GUI for Modelling Cyber Physical System

Qualitätssicherungsdokument

Gruppe 14:	Kevin Julian Trometer <kevin_julian.trometer@stud.tu-darmstadt.de> Dominik Fabio Rieder <dominik_fabio.rieder@stud.tu-darmstadt.de> Teh-Hai Julian Zheng <teh-hai_julian.zheng@stud.tu-darmstadt.de> Edgardo Ernesto Palza Paredes <edgardo_ernesto.palza_paredes@stud.tu-darmstadt.de> Jessey Steven Widhalm <jessey_steven.widhalm@stud.tu-darmstadt.de>
Teamleiter:	Emine Saracoglu <doa_s@live.de>
Auftraggeber:	Andreas Tundis <andrea.tundis@dimes.unical.it > Carlos Garcia <garcia@tk.tu-darmstadt.de> Fachbereich 20 Informatik Telecooperation Lab (TK)
Abgabedatum:	30.09.2016

TECHNISCHE UNIVERSITÄT DARMSTADT

Bachelor-Praktikum SoSe 2016
Fachbereich Informatik

Inhaltsverzeichnis

1 Projektbeschreibung

Andreas Tundis und Carlos Garcia von der Telecooperation Group (TK) untersuchen das Verhalten von Cyber Physical Systems. Dafür benötigen sie eine Software zur grafischen Darstellung und Simulation von intelligenten Stromnetzen, sogenannte *Smart Grids*. Es soll mit einem, von uns erstellten, GUI möglich sein, ein solches *Smart Grid* zu modellieren. In diesem *Smart Grid* sollen verschiedene Stromerzeuger und -verbraucher, wie zum Beispiel Kernkraftwerke und Häuser, durch Stromleitungen verbunden werden. Anhand des Modells wird der Stromfluss simuliert, um so die Eigenschaft des Stromnetzes zu überprüfen.
Hierbei sind folgende Punkte zu beachten:

- sind alle Stromverbraucher genügend versorgt
- liegen zu hohe Spannungen auf den Leitungen
- wurden alle Verbindungen korrekt modelliert

Das Ziel von Cyber Physical Systems liegt in der Vernetzung von Software und physikalischen Komponenten in größeren Infrastrukturen, wobei es nicht nur auf Stromleitungen begrenzt sein muss, sondern potenziell auch für die Simulierung von Wasser- und Gasleitungen genutzt werden könnte. Somit ist es für die Forschung sehr interessant und zugleich auch zu einem späteren Zeitpunkt in der Industrie einsetzbar.

Unsere Aufgabe ist vorrangig die Erstellung einer GUI, welches die oben beschriebene Kernfunktionalität beinhalten soll. Dabei werden für die Modellierung die Stromerzeuger und -verbraucher per *Drag & Drop* platziert. Für das Projekt sind diese Standard-Objekte Kernkraftwerke, Häuser, Transformatoren, Schalter und Leitungen. Weitere Objekte können vom Benutzer nach Bedarf selbst hinzugefügt werden. Des Weiteren sollen Häuser mit Konsumentengeräten gefüllt werden können, die den Stromverbrauch erhöhen. Dazu zählen zum Beispiel Fernseher und Lampen. Neue Konsumentengeräte werden ebenfalls vom Benutzer selbst erstellbar sein. Die Simulation zeigt anschließend die gewünschten Daten, und deren Änderungen, im Stromnetzwerk zu verschiedenen Zeiten.

Unsere Auftraggeber haben viele Ideen, wie das Programm erweitert werden kann, wenn die Kernfunktionalitäten realisiert wurden. Die Erste ist die Mitentwicklung und Implementation von künstlich intelligenten Algorithmen, um Optimierungsprobleme zu lösen. Um sicherzustellen, dass wir unser Programm nicht auf eine Weise konstruieren, die spätere Erweiterungen unmöglich machen, treffen wir uns regelmäßig alle zwei Wochen mit unseren Auftraggebern, um die Aufgaben für die nächste Iteration zu besprechen, aber auch um über zukünftig gewünschte Funktionalitäten zu diskutieren und diese zu priorisieren.

2 Qualitätsziele

Im Rahmen des Projekts „GUI for Modelling Cyber Physical System“ führen wir regelmäßig Maßnahmen für unsere Qualitätssicherungsziele durch, die wir Festgelegt haben, um die Qualität für unseren Projekt zu gewährleisten. Dazu wird das ganze Projekt in der Objekt-Orientierten-Programmiersprache Java in der IDE Eclipse geschrieben, mit Gradle die Build- und Testsautomatisierung umgesetzt und Git zur Versionenkontrolle verwendet.

Im folgenden Abschnitt werden, die Qualitätssicherungsziele beschrieben, die wir zusammen mit unserem Auftraggeber definiert haben und auf welcher Art und Weise wir sicherstellen, wie diese erreicht werden.

2.1 Erweiterbarkeit

Im Rahmen des Projektes werden wir auf die Erweiterbarkeit besonders achten, da das Projekt noch von nachfolgenden Gruppen bearbeitet wird, soll es für andere Entwickler einfach sein weitere Ideen mit einzubauen. Von daher ist es wichtig, dass auch neue Erweiterungen möglich sind. Da alles von Grund auf neu geschrieben wird und es noch wenig Komponente existieren, ist es besonders wichtig, dass diese Komponenten vor allem einfach und schnell zu erweitern sind. Im späteren Verlauf, soll das Projekt nicht nur Stromnetze unterstützen, sondern auch andere Physiklische Ströme wie z.B. Wasser oder Gas. Um die Erweiterbarkeit zu gewährleisten, ist es wichtig ein passendes Design sowie Implementierung zu wählen, sodass zukünftige Anpassungen der Funktionen ohne großen Aufwand erreicht werden kann. Dabei ist beim Design zu beachten, dass ein standardisiertes Entwurfsmuster (Pattern) gewählt wird, um eine hohe Abstraktionsebene zu erreichen.
Um dieses Qualitätsziel zu erreichen, haben wir zwei Maßnahmen umgesetzt. Eine Maßnahme ist die Klassen mit hilfe des Model View Controller Pattern so zu strukturieren, dass alle Klassen möglichst Kompakt gehalten werden können, keine Gottklasse entsteht und der Code insgesamt übersichtlicher wird. Somit hat jede Klasse nur die Methoden, die in ihren Aufgabenbereich fällt. Die zweite Maßnahme besteht aus dem kommentieren des Codes und dem generisch Halten der Struktur unserer Methoden. Um die vollständigen Dokumentierung des Quellcodes sicherzustellen, kommt das Eclipse-Plugin Checkstyle zum einsatz. Sollten JavaDoc Kommentare fehlen, so weist uns Checkstyle darauf hin. Bei den Methoden des Controllers halten wir uns voranging an das Bottum-up Verfahren, da so ähnliche Probleme mit gleichen Methoden gelöst werden.
Dominik Rieder ist dafür Zuständig die Erfüllung diese Qualitätsziels zu erfüllen. Dazu überprüft er am Ende jeder Iteration, mit hilfe von Checkstyle, den Code und setzt sich bei fehlenden Kommentaren mit dem zuständigen Entwickler zusammen. Zudem überblicked er alle, in der Iteration, neu dazu gekommenen Klassen auf Einhaltung unserer Maßnahmen und schaut bei größeren Klassen nach, ob diese in kleinere Subklassen geteilt werden könen.

