

4. Dokumentacja projektu

4.1. Opis systemu

Opisane w niniejszej pracy elementy implementacyjne, po połączeniu z modułami modelu danych oraz optymalizacji, tworzą w pełni funkcjonalną aplikację *InterLight*, wspierając projektowanie oświetlenia wewnętrznego dla zadanej sceny i dostępnych źródeł światła.

4.1.1. Profil użytkownika

Przewidziany jest jeden, domyślny profil użytkownika. Osoby korzystające z aplikacji powinny się cechować:

- umiejętnością obsługi komputera, w tym typowych programów, w stopniu co najmniej lekko zaawansowanym,
- podstawową wiedzę dziedzinową w zakresie projektowania oświetlenia wewnętrznego.

4.1.2. Środowisko użytkowania

Program będzie wykorzystywany do projektowania oświetlenia wewnętrznego i porównywania ze sobą możliwych wariantów rozwiązania. Użytkowany będzie w warunkach domowych lub biurowych na komputerach osobistych (*Personal Computer* – PC), przez osoby zainteresowane tym zagadnieniem.

4.2. Analiza funkcjonalna

4.2.1. Wymagania funkcjonalne

Kluczowe funkcjonalności, które powinna zapewniać aplikacja to:

- a) graficzne stworzenie sceny w postaci rzutu z góry, poprzez:
 - podanie identyfikatora projektu oraz wymiarów obszaru roboczego,
 - narysowanie przy pomocy myszki ścian pokoju, z możliwością wyrównania ich do kątów prostych lub rozciągnięciu na maksymalny wymiar obszaru roboczego, oraz określenie jego wysokości,

- dodanie obiektów: prostopadłościanów oraz walców eliptycznych i edycję ich przy pomocy myszki lub panelu sterującego,
 - dodanie punktów świetlnych (miejsc w których potencjalnie mogą być umieszczone źródła światła), pojedynczo lub poprzez zdefiniowanie siatki takich punktów;
- b) edycję także poprzez bezpośrednią manipulację sceną w formacie JSON, dostarczanego przez moduł modelu danych;
- c) wybranie z listy opraw oświetleniowych (definiowanych przez dane fotometryczne z pliku w formacie IES, opisanego w sekcji 4.6.7) tych, które mają być brane pod uwagę przez moduł optymalizacji;
- d) dodawanie do listy nowej oprawy przez wskazanie zewnętrznego pliku IES;
- e) możliwość uzyskania prostej pomocy w programie na temat jego obsługi;
- f) możliwość zapisu lub wczytania projektu sceny w dowolnym momencie działania programu;
- g) po wprowadzeniu danych wejściowych, uruchomienie analizy modułu optymalizacji, której wynikiem jest lista proponowanych rozwiązań, z możliwością sortowania ich pod względem wskaźników ekonomicznych oraz wizualizacji:
- przy pomocy biblioteki Radiance,
 - w programie, przez oznaczenie obszarów niedoświetlonych, powodujących niespełnianie norm.

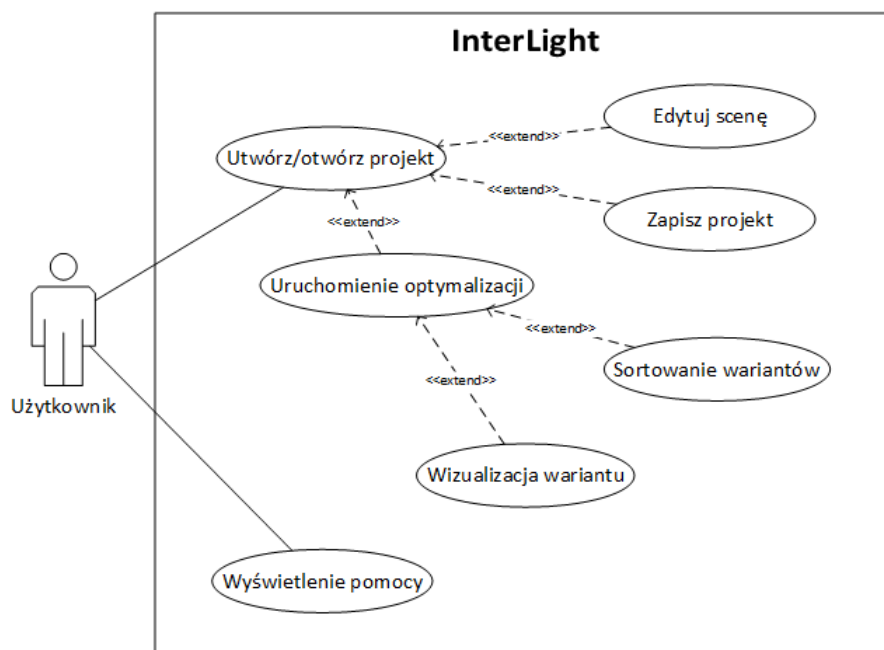
4.2.2. Wymagania нефunkcjonalne

Interfejs użytkownika powinien spełniać następujące wymagania, opisane szczegółowo w podrozdziale 3.1:

- spójność,
- podział na obszary,
- ergonomia,
- intuicyjność,
- responsywność,
- niezawodność i szybkość działania.

Wymagania programowe i sprzętowe dotyczące uruchomienia i korzystania z aplikacji *InterLight* obejmują:

- komputer klasy PC,



Rysunek 4.1: Diagram przypadków użycia

- system operacyjny Windows lub GNU/Linux (działanie na systemie OS X jest możliwe po dodaniu do aplikacji wersji Radiance skompilowanej dla tego systemu, co nie było przedmiotem pracy),
- środowisko Java JRE w wersji 8 update 40 lub nowszej.

Ze względu na pracę modułu optymalizacji, zaleca się posiadanie minimum 1 GB pamięci RAM oraz procesora klasy Pentium 4 lub nowszej.

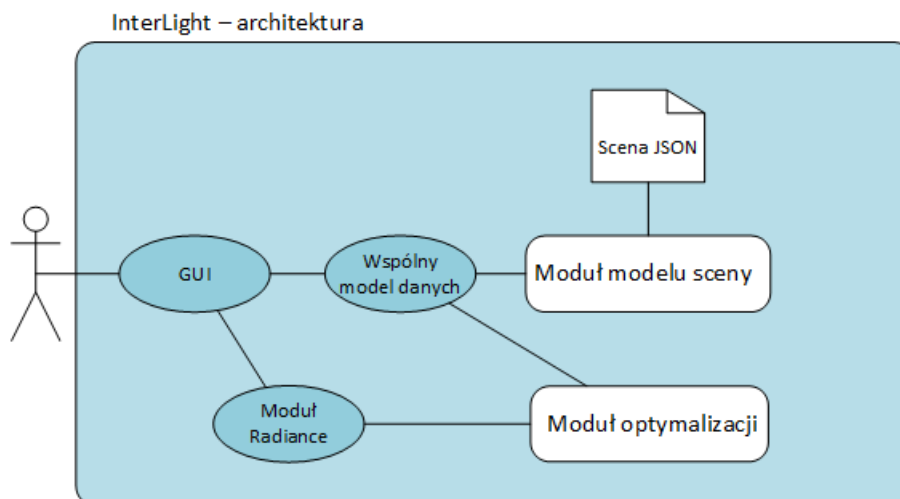
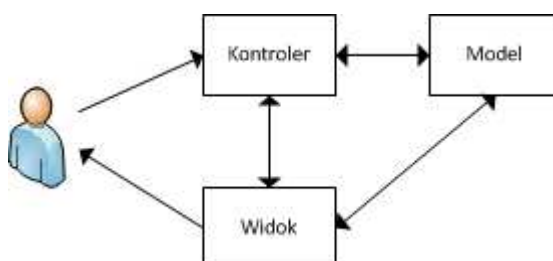
4.2.3. Przypadki użycia

Diagram przypadków użycia, obejmujący funkcjonalności z podrozdziału 4.2.1, znajduje się na rysunku 4.1.

4.3. Architektura programu

Program zrealizowano w postaci samodzielnej aplikacji klienckiej, ze względu na środowisko użytkownika, opisane w podrozdziale 4.1.2 oraz brak konieczności pozyskiwania dodatkowych danych z sieci. Decyzja ta wiąże się z koniecznością dostarczenia użytkownikowi pełnej aplikacji wraz z bibliotekami i strukturą folderów oraz wykonywania wszelkich obliczeń na jego komputerze, co może oznaczać znaczne obciążenie i długi czas oczekiwania w przypadku dużej sceny lub wielu możliwych źródeł światła.

