menu.

4. Przykład działania

Zamodelowano pokój z dwoma oknami, trzema lampami i czterema czujnikami rozmieszczonymi w całym pokoju. Warunki pogodowe ustalono na: 123. dzień roku (początek maja), południe, przeciętny słoneczny dzień (20000 lx). Na mapie widać, że

światło z obu okien nakłada się, dodatkowo pokój rozjaśniają lampy. Na czerwono zaznaczona została obecnie konfigurowana lampa. Wartości odczytywane przez sensory i nadane lampom można przeczytać z prawej strony.



Rys. 3.4.1. Działanie aplikacji.

5. Wnioski

Działanie aplikacji jest zgodne z oczekiwaniami. Wartości szczytywane przez sensory są zgodne z obliczeniami i zmieniają się po dodaniu kolejnej lampy albo zmianie jej konfiguracji. Heat mapa koloruje się prawidłowo. Aplikacja może być bardzo pomocnym narzędziem przy projektowaniu pokoju, szczególnie że umożliwia sprawdzenie zachowania światła w różnych warunkach pogodowych, o różnych porach dnia i roku, dzięki czemu można podejść do planowania z pełnym zestawem faktów. Użytkownik jest w stanie tak zmodyfikować ustawienia lamp, żeby dopasować je do swoich preferencji. Komunikacja między oświetleniem a sensorami jest doskonałym wstępem do stworzenia systemu automatycznie regulującego się oświetlenia, dzięki któremu użytkownik nie sterowałby bezpośrednio lampami, a tylko zadaną jasnością.