2.2 Speicheroptimierung

Für das Projekt ist Speicheroptimierung ein wesentlicher Aspekt, da die Anzahl von Objekten, wie Gebäude, Konsumentengeräte und Verbindungen, in einem Modell nicht beschränkt ist.
Zur Optimierung werden hauptsächlich drei Eigenschaften des Codes von Bedeutung sein. Die genutzten Datenstrukturen müssen sorgfältig ausgewählt werden, es sollen nicht mehr Kopien von Objekten erstellt werden als nötig, und die Art wie die Modelle gespeichert werden muss möglichst komprimiert und dennoch eindeutig sein.
Es folgen mehrere Gründe die eine Speicheroptimierung nötig machen.
Zu einem können die Modelle so schneller gespeichert und geladen werden, und werden weniger Speicher verbrauchen. Des Weiteren senkt ein optimierter Speicher die Gefahr, dass die GUI bei größeren Modellen weniger flüssige arbeitet. Vor allem bei der Simulation, wo größere Mengen an numerischen Daten in geringer Zeit verarbeitet und Visualisiert werden, ist die Notwendigkeit für Optimierung ersichtlich. Selbst bei kleineren Modellen können, besonders durch die Simulation eines Stromflusses, eine große Menge an Daten erzeugt werden. Für eine einfache Analyse des Modells muss dem Nutzer gewährleistet sein, dass diese Daten möglichst schnell und akkurat verarbeitet und repräsentiert werden.
Wir erreichen dieses Qualitätsziele mittels zwei Grundmaßnahmen. Erstens achten wir genau auf die Erstellung der Objekten. Wir erschaffen jedes neue Objekt nur ein einziges Mal. Danach werden nur Kopien von diesem Objekt produziert, das heißt, wir erzeugen nur die Werte, die relevant sind, wie zum Beispiel die ID, der Name und die Koordinaten dieses Objektes.
Zweitens arbeiten wir mit Bildern. Der User hat die Möglichkeit für jedes neu erzeugte Objekt ein Bild hochzuladen. Damit wir den Speicher optimal verwenden können und ihn nicht überlasten, nutzen wir daher die Referenzen der Bilder. Das bedeutet, wir speichern das Bild in einem Ordner ein einziges mal und holen mittels einem Pointer das Referenz, so dass eine Kopie des Objektes erstellt wird.
Edgardo Palza und Jessey Widhalm sind dafür verantwortlich, dass die Speicheroptimierung korrekt umgesetzt wird. Erachten sie die Speicheroptimierungen als nicht zufriedenstellend, diskutieren sie über eine bessere Umsetzung und geben diese an den entsprechenden Entwickler weiter.

2.3 Stabilität

Ein weiteres Qualitätsziel ist die Stabilität. Dies hat folgenden Hintergrund: Zum einen geht es in unserem Projekt um das Modellieren eines intelligenten Stromnetzes und zum anderen können wir nach dem Modellieren den Stromfluss simulieren.
In normalen Fall, sollte es während einer Modellierung, nicht zu einem Programmabsturz kommen. Jedoch falls es zu dieser Situation kommen sollte, muss gewährleistet sein, dass man die Modellierung wiederherstellen kann. Mittels eines Backups welcher über einen Autosave geschieht, kann man ohne Komplikationen die vorherige Modellierung wiederherstellen.
Anders als bei der Modellierung, geht es in der Simulation um die Berechnung des Stromflusses. Hierbei bedeutet Stabilität, dass falls es während der Simulation auf Grund der Berechnung zu Fehlern kommt, die Simulation weder abbricht noch dass der Prozess unaufgefordert beendet wird. Anhand von Stresssimulationen wird anschließend geprüft, wie sich unser Programm bei Last verhält.
In beiden Situationen ist es wichtig, dass das Risiko für das Auftreten eines unerwarteten Feh-

lers, so gering wie möglich ist.

Um eine Stabilität wie sie beschrieben wurde zu gewährleisten, werden folgende Massnahmen getroffen:

1. Autosave:
   Bei jeder Änderung, welche die Datenstruktur beeinflusst (also nicht die grafischen Änderungen wie z.B geänderte Position) bzw. das Verhalten der Simulation beeinflusst, wird der Zustand mit allen Änderungen gespeichert. Hierzu wird bei jeder Änderung eine Temporäre Datei erstellt, die vom letzten bekannten Zustand Lokal gespeichert wird. Nach dem unaufgeforderten Beenden des Programms kann der Zustand wiederhergestellt werden.

2. Automatisierte Tests:
   Auch an dieser Stelle betrachten wir den Einsatz von Tests, über das JUnit Test-Framework, als sinnvoll. Bei jeder Ausführung soll eine Reihe von automatisierten Tests über Gradle durchlaufen werden, damit jeder Entwickler weiß, dass seine Änderungen keine „Unstabilitäten“ verursacht haben. Die Tests sind dementsprechend darauf ausgelegt, alle Basismethoden unter kritischen Parametern zu testen. Auch der richtige Umgang mit Exceptions soll an dieser Stelle behandelt werden.

3. Stresssimulation:
   Um diese Maßnahme zu erreichen, führen wir Stresssimulationen ein, die dazu führen unsere Simulation an Ihre Grenzen zu bringen. Hierbei lassen wir die Applikation, im Simulationsmodus mindestens für zwei Stunden laufen und überprüfen im nachhinein ob die Simulation problemlos und korrekt abgelaufen ist.

Für die Stabilität setzen sich Julian Zheng und Kevin Trometer jede Iteration zusammen, und prüfen ob die Software auf dem aktuellen Stand den Stabilitätsanforderungen gerecht wird. Sollten Fehler bei den automatisierten Test oder Probleme bei den Stresssimulationen auftreten, setzen sie sich zusammen und optimieren die betroffenen Stellen.

A Anhang

A.1 Glossar

Bitte beachten sie folgende Begriffe, die wir in unserem Dokument benutzen werden:

- Canvas: Mit “Canvas” bezeichnen wir die Hauptarbeitsfläche. Hier werden alle “CpsObject” instanziiert, sowie alle Interaktionen ausgeführt.
- CpsObject: Mit dem Begriff “CpsObject” bezeichnen wir alle möglichen Objekte (“HolonObject”, “HolonSwitch” und “CpsEdge”), die in unserer Canvas erzeugt bzw. in unserem System teilnehmen können werden. Alle “CpsObject” bestehen aus Name (benutzerdefiniert), (eindeutige) ID, Typ (entweder “HolonObject”, “HolonSwitch” oder “CpsEdge”) und Nachbarobjekten.
- HolonObject: “HolonObject” erbt direkt von unsere “CpsObject”-Klasse. Weitere Eigenschaften sind: Allgemeine Produktion/Verbrauch, Zustand (ob das Objekt nicht, teilweise oder total mit Energie versorgt wird) und eine Liste von “HolonElement”. Beispielsweise: Haus oder Kraftwerk
- HolonSwitch: Wir bezeichnen alle Schalter in unseren System als “HolonSwitch”. “HolonSwitch” erbt auch von die Klasse “CpsObject”. Weitere Eigenschaften sind: Modus (entweder manuell oder automatisch), Zustand (an oder aus - nur manueller Modus) und ein Array von Werten (zu jedem Zeitschritt wird durch den Graph ein Zustand - an oder aus - beschrieben).
- CpsEdge: Mit dem Begriff “CpsEdge” bezeichnen wir die Verbindungen zwischen Objekten in unserem System. Jede “CpsEdge” enthält eine maximale Kapazität, aktueller Stromfluss, Zustand (aktiv oder kaputt), Quelle und Ziel.
- HolonElement: “HolonElement” sind Objekte, die man zu einem “HolonObject” hinzufügen kann. Ein “HolonElement” besteht aus Name (benutzerdefiniert), (eindeutige) ID, Menge (Anzahl von Kopien dieser “HolonElement”), Energie (positive Energie entspricht Produktion von Strom und negative Energie entspricht Verbrauch von Strom), Zustand (ob das “HolonElement” global aktiv oder inaktive ist) und ein Array von Werten (zur jeden Zeitschritt wird durch den Graph die genaue Energie beschrieben). Beispielsweise: Fernsehen, Radio oder Solarmodul
- Categories: Kategorien werden alle “CpsObject” enthalten bzw. in Gruppen unterteilen. (“Categories”-Liste sind in der linken Spalte zu finden). Beispielsweise: Producer, Consumer oder Component.
- PropertyTable: “PropertyTable” ist eine Tabelle, die wichtige Information über das (im “Canvas” ausgewählte) “CpsObject” zeigt. (“PropertyTable” ist in der rechten Spalte unten zu finden). Gezeigte Informationen des Objektes:

Für “HolonObject”: Name, ID, gesamte Energy und Verbindungen.
Für “HolonSwitch”: Name, ID, Modus, Zustand und Verbindungen.
Für “CpsEdges”: Name, ID, aktuelle Stromfluss, maximale Kapazität und Zustand.