Aplikacja składa się z powiązanych ze sobą modułów, stanowiących finalnie jeden plik wykonywalny. W trakcie działania, program korzysta z dodatkowej struktury folderów w swoim katalogu źródłowym w celu zapisywania projektów i odczytywania źródeł światła w formie plików IES.

Rysunek 4.2: Zarys architektury *InterLight*

Rysunek 4.3: Wzorzec MVC

Poglądowy schemat przedstawiający zarys architektury *InterLight* znajduje się na rysunku 4.2. Części implementowane w niniejszej pracy zostały oznaczone kolorem niebieskim.

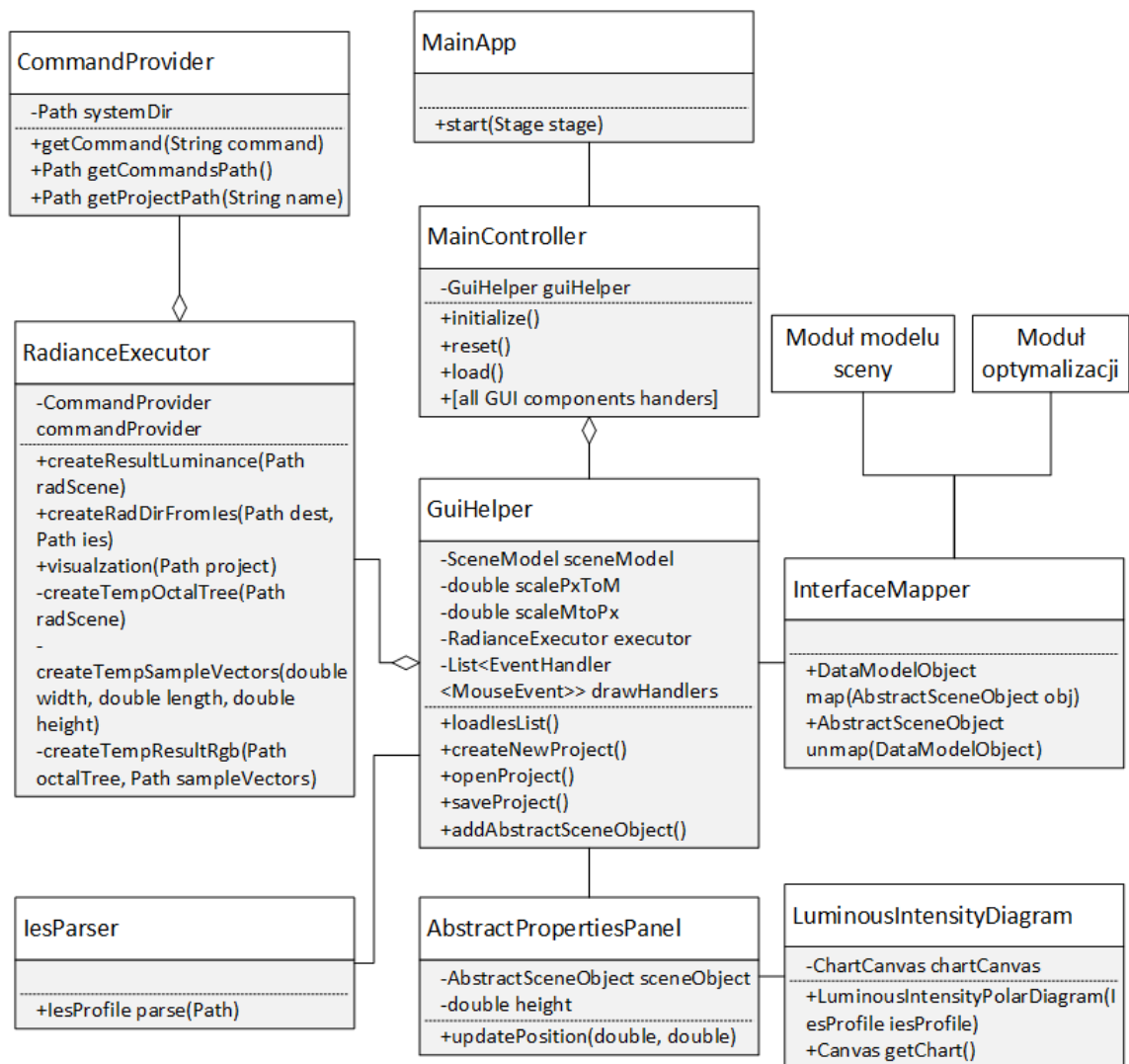
4.4. Wzorzec MVC

Jako podstawowy wzorzec architektoniczny dla programu wybrano Model-Widok-Kontroler (*MVC*, *Model-View-Controller*), przedstawiony na rysunku 4.3. Jest on jednym z najpopularniejszych wzorców dla aplikacji z graficznym interfejsem użytkownika, na jego bazie funkcjonuje *JavaFX*, będąca podstawą architektury GUI. Składa się z następujących warstw:

- **model** - reprezentacja problemu po stronie aplikacji w postaci klas oraz dostęp do nich;
- **widok** - wyświetlanie danych modelu w interfejsie graficznym;
- **kontroler** - obsługa akcji użytkownika, zmiana stanu modelu, odświeżanie widoku.

4.5. Diagram klas

Diagram klas został przedstawiony na rysunku 4.4.

Rysunek 4.4: Diagram klas *InterLight*

4.5.1. Odpowiedzialności

MainApp – główna klasa aplikacji, uruchamia okno programu.

MainController – obsługuje zdarzenia graficznego interfejsu użytkownika.

GuiHelper – klasa obsługująca komponenty graficznego interfejsu użytkownika.

InterfaceMapper – klasa mapująca obiekty sceny na wspólny model danych, będący warstwą połączeniową z pozostałymi modułami.

IesParser – parser pozyskujący dane z tekstowych plików IES.

RadianceExecutor – wykonuje polecenia biblioteki Radiance.

CommandProvider – dostarcza pełnych ścieżek do plików wykonywalnych i zasobów dla danego miejsca uruchomienia programu.

AbstractPropertiesPanel – panel właściwości aktualnie wybranego obiektu sceny.

LuminousIntensityDiagram – dostarcza komponent wyświetlający wykres krzywych rozsyłu światłości dla aktualnie wybranego źródła światła.

4.6. Technologie i narzędzia

4.6.1. Java

Główny język, w którym zaimplementowano niniejszą pracę to Java firmy Oracle w wersji 8. Jest to obiektowy język programowania, jeden z najpopularniejszych na świecie [13], powstały w nieistniejącej już firmie Sun Microsystems w 1995 roku. Kod źródłowy kompilowany jest do kodu bajtowego, uruchamianego na maszynie wirtualnej Java (*JVM - Java Virtual Machine*). Składnia i słowa kluczowe bazują głównie na języku C++.

Najważniejsze cechy języka Java to: automatyczne zarządzanie dynamicznie przydzieloną pamięcią (*garbage collection*), wielowątkowość, niezależność od platformy i bezpieczeństwo.

Do stworzenia programu będącego tematem pracy wybrano język Java głównie ze względu na wieloletnie doświadczenie praktyczne autora, dużą dostępność dodatkowych bibliotek i możliwość pracy na dowolnym systemie operacyjnym. Są to aspekty znacznie ułatwiające pracę programistyczną, co pozwala na skupienie się na architekturze i funkcjonalności aplikacji.

Język Java udostępniany jest na licencji GNU GPL.

4.6.2. JavaFX

W celu zbudowania graficznego interfejsu użytkownika zastosowano technologię JavaFX. Jest to aktualnie wiodąca technologia do tworzenia graficznych interfejsów użytkownika, w tym RIA (*Rich*

Internet Application), w pakiecie JDK (*Java Development Kit*). Zastąpiła w tym względzie mniej funkcjonalną, starszą bibliotekę Swing.

Początkowo, JavaFX była wydawana jako osobny pakiet, a jej głównym komponentem był skryptowy język deklaratywny JavaFX Script, wcześniej znany pod nazwą F3 (*Form Follows Function*), kompilowany do kodu bajtowego Java. Jego składnia była mocno odmienna od znanego, chociażby z biblioteki Swing, wołania metod w celu konstrukcji obiektów i zmiany ich właściwości.

Od wersji 2.0, wydanej w 2011 roku, zrezygnowano z dodatkowego języka skryptowego i zdecydowano, że całą logikę implementować będzie się za pomocą Java, a interfejs w deklaratywnym języku FXML, bazującym na XML.

Graficzny interfejs aplikacji w tej technologii składa się z następujących elementów:

- pliku FXML, opisującego strukturę i komponenty,
- kontrolera w języku Java, definiującego zachowania i akcje komponentów,
- opcjonalnego pliku CSS (*Cascading Style Sheet*), określającego szczegółowy wygląd komponentów, jeśli ma być inny niż domyślny.

Do tworzenia aplikacji wykorzystano też program JavaFX Scene Builder, pomocnicze narzędzie do tworzenia plików FXML za pomocą metody WYSIWYG (*What You See Is What You Get*), ułatwiające ustalanie położenia komponentów.

4.6.3. Apache Maven

Narzędziem do automatyzacji budowy aplikacji Java był Maven, produkt fundacji Apache, wywodzący się z projektu Jakarta.

Umożliwia zarządzanie cyklem życia projektu, wspiera architekturę modułową i zarządza zewnętrznymi zależnościami z centralnego repozytorium. Twórcy kierują się strategią *Convention Over Configuration*, czyli minimalizacją niezbędnej konfiguracji na rzecz ogólnie przyjętej konwencji. Centralnym punktem modułu dla Maven jest POM (*Project Object Model*), plik XML, zawierający konfigurację. Rozszerzalność o wtyczki (*plug-in*) zapewnia wsparcie dla wielu różnych zastosowań i technologii.

Maven, jak inne projekty fundacji Apache, rozpowszechniany jest na otwartej licencji *Apache License*.

4.6.4. Git

Systemem kontroli wersji dla projektu *InterLight* był Git. Został stworzony w 2005 roku przez Linusa Torvaldsa do rozwoju jądra Linuksa.

Jest to system rozproszony, gdzie nacisk został położony na szybkość działania. Posiada wszystkie typowe funkcjonalności systemów kontroli wersji: ochrona przed błędami w repozytorium, obsługa gałęzi oraz łączenie zmian. Git jest, obok SVN, najczęściej używanym narzędziem tego typu, wykorzystującym go znane projekty, takie jak GNOME, GIMP, Ruby on Rails czy jQuery.

System Git jest udostępniany na licencji GNU GPL.

4.6.5. Eclipse IDE

Kod aplikacji napisano przy pomocy programu Eclipse IDE, rozwijanego przez Fundację Eclipse, założoną przez IBM w 2004 roku.

Eclipse to framework do tworzenia rozbudowanych aplikacji, rozszerzalnych za pomocą wtyczek. Najpopularniejszą jego implementacją jest właśnie zintegrowane środowisko programistyczne dla języka Java. Posiada wszystkie standardowe funkcje dla takiego typu programu, podobnie jak konkurencyjne, popularne środowiska NetBeans i IntelliJ IDEA. Wybór podyktowany był jedynie przyzwyczajeniem i osobistą preferencją.

Udostępniany jest na licencji otwartego oprogramowania EPL (*Eclipse Public License*).

4.6.6. Radiance

Radiance [14] to pakiet programów w języku C do analizy i wizualizacji oświetlenia, rozwijany przez laboratorium w Berkeley (*Lawrence Berkeley National Laboratory*).

Pozwala między innymi wyliczyć natężenie oświetlenia w określonych punktach dla zadanej w pliku tekstowym sceny, a także wizualizować ją. Jednym z głównych celów *InterLight* jest wykorzystanie właśnie tych funkcjonalności.

Radiance wykorzystuje ulepszoną i zmodyfikowaną wersję algorytmu śledzenia promieni, opisanego w podrozdziale 2.2.

Pakiet rozpowszechniany jest na zasadach własnej licencji (*The Radiance Software License*), darmowej i otwartej, wymagającej jedynie podania informacji o twórcach i treści licencji.

4.6.7. IES

IES [15] jest formatem plików tekstowych zaproponowanym przez organizację IESNA (*Illuminating Engineering Society of North America*) do opisu danych fotometrycznych, czyli natężenia światła w poszczególnych punktach siatki w kształcie sfery oraz geometrii dystrybucji światła na zewnątrz oprawy oświetleniowej (bryła fotometryczna).

Takie dane mają znaczącą przewagę nad standardowymi źródłami światła w programach do projektowania i grafiki 3D, gdyż nie ma tam miejsca na zniekształcenia spowodowane na przykład szkłem żarówki, czy powłoką refleksyjną stosowaną często w żarówkach halogenowych.

Plik IES ma rozszerzenie *.ies i zawiera sekcję metadanych, definicję podstawowych parametrów źródła oraz macierz wartości natężenia światła. Producenci profesjonalnego wyposażenia oświetleniowego udostępniają takie pliki dla swoich produktów.

W aplikacji *InterLight* za ich pomocą definiuje się, dostępne dla modułu optymalizacji, źródła światła, ze względu wsparcia obsługi tego formatu przez pakiet Radiance.

Listing 4.1: Fragment InterLight Maven (*parent*)

```
1 <project ...>
2   <groupId>pl.edu.agh.kis</groupId>
3   <artifactId>interlight</artifactId>
4   <version>1.0.0</version>
5   <packaging>pom</packaging>
6   <name>InterLight</name>
7
8   <modules>
9     <module>../interlight-common-datamodel</module>
10    <module>../interlight-model</module>
11    <module>../interlight-radiance</module>
12    <module>../interlight-opt</module>
13    <module>../interlight-interface</module>
14  </modules>
15
16  <dependencies>
17    ...
18  </dependencies>
19 </project>
```

4.6.8. JFreeChart

JFreeChart to rozwijana od 2000 roku biblioteka do rysowania wykresów w aplikacjach Java, od niedawna zapewniająca też pełne wsparcie dla JavaFX. Wykorzystano ją do wyświetlania krzywych rozsyłu światłości.

Udostępniania jest na licencji LGPL.

4.7. Implementacja

4.7.1. Moduły Maven

Architektura projektów Maven została połączona wspólnym rodzicem (*parent*), który nie zawiera implementacji, a jedynie plik POM, ze wspólnymi zewnętrznymi zależnościami oraz wyszczególnionymi modułami aplikacji, jak na listingu 4.1.

Wykonując cele cyklu życia projektu Maven, można budować wszystkie moduły i całą aplikację jednocześnie. W niniejszej pracy zaimplementowano:

- **interlight-interface** – główny moduł zawierający graficzny interfejs użytkownika i konfigurację budowania pliku wykonywalnego,
- **interlight-common-datamodel** – wspólne dla wszystkich projektów klasy modelu danych,

Listing 4.2: MainApp.java

```
1 public class MainApp extends Application {
2
3     public static void main(String[] args) throws Exception {
4         launch(args);
5     }
6
7     public void start(Stage stage) throws Exception {
8         Locale.setDefault(Locale.US);
9         FXMLLoader loader = new FXMLLoader();
10        loader.setLocation(getClass().getResource("/fxml/main.fxml"));
11        Parent rootNode = (Parent) loader.load(getClass().getResourceAsStream(
12            "/fxml/main.fxml"));
13        Scene scene = new Scene(rootNode);
14        scene.getStylesheets().add("/styles/style.css");
15        stage.setTitle("InterLight");
16        stage.setScene(scene);
17        stage.show();
18    }
19 }
```

- **interlight-radiance** – moduł umożliwiający wykonywanie programów pakietu Radiance przez moduły interfejsu i optymalizacji.

4.7.2. Główna klasa InterLight

MainApp jest główną klasą projektu. Jej metoda main uruchamia się przy starcie pliku wykonywalnego. W przypadku aplikacji JavaFX rozszerza klasę Application z pakietu javafx.application i implementuje jej metodę start, która ładuje interfejs JavaFX i ładuje odpowiedni arkusz stylów CSS.

Implementację klasy głównej InterLight przedstawiono na listingu 4.2.

4.7.3. Budowanie aplikacji

Budowaniem pliku wykonywalnego aplikacji w technologii JavaFX można zarządzać przy pomocy wtyczki `javafx-maven-plugin` do narzędzia Apache Maven. Listing 4.3 przedstawia fragment pliku POM dla modułu interfejsu z jej konfiguracją, na którą składają się: wskazanie głównej klasy w parametrze `mainClass` oraz miejsca w cyklu życia projektu Maven, w którym budowanie ma się odbyć (`phase`, `goal`).