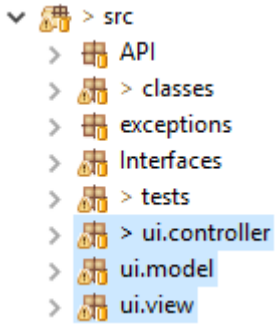
- HolonElementTable: Mit dem Begriff “HolonElementTable” wird die Tabelle, welche die Informationen über alle “HolonElement” der (im “Canvas” ausgewählte) “CpsObject” bezeichnet. (“HolonElementTable” ist in der rechten Spalte in der Mitte zu finden). Gezeigte Information: Name, ID, Energie, Menge und Zustand.
- Graph: Der Graph wird benutzt, um das Verhalten von “HolonElement” bzw. “HolonSwitch” pro Zeitschritt zu beschreiben. (Der Graph ist in der rechten Spalte oben zu finden).
- Modellierung und Simulation: Die bezeichnen beide Modi, welche unser Software anbietet. Beide betrachten verschiedenes Verhalten der Objekten im System. Beispielsweise: Strom-fluss und Zeitschrittabhängigkeit. Für mehr Information siehe Abschnitt A.6 Short-Story.

A.2 Erweiterbarkeit-Maßnahmen

A.2.1 MVC Pattern

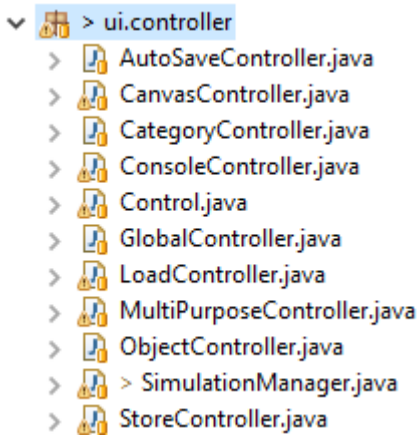
Aufgrund der strikten Trennung von Logik, Daten und Visualisierung ist (leichte) Erweiterbarkeit dadurch gewährleistet, dass man gewünschte Änderungen nur in den jeweiligen Klassen für Controller, View und Model vornehmen muss. Die anderen Aspekte können dann beibehalten werden. Möchte man also eine komplett andere Art von Visualisierung umsetzen oder die vorhandene modifizieren, ist nur das erstellen einer neuen View-Klasse bzw. das Ergänzen der bereits vorhandenen View-Klassen notwendig. Die Logik und das Model kann dann einfach übernommen werden. Analog kann man für die anderen Aspekte vorgehen.

Um das MVC Design umzusetzen, wurde zunächst für jede Komponente ein eigenes Package erstellt.



Controller:

Im Controller Package befinden sich alle Arten von Controllern. Um eine Gottklasse zu vermeiden haben wir die Funktionalitäten der Controller auf verschiedene Bereiche aufgeteilt. So ist z.B der CategoryController.java nur für Manipulation der Kategorie Daten zuständig. Die Klasse Control.java beinhaltet alle anderen Controller. Sie wertet die Anweisungen aus und entscheidet welcher spezielle Controller sie anschließend ausführen wird.



So sehen alle Attribute der Klasse Control aus. Sowohl das Model als auch alle anderen Controller sind enthalten.

```
public class Control {

	private Model model;

	private final ConsoleController consoleController;
	private final MultiPurposeController multiPurposeController;
	private final CategoryController categoryController;
	private final ObjectController objectController;
	private final CanvasController canvasController;
	private final GlobalController globalController;
	private final StoreController storeController;
	private final LoadController loadController;
	private final AutoSaveController autoSaveController;
	private SimulationManager simulationManager;
	private String autoPath = "";
```

Beispiel Löschen einer Kategorie:

deleteCategory löscht eine Kategorie mit gegebenem Namen. Da es sich dabei um Manipulation der Kategorie Daten handelt, ruft Control die Methode deleteCategory der Klasse CategoryController auf.

```
	/**
	 * delete a given Category.
	 *
	 * @param cat
	 *            the Category
	 */
	public void deleteCategory(String cat) {
		categoryController.deleteCategory(cat);
	}
```

deleteCategory in der Klasse CategoryController bedient sich des MultiPurposeControllers, hier mpC, um die zu löschende Kategorie aufgrund ihres Namens zu finden und sie anschließend der removeCategory Methode zu übergeben.

```
	/**
	 * delete a given Category.
	 *
	 * @param category
	 *            the Category
	 */
	public void deleteCategory(String category) {
		removeCategory(mpC.searchCat(category));
	}
```

removeCategory löscht nun endgültig eine gegebene Kategorie aus dem Model und sorgt dafür dass alle Indizes innerhalb der Kategorien konsistent bleiben. Anschließend wird die Methode notifyCatListeners ausgeführt.

```
/**
 * remove a Category from Model.
 *
 * @param c
 *            Category
 */
public void removeCategory(Category c) {
        mpC.decIdx(c.getName(), model.getCgIdx());
        model.getCgIdx().remove(c.getName());
        model.getCategories().remove(c);

        notifyCatListeners();
}
```

notifyCatListeners ruft nun bei allen im Model angemeldeten Listenern die onChange Methode auf, die durch das Interface CategoryListener gegeben ist. In diesem Fall ist nur die GUI angemeldet, welche nun das „Update“ auch visualisiert. Somit ist der typische Datenstrom des MVC-Designs erfüllt.

```
/**
 * notifies all listeners about changes in the Categories.
 */
public void notifyCatListeners() {
        for (CategoryListener l : model.getCategoryListeners()) {
                l.onChange(model.getCategories());
        }
}
```

Beispiel Löschen eines ausgewählten Objects auf der Modelierungsfläche:

deleteSelectedObject löscht ein gegebenes Object. Diesmal handelt es sich um Manipulation der Object Daten, weswegen auch deleteSelectedObject der Klasse ObjectController aufgerufen wird.

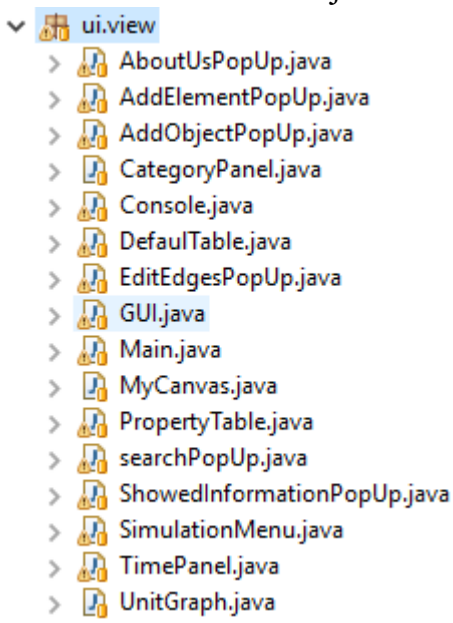
```
/**
 * deletes a selectedObject.
 *
 * @param obj
 *            Cpsobject
 */
public void deleteSelectedObject(AbstractCpsObject obj) {
        model.getSelectedCpsObjects().remove(obj);
}
```

Der Datenstrom verläuft nun Analog zu deleteCategory mit dem einzigen Unterschied, dass am Ende nicht die Methode notifyCatListeners, sondern repaint der Klasse MyCanvas aufgerufen wird. Diese ist sozusagen das Äquivalent zur Update Methode, da sie alle Daten auf der Modelierungsfläche neu zeichnet und somit auch die Änderungen übernommen werden.

View:

Im View Package befinden sich alle Klassen, die das Model in irgendeiner Art visualisieren. Auch hier wurden alle visuellen Komponenten von einander strikt getrennt. Somit hat jedes Pop-Up-Fenster und jeder Graph seine eigene Klasse. Die Modellierungsfläche ist ebenfalls in

die eigene Klasse MyCanvas.java ausgelagert worden. Die GUI.java Klasse beinhaltet wiederum alle Klassen in diesem Package als Attribut und entscheidet wann und wie die einzelnen Klassen angezeigt werden. Desweiteren leitet die GUI Klasse jegliche Inputs seitens des Users weiter an Control.java. Somit ist schonmal ein Aspekt des MVC Verhaltens gewährleistet.