Listing 4.3: Fragment POM modułu interfejsu

```
1 <plugin>
2   <groupId>com.zenjava</groupId>
3   <artifactId>javafx-maven-plugin</artifactId>
4   <version>8.1.2</version>
5   <configuration>
6     <mainClass>pl.edu.agh.kis.interlight.fx.MainApp</mainClass>
7   </configuration>
8   <executions>
9     <execution>
10      <goals>
11        <goal>jar</goal>
12      </goals>
13      <phase>install</phase>
14    </execution>
15  </executions>
16 </plugin>
```

4.7.4. Architektura JavaFX

Zgodnie z opisem w sekcji 4.6.2, podstawową strukturę aplikacji w technologii JavaFX stanowią plik FXML oraz klasa kontrolera.

Plik FXML składa się, w ogólnym przypadku, kolejno z:

- deklaracji (nagłówka) pliku XML,
- importów pakietów, analogicznie jak w kodzie języka Java,
- drzewiastej struktury komponentów określającej układ.

Każdy komponent graficznego interfejsu użytkownika jest osobnym znacznikiem XML. Jego atrybuty mogą określać wygląd, położenie, domyślne wartości pól, identyfikator JavaFX i nazwę metody nasłuchującej w kontrolerze.

Fragment pliku `main.fxml` na listingu 4.4 pokazuje przykładowy element interfejsu w *InterLight*: pole tekstowe `TextField`, posiadające parametry: identyfikator `fx:id` (do identyfikacji w kontrolerze) oraz położenie w komponencie nadrzędnym `GridPane`. Wszystko znajduje się w głównym (*root*) elemencie `VBox`, który poza rozmiarem, posiada wymagane atrybuty: `xmlns:fx` określający przestrzeń nazw JavaFX oraz `fx:controller` specyfikujący klasę kontrolera (pełna nazwa pakietowa).

Klasa kontrolera `MainController`, pisana w języku Java, służy do obsługi zdarzeń, dodatkowych ustawień komponentów i zmiany widoków interfejsu. Odnoszenie się do elementów pliku FXML, stanowiących komponenty GUI, odbywa się za pomocą ich identyfikatorów i adnotacji `@FXML`. Fragment pliku kontrolera, przedstawiający obsługę opisanego wcześniej przykładowego komponentu, pokazano na listingu 4.5. Łącząc pole tekstowe z pliku FXML za pomocą adnotacji z polem `projectName`,

Listing 4.4: Fragment main.fxml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <?import javafx.scene.image.*?>
4 <?import javafx.scene.text.*?>
5 <?import javafx.geometry.*?>
6 <?import javafx.scene.control.*?>
7 <?import java.lang.*?>
8 <?import javafx.scene.layout.*?>
9
10 <VBox prefHeight="700.0" xmlns="http://javafx.com/javafx/8.0.40"
11       xmlns:fx="http://javafx.com/fxml/1"
12       fx:controller="pl.edu.agh.kis.interlight.fx.MainController">
13   <children>
14     ...
15     <GridPane hgap="10.0" maxHeight="200.0" maxWidth="200.0" vgap="10.0">
16       ...
17       <TextField fx:id="projectName" GridPane.columnIndex="1"
18                 GridPane.rowIndex="1" />
19       ...
20     </GridPane>
21     ...
22   </children>
23 </VBox>
```

Listing 4.5: Fragment MainController.java

```
1 public class MainController {  
2  
3     @FXML  
4     private TextField projectName;  
5  
6     @FXML  
7     private void initialize() {  
8         projectName.setText(guiHelper.createNewProjectName());  
9         ...  
10    }  
11  
12    ...  
13 }
```

można z niego korzystać w typowy dla języka Java sposób. W linii 8. pokazane jest ustawienie wartości pola tekstowego metodą `projectName.setText()`. Oznaczona adnotacją metoda `initialize()` jest standardowo wołana przez JavaFX przy inicjalizowaniu interfejsu.

4.7.5. Graficzny interfejs użytkownika

W większej części graficzny interfejs użytkownika został wykonany za pomocą podstawowych komponentów JavaFX. Aby jednak zapewnić spełnianie wszystkich wymagań z podrozdziału 4.2.2, zaimplementowano kilka niestandardowych rozwiązań. Przedstawiono je na listingu 4.6 na przykładzie pola określającego szerokość sceny. Analogicznie zaimplementowano obsługę pola określającego jej długość.

Zdecydowano się na stały rozmiar okna programu, w związku z tym wymiary sceny, za które odpowiedzialna jest para pól tekstowych, powinny być zachowane w ustalonej proporcji. Dodano więc obsługę zdarzenia zmiany szerokości, aby automatycznie modyfikowało długość, za pomocą dodania `ChangeListener` i implementując metodę `changed`.

Drugą ważną funkcjonalnością jest walidacja wszystkich edytowalnych pól w responsywny sposób, co osiągnięto również przez wyżej wspomniane zdarzenie. Przy każdej zmianie wartości pola tekstowego, rzutuje się ją na liczbę, na przykład przy pomocy metody `Double.parseDouble()`. W przypadku wystąpienia wyjątku `NumberFormatException`, na komponencie ustawiana jest pseudoklasa `error` dla arkusza stylów CSS.

4.7.6. Uruchamianie Radiance

Uruchamianie pakietu Radiance w module *interlight-radiance* realizowane jest przy pomocy dwóch klas.

Listing 4.6: Obsługa pola szerokości sceny

```
1 @FXML
2 private TextField sceneWidth;
3 @FXML
4 private TextField sceneLength;
5 @FXML
6 private Button btnNewScene;
7
8 @FXML
9 private void initialize() {
10     sceneWidth.textProperty().addListener(new ChangeListener<String>() {
11         @Override
12         public void changed(ObservableValue<? extends String> observable,
13             String oldValue, String newValue) {
14             sceneWidth.pseudoClassStateChanged(guiHelper.getErrorClass(), false);
15             btnNewScene.setDisable(false);
16             try {
17                 Double value = Double.parseDouble(newValue);
18                 sceneLength.setText(String.valueOf(value
19                     * GuiHelper.CANVAS_HEIGHT / GuiHelper.CANVAS_WIDTH));
20             } catch (NumberFormatException nfe) {
21                 sceneWidth.pseudoClassStateChanged(guiHelper.getErrorClass(),
22                     true);
23                 btnNewScene.setDisable(true);
24             }
25         }
26     });
27 }
```

Listing 4.7: Rozpoznawanie systemu operacyjnego

```
1 private String createSystemDir() {
2     String osName = System.getProperty("os.name").toLowerCase();
3     if (osName.contains("win")) {
4         return "win";
5     } else if (osName.contains("linux")) {
6         if (System.getProperty("os.arch").contains("64")) {
7             return "linux64";
8         } else {
9             return "linux32";
10        }
11    } else {
12        throw new UnsupportedOperationException("Unsupported OS");
13    }
14 }
```

Klasa `CommandProvider` dostarcza pełnych ścieżek do poleceń Radiance, które różnią się w zależności od systemu operacyjnego, gdyż Radiance został dołączony w formie skompilowanych plików wykonywalnych w języku C. Rozpoznawanie systemu odbywa się poprzez odczytywanie właściwości systemowych z klasy `java.lang.System`, co zaprezentowano na listingu 4.7.

Programy pakietu Radiance są wykonywane przez klasę `RadianceExecutor`. W znacznej większości są to aplikacje pracujące w środowisku konsoli. Ich uruchamianie umożliwia klasa `java.lang.ProcessBuilder`, która przyjmuje listę parametrów.

Przykład na listingu 4.8 to metoda `createResultLuminance`, uruchamiająca prosty program do konwersji rekordów z pliku tekstowego do innego pliku. Potrzebne jest więc przekierowanie standardowego wejścia i wyjścia na ścieżki będące argumentami metody, jak w liniach 9 i 10.

4.7.7. Wykres krzywych rozsyłu światłości

Tworzenie wykresów krzywych rozsyłu światłości jest możliwe dzięki danym z plików fotometrycznych IES. Ich parsowaniem zajmuje się klasa `IesParser` przy pomocy `java.util.Scanner`, która tworzy obiekt `IesParser`, zawierający między innymi wartości luminancji.

Wykres uzyskiwany jest dzięki bibliotece `JFreeChart`. Od najnowszej wersji 1.0.19 dostarcza komponentów typu `JavaFX`, gotowych do dodania do graficznego interfejsu użytkownika, jak na przykład `ChartCanvas`. Całość zaprezentowana jest na listingu 4.9. Najpierw tworzone są serie danych oraz obiekt renderujący, który definiuje dla nich sposoby wyświetlania. Następnie inicjowany jest `ChartCanvas` za pomocą standardowego diagramu polarnego `JFreeChart`, uzyskanego dzięki implementacji wzorca projektowego fabryki (*factory*).

Listing 4.8: Uruchamianie programu pakietu Radiance

```
1 public Path createResultLuminance(Path destination, Path rgb) throws IOException {
2     List<String> commands = new ArrayList<String>();
3     commands.add(commandProvider.createCommand(RadianceExecutor.COMMAND_RCALC));
4     commands.add("-e");
5     commands.add("$1=$1;$2=$2;$3=$3;$4=47.4*$4+120*$5+11.6*$6");
6     ProcessBuilder builder = new ProcessBuilder(commands);
7     builder.directory(new File(commandProvider.getCommandsPath()));
8     builder.redirectErrorStream(true);
9     builder.redirectInput(rgb.toFile());
10    builder.redirectOutput(destination.toFile());
11    Process process = builder.start();
12    try {
13        process.waitFor();
14    } catch (InterruptedException e) {
15        e.printStackTrace();
16    }
17    return destination;
18 }
```

4.7.8. Integracja

Integrację między komponentami zrealizowano przez wywoływanie metod API przy wykorzystaniu obiektów wspólnego modelu danych.

Moduł interfejsu posiada klasę `InterfaceMapper`, zawierającą statyczne metody potrafiące przekształcać obiekty ze wspólnego modelu danych do modelu graficznego interfejsu użytkownika i odwrotnie. Przykład dla klas zawierających ogólne dane sceny i projektu przedstawiono na listingu 4.10.

4.8. Użytkowanie aplikacji

4.8.1. Dostarczenie

Aplikacja *InterLight* dostarczana jest w postaci wykonywalnego pliku JAR oraz wymaganej, dodatkowej struktury folderów:

- *lib* – zawierający biblioteki, moduły, zależności zewnętrzne,
- *res* – katalog na pliki IES, wraz z kilkoma przykładowymi,
- *projects* – katalog na zapisane projekty, wraz z przykładem.

Program uruchamia się plikiem wykonywalnym *interlight-interface-1.0.0-jfx.jar* lub z poziomu konsoli poleceniem `java -jar interlight-interface-1.0.0-jfx.jar`.

Listing 4.9: Tworzenie wykresu krzywych rozsyłu światłości

```

1 XYSeriesCollection data = new XYSeriesCollection();
2 for (int i = 0; i < iesProfile.getHorizontalAngles().length; i++) {
3     if (iesProfile.getHorizontalAngles()[i] % 90 == 0) {
4         XYSeries series = new XYSeries("Series " + i);
5         for (int j = 0; j < iesProfile.getVerticalAngles().length; j++) {
6             series.add(iesProfile.getVerticalAngles()[j],
7                 iesProfile.getLumenValues()[i][j]);
8         }
9         data.addSeries(series);
10    }
11 }
12 DefaultPolarItemRenderer renderer = new DefaultPolarItemRenderer();
13 renderer.setSeriesShape(0, new Rectangle(1, 1), false);
14 renderer.setSeriesShape(1, new Rectangle(1, 1), false);
15 renderer.setSeriesShape(2, new Rectangle(1, 1), false);
16 renderer.setSeriesShape(3, new Rectangle(1, 1), false);
17
18 final JFreeChart chart = ChartFactory.createPolarChart(null, data, false,
19     false, false);
20 ((PolarPlot) chart.getPlot()).setRenderer(renderer);
21 chart.setBackgroundPaint(new java.awt.Color(244, 244, 244));
22 ChartCanvas chartCanvas = new ChartCanvas(chart) {
23     public void chartChanged(ChartChangeEvent event) {}
24 };

```

Listing 4.10: Transformacja obiektów