Beispiel Hinzufügen eines Objects in eine Kateogrie:

Zu sehen ist hier ein Abschnitt der Methode die ausgeführt wird, wenn der Add-Button ausgeführt wird. Mit selectedNode.getLevel ( ) == 1 wird zunächst überprüft ob es sich bei dem ausgewählten Ordner Knoten um eine Kategorie (welche immer die Tiefe 1 haben) handelt. Anschließend wird das PopUp Fenster addObjectPopUP erstellt und die richtigen Parameter werden gesetzt. Auch der Controller wird als Parameter übergeben.

```
btnAdd.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent arg0) {
                .
                .
                .
                if (selectedNode.getLevel() == 1) {
                        AbstractCpsObject tmp = new HolonObject("");
                        addObjectPopUP = new AddObjectPopUp(false, tmp, null);
                        addObjectPopUP.setVisible(true);
                        addObjectPopUP.setController(controller);
                        addObjectPopUP.setCategory(selectedNode.toString());
                }
        }
}
```

Hier ist nun ein Abschnitt der Methode zu sehen die ausgeführt wird, wenn der OK-Button geklickt wird. Die addObject Methode des Controllers wird mit den benötigten Parametern ausgeführt. GivenCategory wurde bereits beim erstellen des PopUp-Fensters gesetzt. Die restlichen Parameter sind von den Eingaben des Users abhängig. Ab dieser Stelle verläuft die Verarbeitung der Daten analog zu den Beispielen aus Control. Ebenfalls Analog werden mit Inputs in anderen PopUp-Fenstern umgegangen.

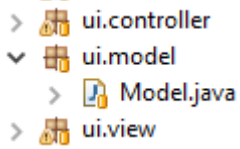
```
okButton.addMouseListener(new MouseAdapter() {
        public void mouseClicked(MouseEvent e) {
```

```
                .
                .
                .

        controller.addObject(controller.searchCategory(givenCategory), objectName.getText(), hElements, imagePath);
        }
}
```

Model:

Im Model Package befinden sich alle Klassen die nur Daten, mit jeweiligen Gettern und Settern enthalten. In unserem Fall nur die Klasse Model.java.



Hier sind nun alle Attribute der Model Klasse zu sehen. Für alle Attribute gibt es noch entsprechende Getter und Setter Methoden.

```
public class Model {

        // Global Variables
        private static int sCALE = 50; // Picture Scale
        private static int sCALEdIV2 = sCALE / 2;
        private static final int ITERATIONS = 100;
        private int curIteration = 0;
        // ID of the Selected Object
        private AbstractCpsObject selectedCpsObject = null;
        private HolonElement selectedHolonElement;
        private CpsEdge selectedEdge;
        private ArrayList<AbstractCpsObject> selectedObjects = new ArrayList<AbstractCpsObject>();
        private ArrayList<AbstractCpsObject> clipboardObjects = new ArrayList<AbstractCpsObject>();
        private Console console;

        // Iteration Speed
        private int timerSpeed = 1000;

        // Simulation boolean
        private boolean isSimulation = false;

        private int selectedID = 0;
        // number of the current autosave
        private int autoSaveNr = -1;
        // number of max simultaneous autosaves
        private int numberOfSaves = 35;
        /*
         * Array of all categories in the model. It is set by default with the
         * categories ENERGY, BUILDINGS and COMPONENTS
         */
        private ArrayList<Category> categories;

        /*
         * Array of all CpsObjects in our canvas. It is set by default as an empty
         * list.
         */
        private ArrayList<AbstractCpsObject> objectsOnCanvas;

        private HashMap<String, Integer> cgIdx;
        private HashMap<Integer, Integer> cvsObjIdx;

        /*
         * Array of all CpsObjects in our canvas. It is set by default as an empty
         * list.
         */
        private ArrayList<CpsEdge> edgesOnCanvas;

        /*
         * Array for all Listeners
         */
```

```
    private List<CategoryListener> categoryListeners;
    private List<ObjectListener> objectListeners;
```

A.2.2 Checkstyle

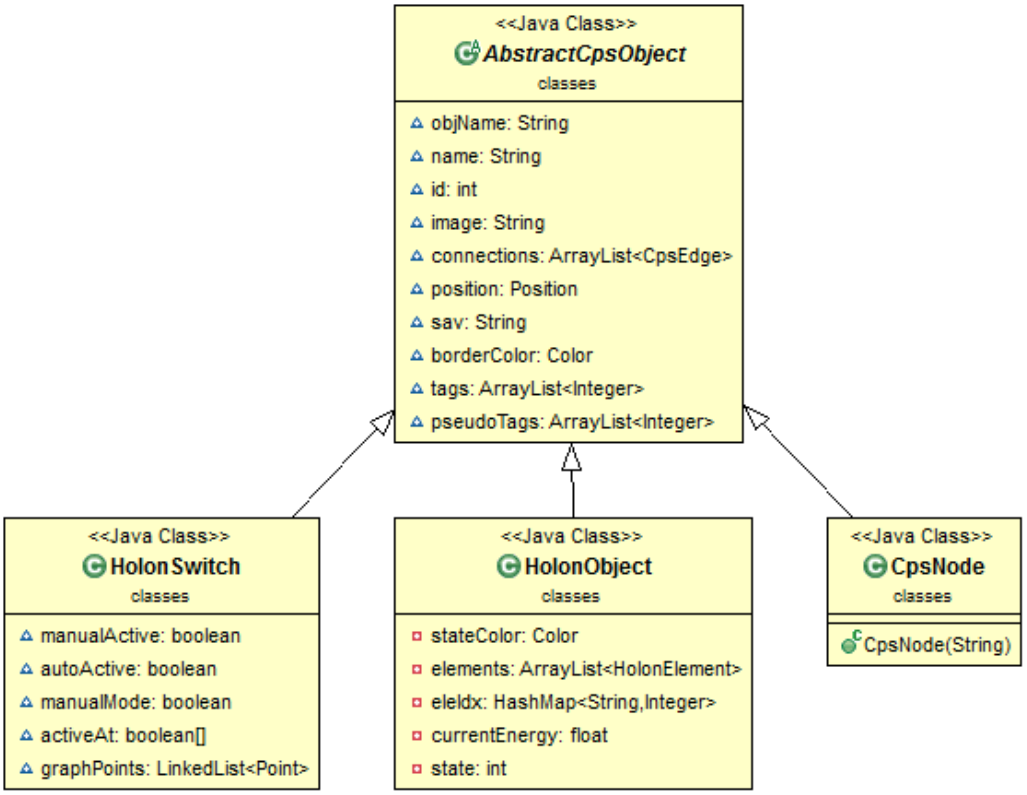
Für eine saubere Dokumentation wurde während des Projektes der gesamte Code mit Checkstyle überwacht. Dadurch konnten wir sichergehen, dass wir bei allen Klassen und Methoden die nötigen Javadoc Kommentare hinzugefügt haben, damit man sich für eine mögliche Erweiterung schnell einlesen kann. Zusätzlich haben wir darauf geachtet, dass die Namenskonventionen von Klassen, Methoden und Variablen eingehalten werden und unnötiger Code und Imports entfernt werden. Damit konnten wir dann am Ende eine vollständige und korrekte Dokumentation erstellen.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE module PUBLIC "-//Puppy Crawl//DTD Check Configuration 1.3//EN"
"http://www.puppycrawl.com/dtds/configuration_1_3.dtd">
<!--
    This configuration file was written by the eclipse-cs plugin configuration
editor
-->
<!--
    Checkstyle-Configuration: BP Configuration
    Description:
Configuration for the Bachelor Praktikum
-->
<module name="Checker">
  <property name="severity" value="warning"/>
  <module name="TreeWalker">
    <module name="JavadocMethod">
      <property name="suppressLoadErrors" value="true"/>
    </module>
    <module name="JavadocStyle"/>
    <module name="JavadocType"/>
    <module name="JavadocVariable">
      <property name="severity" value="ignore"/>
      <metadata name="net.sf.eclipsecs.core.lastEnabledSeverity"
value="inherit"/>
    </module>
    <module name="WriteTag"/>
    <module name="NonEmptyAtclauseDescription"/>
    <module name="JavadocTagContinuationIndentation"/>
    <module name="SummaryJavadoc"/>
    <module name="AtclauseOrder"/>
    <module name="JavadocParagraph"/>
    <module name="SingleLineJavadoc"/>
    <module name="AbstractClassName"/>
    <module name="ClassTypeParameterName"/>
    <module name="ConstantName"/>
    <module name="LocalFinalVariableName"/>
    <module name="LocalVariableName"/>
    <module name="MemberName"/>
    <module name="MethodName"/>
    <module name="MethodTypeParameterName"/>
    <module name="InterfaceTypeParameterName"/>
    <module name="PackageName"/>
    <module name="ParameterName"/>
    <module name="StaticVariableName"/>
    <module name="TypeName"/>
    <module name="AbbreviationAsWordInName"/>
    <module name="CatchParameterName"/>
    <module name="AnnotationUseStyle"/>
    <module name="MissingDeprecated"/>
    <module name="MissingOverride"/>
    <module name="PackageAnnotation"/>
    <module name="SuppressWarnings"/>
    <module name="AnnotationLocation"/>
    <module name="AvoidStarImport">
      <property name="severity" value="ignore"/>
      <metadata name="net.sf.eclipsecs.core.lastEnabledSeverity"
value="inherit"/>
    </module>
    <module name="AvoidStaticImport">
      <property name="severity" value="ignore"/>
      <metadata name="net.sf.eclipsecs.core.lastEnabledSeverity"
value="inherit"/>
    </module>
```

```
    <module name="IllegalImport">
      <property name="severity" value="ignore"/>
      <metadata name="net.sf.eclipsecs.core.lastEnabledSeverity"
value="inherit"/>
    </module>
    <module name="ImportOrder">
      <property name="severity" value="ignore"/>
      <metadata name="net.sf.eclipsecs.core.lastEnabledSeverity"
value="inherit"/>
    </module>
    <module name="RedundantImport"/>
    <module name="UnusedImports"/>
    <module name="ImportControl">
      <property name="severity" value="ignore"/>
      <metadata name="net.sf.eclipsecs.core.lastEnabledSeverity"
value="inherit"/>
    </module>
    <module name="CustomImportOrder">
      <property name="severity" value="ignore"/>
      <metadata name="net.sf.eclipsecs.core.lastEnabledSeverity"
value="inherit"/>
    </module>
    <module name="ParameterNumber"/>
    <module name="MethodCount"/>
    <module name="GenericWhitespace"/>
    <module name="ClassDataAbstractionCoupling">
      <property name="severity" value="ignore"/>
      <metadata name="net.sf.eclipsecs.core.lastEnabledSeverity"
value="inherit"/>
    </module>
  </module>
  <module name="JavadocPackage">
    <property name="severity" value="ignore"/>
    <metadata name="net.sf.eclipsecs.core.lastEnabledSeverity" value="inherit"/>
  </module>
  <module name="Header">
    <property name="severity" value="ignore"/>
    <metadata name="net.sf.eclipsecs.core.lastEnabledSeverity" value="inherit"/>
  </module>
  <module name="RegexpHeader">
    <property name="severity" value="ignore"/>
    <metadata name="net.sf.eclipsecs.core.lastEnabledSeverity" value="inherit"/>
  </module>
  <module name="FileLength"/>
</module>
```

A.2.3 Generischer Charakter der Methoden

Da der Hauptbestandteil des Programms die Simulation von Smartgrid Komponenten ist, ist es auch wichtig Erweiterbarkeit hinsichtlich neuer Komponenten mit eventuell anderem Verhalten zu gewährleisten. Um dies zu ermöglichen wurde eine Oberklasse erstellt, die zunächst alle Standardmethoden und Attribute beinhalten.



AbstractCpsObject ist die Oberklasse mit Attributen die jedes Objekt benötigt bzw. benötigen wird. So braucht zum Beispiel jedes Objekt auf der Modellierungsfläche das Attribut Position. Zusätzlich sind noch jeweilige Getter und Setter enthalten. Wie man in der Model Klasse sehen kann (siehe Unterpunkt View bei MVC), ist das Attribut objectsOnCanvas, also alle Objekte die sich auf der Modellierungsfläche befinden, vom Typ ArrayList<AbstractCpsObject>. Das heißt, das Model unterscheidet nicht weiter um welche Objekte es sich genau handelt. Dies wird erst in der Controller Komponente SimulationManager, in der letztendlich das Verhalten erzeugt wird, getan. HolonSwitch, HolonObject und CpsNode erben nun von AbstractCpsObject und ergänzen Attribute und Methoden die ihr Verhalten bestimmen. Diese drei Klassen sind die Default Klassen die wir zur Verfügung stellen.

Ein Beispiel für den generischen Charakter der Methoden wäre das Zuordnen aller Objekte auf der Modellierungsfläche zu Sub-Nets.

```
/**
 * recursivly generates a subnet of all objects, that one specific object is
 * connected to.
 *
 * @param cps
 *            AbstractCpsObject
 * @param visited
```

```
*            visited Array of Integer
* @param sN
 *            Subnets
 * @return Subnet
 */
public SubNet buildSubNet(AbstractCpsObject cps, ArrayList<Integer> visited, SubNet sN) {
        visited.add(cps.getID());
        if (cps instanceof HolonObject) {
                sN.getObjects().add((HolonObject) cps);
        }
        if (cps instanceof HolonSwitch) {
                sN.getSwitches().add((HolonSwitch) cps);
        }
        removeFromToHandle(cps.getID());
        AbstractCpsObject a;
        AbstractCpsObject b;
        for (CpsEdge edge : cps.getConnections()) {
                a = edge.getA();
                b = edge.getB();
                if (!(cps instanceof HolonSwitch) && edge.getState()) {
                        if (!(sN.getEdges().contains(edge))) {
                                sN.getEdges().add(edge);
                        }
                }
                if(cps instanceof HolonSwitch && ((HolonSwitch)cps).getState(timeStep) && edge.getState()){
                        if (!(sN.getEdges().contains(edge))) {
                                sN.getEdges().add(edge);
                        }
                }
                if (!visited.contains(a.getID()) && legitState(cps)) {
                        sN = buildSubNet(a, visited, sN);
                }
                if (!visited.contains(b.getID()) && legitState(cps)) {
                        sN = buildSubNet(b, visited, sN);
                }
        }
        return sN;
}
```

Ohne im Detail auf die Funktionsweise des Codeabschnitts einzugehen sieht man, dass an einigen Stellen der Parameter cps vom Typ AbstractCpsObject mittels des instanceof Befehls auf seinen genauen Typ überprüft und dementsprechend behandelt wird. So ähnlich wird es in allen anderen Methoden gehandhabt die das Verhalten der Objekte in irgendeiner Form beachten müssen. Es liegt also ein generisches Verhalten vor, da jedes Objekt vom Typ AbstractCpsObject zulässig für unsere Methoden sind.

Eine Konkrete Erweiterungen die unsere Auftraggeber genannt hat, wäre zum Beispiel das Hinzufügen eines Transformators. Dies ist nun leicht umsetzbar indem man die Klasse Transformator erstellt und von AbstractCpsObject erben lässt. Nun muss man nur noch in den Methoden, in denen es relevant wäre ob es sich bei einem bestimmten Objekt um einen Transformator handelt entsprechend ergänzen wie die Methode zu reagieren hat.

A.2.4 Bottom-Up Verfahren

Ein weiterer Aspekt der Erweiterbarkeit gewährleisten soll ist das Aufbrechen der Probleme nach dem Buttom Up Verfahren. Objekt spezifische Methoden werden in den Klassen der jeweiligen Objekte implementiert und der Controller liefert letztendlich nur bestimmte Objekte zurück. Ist dann wieder das Verhalten des Objekts relevant wird, wie beim Abschnitt Generischer Charakter der Methoden bereits erklärt, zunächst überprüft von welchem Typ das Objekt ist und anschließend nach dem Typecast die objektspezifische Methode(n) ausgeführt.

Der folgende Codeabschnitt aus der Klasse HolonObject beschreibt die Methode getCurrentEnergyAtTimeStep(x) welche die aktuelle Energie für einen bestimmten Zeitpunkt zurück liefert.