```

1 public static IScene map(SceneModel scene) {
2     return new IScene(scene.getName(), scene.getSceneWidthM(),
3         scene.getSceneLengthM());
4 }
5
6 public static SceneModel unmap(IScene iScene) {
7     SceneModel scene = new SceneModel();
8     scene.setName(iScene.getName());
9     scene.setSceneWidthM(iScene.getWidth());
10    scene.setSceneLengthM(iScene.getLength());
11    return scene;
12 }

```

4.8.2. Obsługa

Pracę z programem rozpoczyna możliwość stworzenia nowego projektu, przez podanie jego nazwy i wymiarów obszaru roboczego, które automatycznie zachowują stosunek 3:5 lub też otwarcia wcześniej zapisanego (rysunek 4.5).

Następnie przechodzi się do głównego ekranu programu, zawierającego obszar roboczy – widoku sceny w rzucie z góry. Po lewej stronie znajduje się panel kontrolny typu „akordeon”, którego zakładki oznaczają kolejne kategorie elementów projektu. Wybór następuje przez kliknięcie nagłówka. Gdy wybrana jest dana zakładka, tylko obiekty z nią związane będą aktywne w obszarze roboczym.

Pierwszą zakładką panelu kontrolnego jest pokój. Ściany są rysowane za pomocą myszki. Lewym przyciskiem można dodać kolejny punkt wielokąta, a prawym usunąć. Metodą „przeciągnij i upuść” (*drag and drop*) przesuwa się punkty w granicach obszaru roboczego. Dodatkowo w lewym panelu dostępne są opcje stworzenia ścian pokoju na pełny dostępny wymiar oraz wyrównywania kolejnych dodawanych punktów do kątów prostych (rysunek 4.6).