```
/**
 * Getter for the current energy at a given timestep.
 *
 * @param x
 *            timestep
 * @return corresponding energy
 */
public float getCurrentEnergyAtTimeStep(int x) {
        float temp = 0;
        for (HolonElement e : getElements()) {
                if (e.getActive()) {
                        temp = temp + e.getTotalEnergyAtTimeStep(x);
                }
        }
        currentEnergy = temp;
        return currentEnergy;
}
```

Hier ist bereits schon zu sehen, dass beim Oject e die Methode e.getTotalEnergyAtTimeStep(x) aus der Klasse HolonElement aufgerufen wird (die Summe bildet dann die gesamt Energie des HolonObjects). Würde man also etwas daran ändern wollen wie sich die Energie einzelner Elemente berechnet, müsste man nur die Methode getTotalEnergyAtTimeStep(x) der Klasse HolonElement ändern. Die Methode getTotalEnergyAtTimeStep(x) aus der Klasse HolonObject (also der hier gezeigte Codeabschnitt), würde diese Änderung dann übernehmen ohne darüber Bescheid wissen zu müssen wie sie genau implementiert wurde, da sie lediglich die Methode aufruft.

Hier ist nun ein weiterer Codeabschnitt zu sehen, diesmal aus der Klasse SimulationManager welche ein Bestandteil der Control Klasse ist. Wie bei der vorher gezeigten Methode, wird für jedes HolonObject hl die Methode hl.getCurrentEnergyAtTimeStep(x) aufgerufen. Auch hier gilt wieder, dass der SimulationManager nicht darüber Bescheid wissen muss wie die Methode implementiert wurde.

```
/**
 * calculates the energy of either all producers or consumers.
 *
 * @param type
 *            Type
 * @param sN
*            Subnet
 * @param x
 *            Integer
 *
 * @return The Energy
 */
public float calculateEnergy(String type, SubNet sN, int x) {
        float energy = 0;
        for (HolonObject hl : sN.getObjects()) {
                if (type.equals("prod")) {
                        if (hl.getCurrentEnergyAtTimeStep(x) > 0) {
                                energy = energy + hl.getCurrentEnergyAtTimeStep(x);
                                hl.setState(3);
                        }
                }
                if (type.equals("cons")) {
                        if (hl.getCurrentEnergyAtTimeStep(x) < 0) {
                                energy = energy + hl.getCurrentEnergyAtTimeStep(x);
                                hl.setState(1);
                        }
```

```
                }
                if (hl.getCurrentEnergyAtTimeStep(x) == 0) {
                        hl.setState(0);
                }
        }
        return energy;
}\\
                \newline
                Durch das Buttom Up Verfahren ist es also m glich Methoden die an bestimmte Klassen gebunden sind, nur
                     in den jeweiligen Klassen zu erweitern oder zu ver ndern um somit das Verhalten der einzelnen
                     Klassen im gesamten Programm zu  ndern .
```

A.3 Speicheroptimierung-Maßnahmen

A.3.1 Objekte

Für die Speicheroptimierung, folgten wir zwei verschiedene Ansätze.
Zum einen sollten Objekte nicht immer wieder neu angelegt werden, sondern sollten durch einen Copy-Konstruktor kopiert werden. Lediglich fehlende Attribute, die vorher nicht benötigt waren, werden ergänzt.

```
        /**
         * Constructor for a CpsObject with an unique ID.
         *
         * @param objName
         *            of the Object
         */
        public AbstractCpsObject(String objName) {
                setObjName(objName);
                setName(objName);
                setImage("/Images/Dummy_House.png");
                tags = new ArrayList<Integer>();
                pseudoTags = new ArrayList<Integer>();
        }

        /**
         * Constructor for a new CpsObject with an unique ID (This constructor
         * correspond to the interaction between the Categories and Canvas)-->
         * actually the "new" Object is a copy.
         *
         * @param obj
         *            Object to be copied
         */
        public AbstractCpsObject(AbstractCpsObject obj) {
                setObjName(obj.getObjName());
                setName(obj.getObjName());
                setConnections(new ArrayList<CpsEdge>());
                setPosition(new Position());
                setID(IdCounter.nextId());
                setImage(obj.getImage());
        }
```

```
        /**
         * Create a new HolonElement with a user-defined name, amount of the same
         * element and energy per element.
         *
         * @param eleName
         *            String
         * @param amount
         *            int
         * @param energy
         *            float
         */
        public HolonElement(String eleName, int amount, float energy) {
                setEleName(eleName);
                setAmount(amount);
```

```
                setEnergy(energy);
                setActive(true);
                setSign(energy);
                setEnergyAt(energy);
                setId(IdCounterElem.nextId());
        }

        /**
         * Create a copy of the HolonElement given each one a new ID.
         *
         * @param element
         *            element to copy
         */
        public HolonElement(HolonElement element) {
                setEleName(element.getEleName());
                setAmount(element.getAmount());
                setEnergy(element.getEnergy());
                setActive(element.getActive());
                setSign(element.getEnergy());
                setEnergyAt(element.getEnergy());
                for (int i = 0; i < energyAt.length; i++) {
                        energyAt[i] = element.getEnergyAt()[i];
                }
                for (Point p :element.getGraphPoints()) {
                        this.graphPoints.add(new Point((int)p.getX(), (int)p.getY()));
                }
                setSav("CVS");
                setObj(element.getObj());
                setId(IdCounterElem.nextId());
        }
```

A.3.2 Bilder

Zum anderem Nutzen wir für die Bilder folgenden Ansatz. Abgesehen von den Default-Objekte hat der Nutzer die Möglichkeit eigene Objekte zu erstellen, bei dem er/sie ein Bild für jedes neu erzeugte Objekt auswählen muss. Diese Bilder werden einmalig in das Programm geladen, sodass alle zukünftige Kopien dieses Objekts auf dieses Bild zugreifen.

Hier werden die Default-Objekte mit Ihren Bildern bei der Initialisierung instanziiert und in der View abgerufen.

```
                addNewHolonObject(mpC.searchCat("Energy"), "Power Plant", new ArrayList<HolonElement>(),
                                "/Images/power-plant.png");
                addNewHolonObject(mpC.searchCat("Building"), "House", new ArrayList<HolonElement>(),
                        "/Images/home-2.png");
                addNewHolonSwitch(mpC.searchCat("Component"), "Switch", "/Images/switch-on.png");
```

```
File checkPath = new File(cps.getImage());
if (checkPath.exists()) {
        imgR = new ImageIcon(cps.getImage()).getImage().getScaledInstance(50, 50,
        java.awt.Image.SCALE_SMOOTH);
} else {
        imgR = new ImageIcon(this.getClass().getResource(cps.getImage())).getImage().getScaledInstance(50, 50,
              java.awt.Image.SCALE_SMOOTH);
}
if (imgR != null) {
        label.setIcon(new ImageIcon(imgR));
}
```

A.4 Stabilität-Maßnahmen

A.4.1 Autosaves

Die Software erstellt bei Änderungen auf der Arbeitsfläche, ähnlich anderen Editoren, temporäre Autosaves. Diese ermöglichen es dem Benutzer ungewollten und/oder falschen Änderungen rückgängig zu machen, beziehungsweise wiederherzustellen. Um bei Programmabstürzen den Schaden zu minimieren, werden in solchen Fällen die Autosaves nicht gelöscht und dem Benutzer beim Neustart, falls gewollt, zur Verfügung gestellt.

```
            if (dest.listFiles().length > 1) {
                    int dialogButton = JOptionPane.YES_NO_OPTION;
                    int dialogResult = JOptionPane.showConfirmDialog(null, "Old autosave file was found, should it
                         be loaded?",
                                    "Warning", dialogButton);
                    if (dialogResult == JOptionPane.YES_OPTION) {
                            model.setAutoSaveNr(dest.listFiles().length - 1);
                            mntmRedo.doClick();
                    } else {
                            controller.deleteDirectory(dest);
                            dest.mkdirs();
                            try {
                                    controller.autoSave();
                            } catch (IOException e1) {
                                    // TODO Auto-generated catch block
                                    e1.printStackTrace();
                            }
                    }

            }
    }
```

A.4.2 Automatisierte Tests

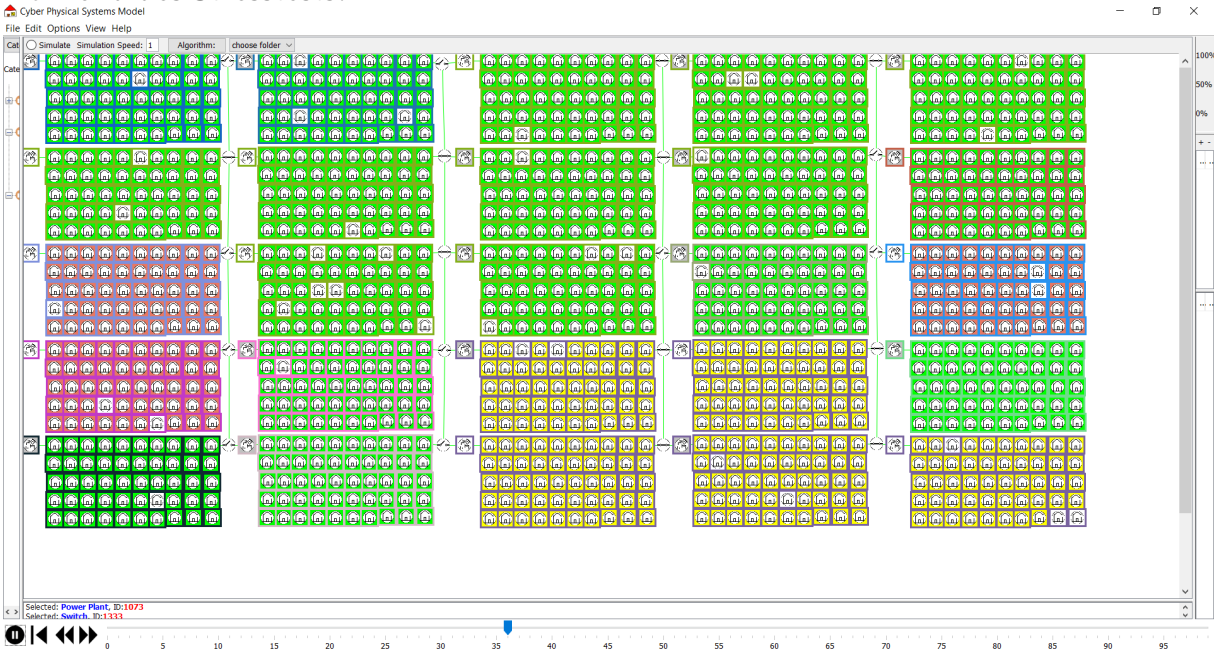
Seit Anfang des Projekts haben wir es uns zur Aufgabe gemacht, alle unsere Klassen über das JUnit-Framework zu testen. Da unser Buildprozess über Gradle abläuft, müssen erst alle Tests bestanden werden, bevor unser Projekt überhaupt gebuildet werden kann. Mit den Tests stellen wir sicher, dass alle neu Implementierte Klassen funktionieren und es zu keinen Fehlverhalten kommt.

A.4.3 Stresssimulation

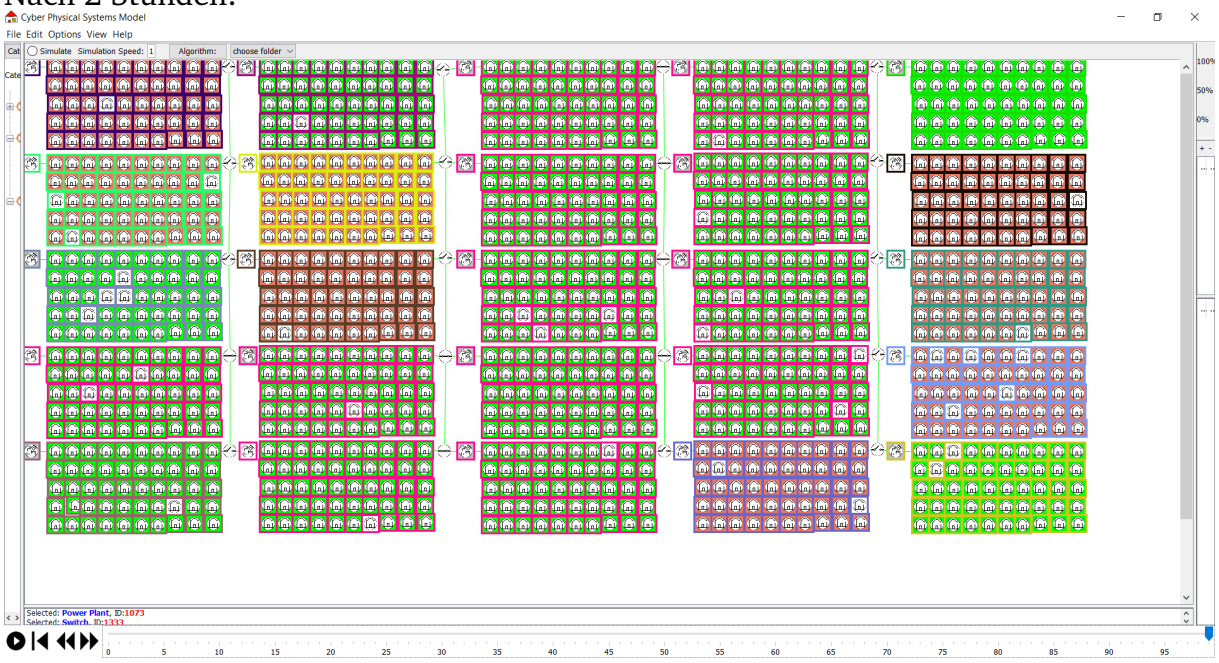
Für den Stresstest haben wir Manuel 1337 Objekte erstellt und sie zu einem Großen Netzwerk verbunden. Zudem haben wir einen Algorithmus geschrieben, der nach Jedem Schritt allen Objekt vom selben Subnetz die Selbe Randfarbe, welche zufällig ermittelt wird, zuteilt. Zusätzlich werden die Werte aller Elemente in allen Objekten zufällig verändert und alle Schalter werden zufällig an oder ausgeschaltet. Nachdem die Simulation durchgelaufen ist wird sie automatisch wieder neu gestartet. Durch den Algorithmus wird garantiert, dass jeder Durchlauf sich von den anderen unterscheidet und decken viele mögliche Fälle ab, die bei der Simulation auftreten können. Die Dauer des Stresstests beträgt 2 Stunden.

Am Anfang der Stresssimulation:

Während des Stresstests:



Nach 2 Stunden:



Während den 2 Stunden lief das Programm ohne Probleme und es ist kein Fehler aufgetreten. Zudem gab es keinen merklichen Performance Verlust. Alle Objekte wurden auch nach 2 Stunden korrekt dargestellt und die Subnetze wurden in der richtig Farbe eingefärbt. Daraus können wir schließen, dass Nutzer mit Netzwerken dieser Größenordnung keine Probleme haben werden.

A.4.3.1 Liste aller Tests

```
package tests;

import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;

import classes.HolonObject;
import ui.controller.CategoryController;
import ui.controller.MultiPurposeController;
import ui.model.Model;

/**
 * Tests for the CategoryController.
 *
 * @author Gruppe14
 */
public class PraktikumHolonsTestCategoryController {

        protected PraktikumHolonsAdapter adapter;
        protected Model model;
        protected MultiPurposeController mp;
        protected CategoryController controller;

        /**
         * Setup for the Tests.
         */
        @Before
        public void setUp() {
                adapter = new PraktikumHolonsAdapter();
                model = new Model();
                mp = new MultiPurposeController(model);
                controller = new CategoryController(model, mp);
        }
```

```
/**
 * tests for the Initial Categories.
 */
@Test
public void testInitialCategories() {
        assertTrue("Number of Categories is not 3", model.getCategories().size() == 3);
        assertTrue("Second Category is not Building",
             model.getCategories().get(1).getName().equals("Building"));
        assertTrue("Category Building is Empty", !model.getCategories().get(1).getObjects().isEmpty());
        assertEquals("Object is not a Power Plant", mp.searchCat("Energy").getObjects().get(0).getObjName(),
                        "Power Plant");
        assertFalse("A Switch should not be a Holon Object",
                        mp.searchCatObj(mp.searchCat("Component"), "Switch") instanceof HolonObject);
}

/**
 * Basic tests for adding new Categories.
 */
@Test
public void testAddingCategoriesMinimal() {
        controller.addNewCategory("University");
        controller.addNewCategory("Hospital");
        assertTrue("Number of Categories is not 5", model.getCategories().size() == 5);
        controller.addNewCategory("Energy");
        assertTrue("Number of Categories is not 6", model.getCategories().size() == 6);
        assertTrue("Name of the Duplicate: Energy was not changed to Energy_0",
                        model.getCategories().get(5).getName().equals("Energy_0"));
        controller.addNewCategory("Energy");
        assertTrue("Number of Categories is not 7", model.getCategories().size() == 7);
        assertTrue("Name of the Duplicate: Energy was not changed to Energy_1",
                        model.getCategories().get(6).getName().equals("Energy_1"));
}