Po przejściu do drugiej zakładki po lewej stronie można dodawać obiekty: prostopadłościąny oraz walce eliptyczne (symbolizujące na przykład stoły i szafy) za pomocą przycisków i przemieszczać je podobnie jak wcześniejsze punkty lub też edytować parametry aktualnie zaznaczonego bezpośrednio w panelu (rysunek 4.7).

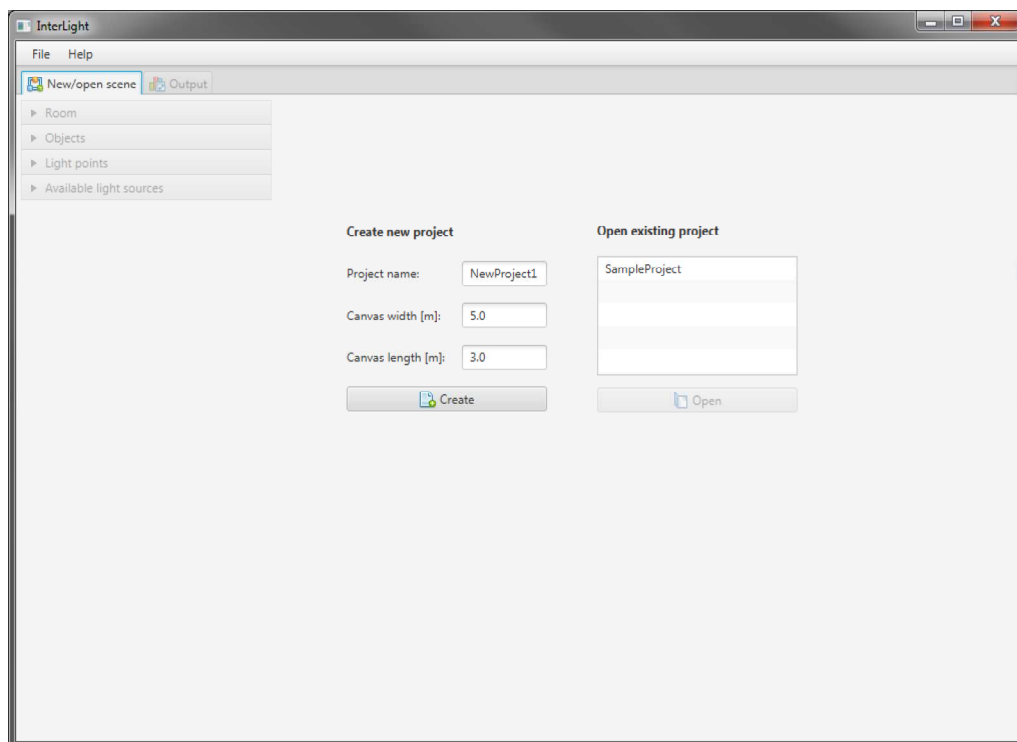
Punkty świetlne, czyli miejsca gdzie potencjalnie da się umieścić źródła światła, można dodawać na dwa sposoby, przy pomocy przycisków: pojedynczy punkt (rysunek 4.8) lub zdefiniowaną siatkę punktów (rysunek 4.9). Edycję i przemieszczanie wykonuje się podobnie jak obiekty z poprzedniego kroku.

Ostatnia zakładka panelu kontrolnego pozwala wybrać z listy te źródła światła, które mają być brane pod uwagę przy optymalizacji. Można też dodać nowe za pomocą przycisku, wskazując plik IES, który zostanie skopiowany do struktury katalogów programu. Po zaznaczeniu wyświetlany jest po lewej stronie podgląd wykresu krzywych rozsyłu światłości (rysunek 4.10).

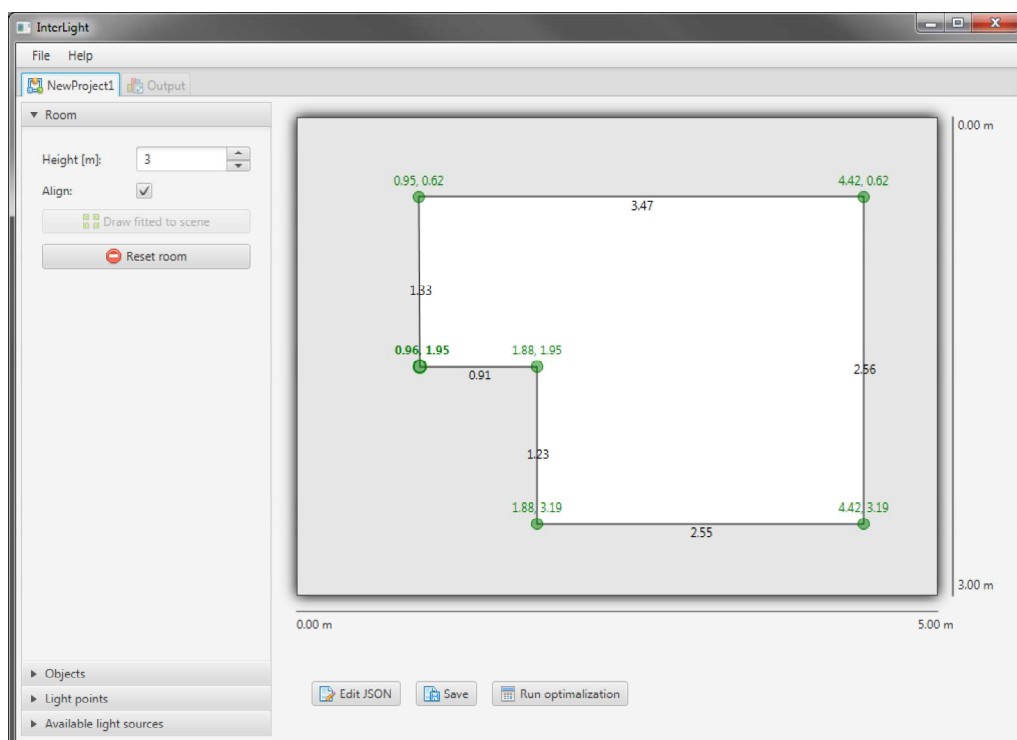
W dowolnym momencie można zapisać cały projekt na dysku w folderze *projects* w formacie JSON, przez wybranie opcji z paska menu *File* → *Save* lub przycisk *Save* w dolnym panelu.

Dzięki API modułu modelu danych, przy użyciu przycisku *Edit JSON*, można bezpośrednio edytować scenę w formacie JSON w nowym oknie (rysunek 4.11). Umożliwia ono zarówno manipulację w polu tekstowym, jak i walidację w sensie syntaktycznym (składniowym). Po prawej stronie wyświetlany jest też wzór poprawnego i kompletnego opisu.

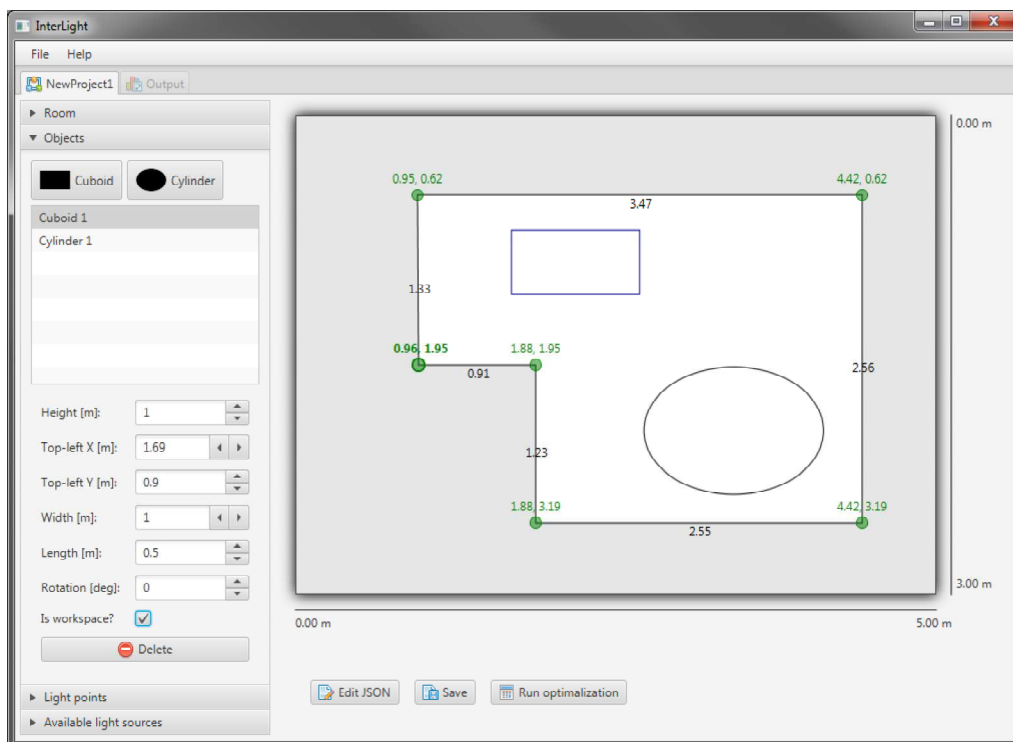
Po wprowadzeniu wszystkich danych wejściowych, uruchamia się moduł optymalizacji za pomocą przycisku *Run optimization*. Jako wynik, zwracana jest lista potencjalnych rozwiązań spełniających normy, wraz ze wskaźnikami ekonomicznymi i możliwością sortowania po nich (rysunek 4.12). Zaznaczenie propozycji wyświetla panel ze szczegółami rozwiązania, czyli jakie źródło światła umieścić w którym punkcie oraz dwie możliwości wizualizacji: za pomocą Radiance (rysunek 4.13) lub prostego oznaczenia słabo oświetlonych obszarów (rysunek 4.14).



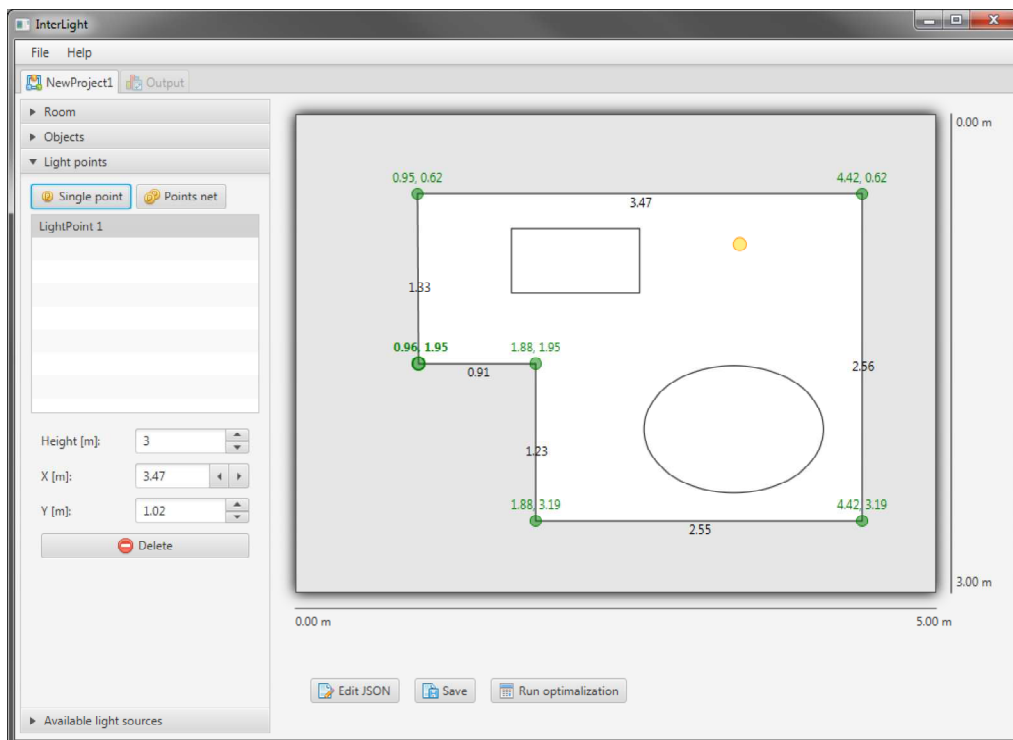
Rysunek 4.5: Tworzenie nowego projektu



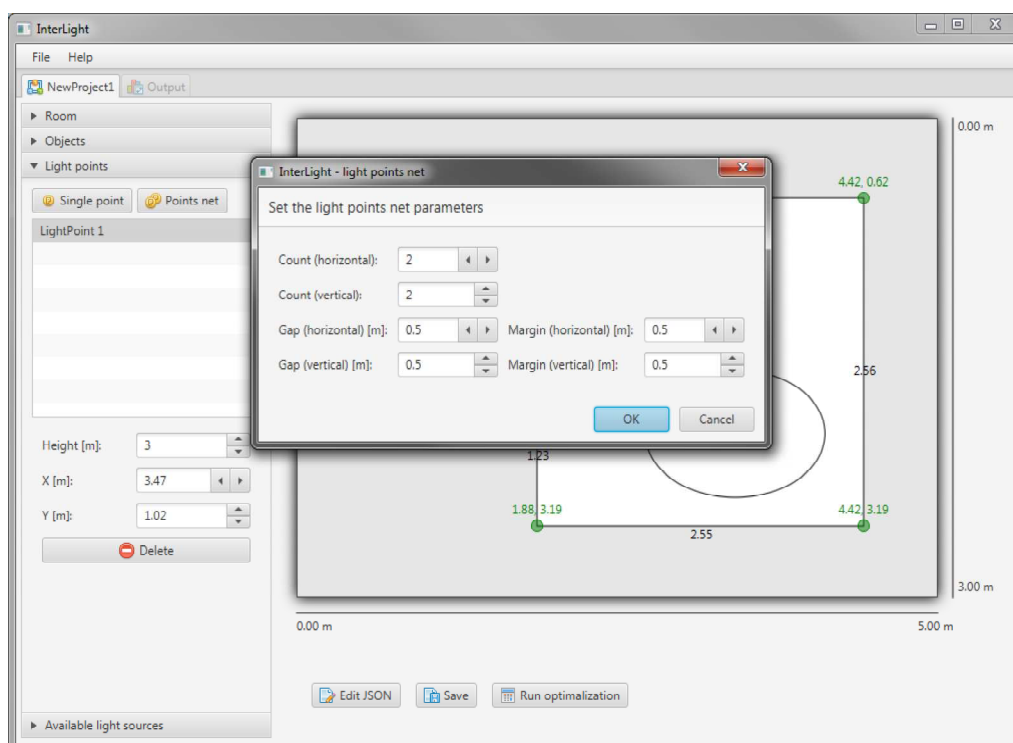
Rysunek 4.6: Rysowanie pokoju



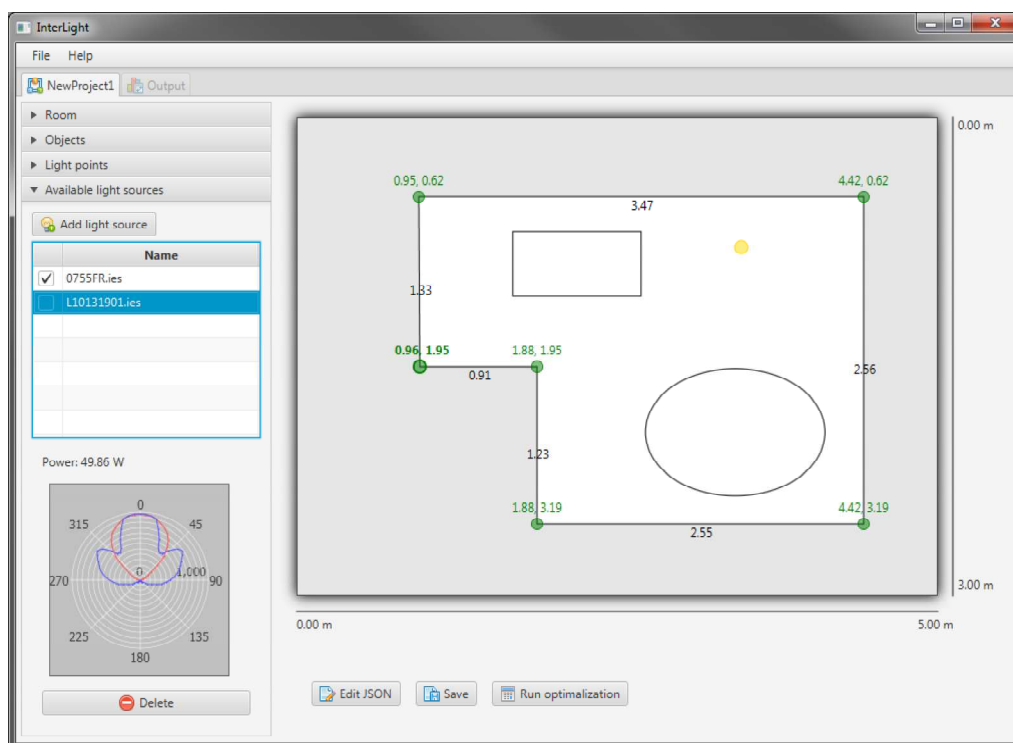
Rysunek 4.7: Dodawanie obiektów



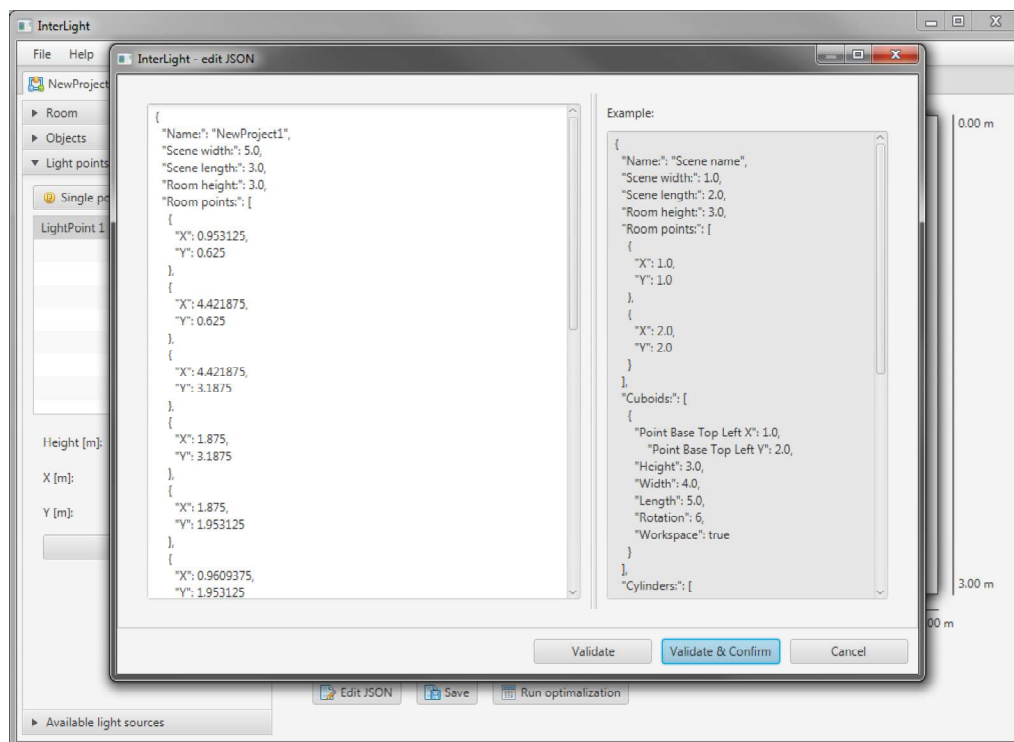
Rysunek 4.8: Dodawanie punktów świetlnych



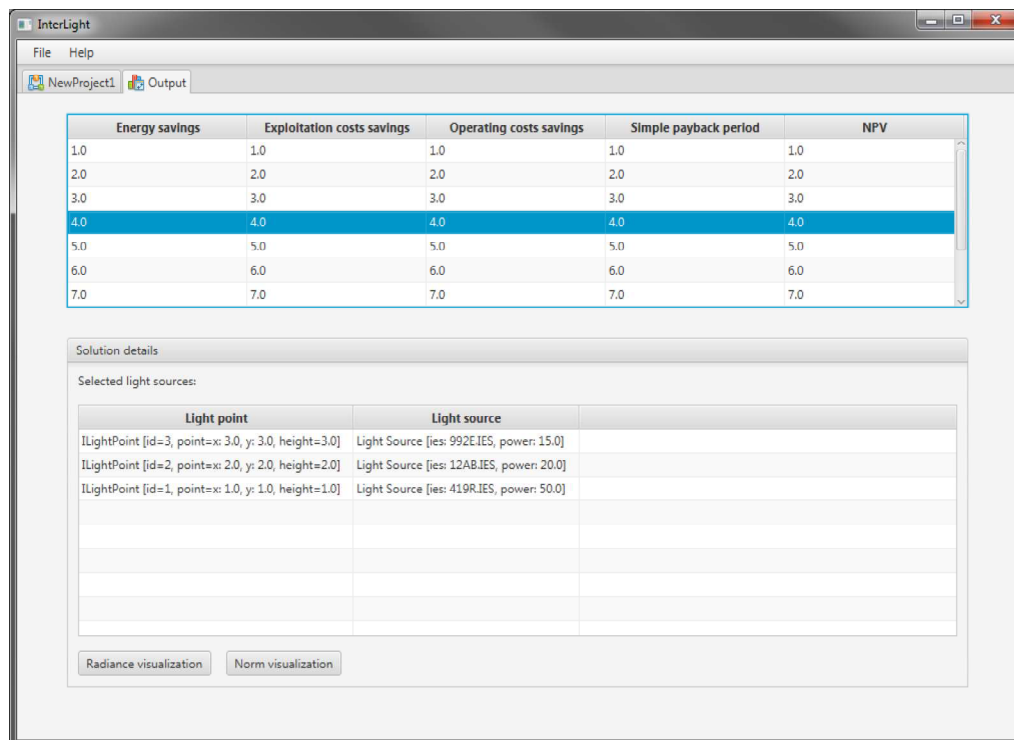
Rysunek 4.9: Dodawanie siatki punktów świetlnych



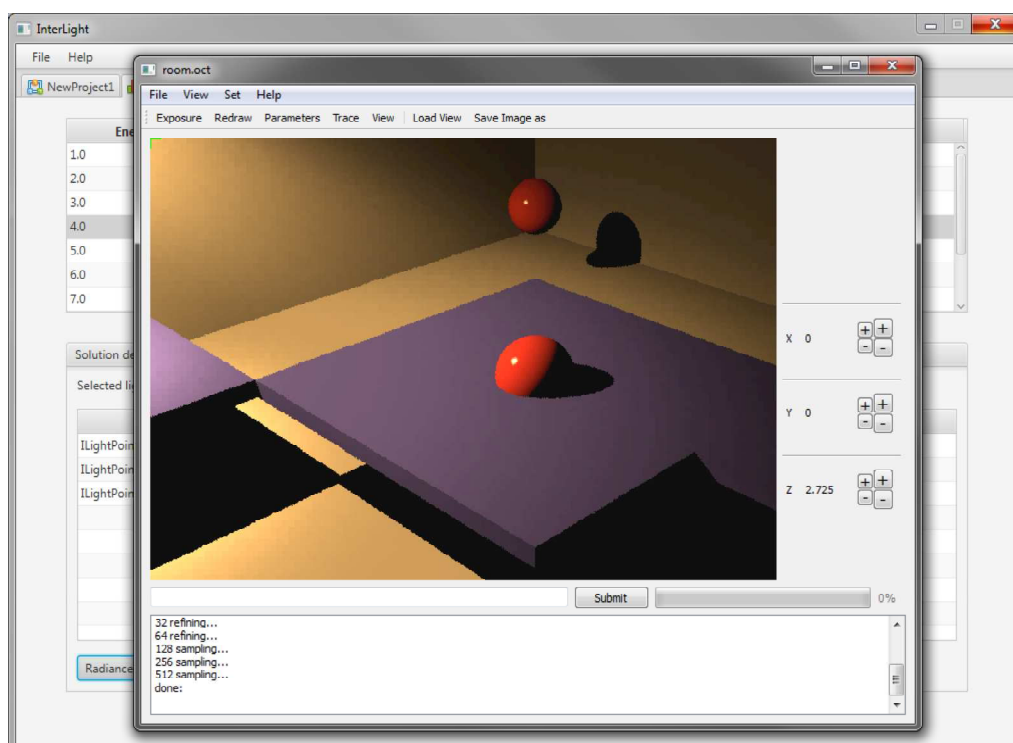
Rysunek 4.10: Wybór źródeł światła



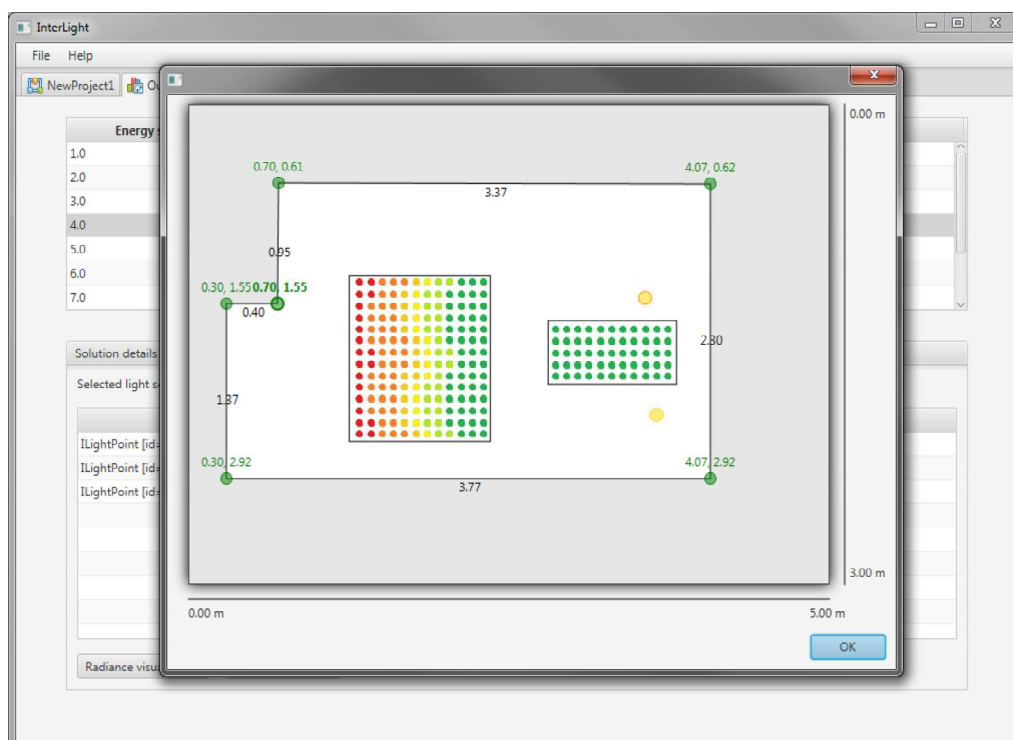
Rysunek 4.11: Edycja sceny JSON



Rysunek 4.12: Wynik optymalizacji



Rysunek 4.13: Wizualizacja Radiance



Rysunek 4.14: Wizualizacja spełniania norm