/**
 * Basic tests for deleting Categories.
 */
@Test
public void testDeletingCategoriesMinimal() {
        assertTrue("Number of Categories is not 3", model.getCategories().size() == 3);
        assertTrue("2nd Category does not Match", model.getCategories().get(1).getName().equals("Building"));
        controller.deleteCategory("Building");
        assertTrue("Number of Categories is not 2", model.getCategories().size() == 2);
        assertTrue("Former 2nd Category was not deleted",
             model.getCategories().get(1).getName().equals("Component"));
        controller.deleteCategory("Energy");
        assertTrue("Number of Categories is not 1", model.getCategories().size() == 1);
        assertTrue("1st Category was not Component",
             model.getCategories().get(0).getName().equals("Component"));
}

/**
 * Extended tests for adding and deleting Categories.
 */
@Test
public void testAddingAndDeletingCategoriesExtended() {
        for (int i = 1; i <= 50; i++) {
                controller.addNewCategory(adapter.generate(i));
                assertTrue("Catagory:" + adapter.generate(i) + " was not added", model.getCategories().size()
                     == i + 3);
        }

        assertEquals("Category does not match", model.getCategories().get(29).getName(), "AA");
        controller.deleteCategory("AA");
        assertTrue("Catagory:AA was not deleted", model.getCategories().size() == 52);
        assertEquals("Category does not match", model.getCategories().get(29).getName(), "AB");
        assertEquals("Category does not match", model.getCategories().get(30).getName(), "AC");
        controller.deleteCategory("AB");
        assertTrue("Catagory:AB was not deleted", model.getCategories().size() == 51);
        assertEquals("Category does not match", model.getCategories().get(29).getName(), "AC");
        controller.deleteCategory("AD");
        assertTrue("Catagory:AD was not deleted", model.getCategories().size() == 50);
        assertEquals("Category does not match", model.getCategories().get(29).getName(), "AC");
        assertEquals("Category does not match", model.getCategories().get(30).getName(), "AE");
        controller.deleteCategory("Energy");
        assertTrue("Catagory:Energy was not deleted", model.getCategories().size() == 49);

        for (int i = 1; i <= 10; i++) {
                controller.deleteCategory(adapter.generate(i));
                assertTrue("Catagory was not deleted", model.getCategories().size() == 49 - i);
```

```
                }
                assertEquals("Category does not match", model.getCategories().get(3).getName(), "L");
        }

        /**
         * Basic tests for adding and deleting Objects.
         */
        @Test
        public void testAddingAndDeletingObjectsMinimal() {
                controller.addNewHolonObject(mp.searchCat("Energy"), "Power Plant", null, "");
                controller.addNewHolonObject(mp.searchCat("Energy"), "Power Plant", null, "");
                controller.addNewHolonObject(mp.searchCat("Energy"), "Solar Plant", null, "");

                assertTrue("Number of Objects in Energy is not 4", mp.searchCat("Energy").getObjects().size() == 4);
                assertTrue("Number of Object-Indices in Energy is not 4", mp.searchCat("Energy").getObjIdx().size() ==
                     4);
                assertTrue("Object was not renamed to \"Power Plant_0\" ",
                                mp.searchCat("Energy").getObjects().get(1).getObjName().equals("Power Plant_0"));
                assertTrue("Object was not renamed to \"Power Plant_1\" ",
                                mp.searchCat("Energy").getObjects().get(2).getObjName().equals("Power Plant_1"));
                assertTrue("3rd Object was not \"Power Plant_1\"",
                                mp.searchCat("Energy").getObjects().get(2).getName().equals("Power Plant_1"));
                controller.deleteObject("Energy", "Power Plant_1");
                assertTrue("3rd Object was not deleted",
                                mp.searchCat("Energy").getObjects().get(2).getName().equals("Solar Plant"));
                assertTrue("Number of Objects in Energy is not 3", mp.searchCat("Energy").getObjects().size() == 3);
                controller.addNewHolonObject(mp.searchCat("Energy"), "Solar Plant", null, "");
                assertTrue("Object was not renamed to \"Solar Plant_0\" ",
                                mp.searchCat("Energy").getObjects().get(3).getObjName().equals("Solar Plant_0"));
                assertTrue("Number of Objects in Energy is not 4", mp.searchCat("Energy").getObjects().size() == 4);
        }

        /**
         * Extended tests for adding and deleting Objects.
         */
        @Test
        public void testAddingAndDeletingObjectsExtended() {

                for (int i = 1; i <= 25; i++) {
                        controller.addNewHolonObject(mp.searchCat("Energy"), adapter.generate(i), null, null);
                        //creating duplicates
                        controller.addNewHolonObject(mp.searchCat("Energy"), adapter.generate(i), null, null);

                        assertTrue("Objects were not added", mp.searchCat("Energy").getObjects().size() == i * 2 + 1);

                        assertTrue("Object was not renamed to \"" + adapter.generate(i) + "_0\"", mp.searchCat("Energy")
                                        .getObjects().get(i * 2).getObjName().equals(adapter.generate(i) + "_0"));
                }
                //deleting the duplicates
                controller.deleteObject("Energy", "E_0");
                assertTrue("Object was not deleted", mp.searchCatObj(mp.searchCat("Energy"), "E_0") == null);
                controller.deleteObject("Energy", "F_0");
                assertTrue("Object was not deleted", mp.searchCatObj(mp.searchCat("Energy"), "F_0") == null);
                controller.deleteObject("Energy", "G_0");
                assertTrue("Object was not deleted", mp.searchCatObj(mp.searchCat("Energy"), "G_0") == null);
                controller.deleteObject("Energy", "H_0");
                assertTrue("Object was not deleted", mp.searchCatObj(mp.searchCat("Energy"), "H_0") == null);
                assertTrue("Number of Objects does not match", mp.searchCat("Energy").getObjects().size() == 47);
        }
}
```

```
package tests;

import classes.AbstractCpsObject;
import classes.CpsEdge;
import classes.HolonObject;
import classes.HolonSwitch;
import classes.IdCounter;
import ui.controller.CanvasController;
import ui.controller.CategoryController;
import ui.controller.MultiPurposeController;
import ui.model.Model;

import org.junit.Test;
import org.junit.Before;
import static org.junit.Assert.assertTrue;

import java.awt.Point;
/**
```

```
 * Tests for the CanvasController.
 *
 * @author Gruppe14
 */
public class PraktikumHolonsTestCanvasController {

	protected PraktikumHolonsAdapter adapter;
	protected Model model;
	protected MultiPurposeController mp;
	protected CategoryController cg;
	protected CanvasController controller;

	/**
	 * Setup for the tests.
	 */
	@Before
	public void setUp() {
		adapter = new PraktikumHolonsAdapter();
		model = new Model();
		mp = new MultiPurposeController(model);
		cg = new CategoryController(model, mp);
		controller = new CanvasController(model, mp);
		IdCounter.setCounter(1);
	}

	/**
	 * Tests adding objects.
	 */
	@Test
	public void testAddingObjects() {
		// just adding a few things
		assertTrue("Number of Objects does not Match", model.getObjectsOnCanvas().size() == 0);
		controller.addNewObject(new HolonObject(mp.searchCatObj(mp.searchCat("Energy"), "Power Plant")));
		assertTrue("ID of the Object does not Match", mp.searchByID(1).getName().equals("Power Plant"));
		assertTrue("Type of the Object does not Match", mp.searchByID(1) instanceof HolonObject);
		assertTrue("Number of Objects does not Match", model.getObjectsOnCanvas().size() == 1);
		controller.addNewObject(new HolonObject(mp.searchCatObj(mp.searchCat("Energy"), "Power Plant")));
		assertTrue("ID of the Object does not Match", mp.searchByID(2).getName().equals("Power Plant"));
		assertTrue("Type of the Object does not Match", mp.searchByID(2) instanceof HolonObject);
		assertTrue("Number of Objects does not Match", model.getObjectsOnCanvas().size() == 2);
		controller.addNewObject(new HolonObject(mp.searchCatObj(mp.searchCat("Energy"), "Power Plant")));
		assertTrue("ID of the Object does not Match", mp.searchByID(3).getName().equals("Power Plant"));
		assertTrue("Type of the Object does not Match", mp.searchByID(3) instanceof HolonObject);
		assertTrue("Number of Objects does not Match", model.getObjectsOnCanvas().size() == 3);
		controller.addNewObject(new HolonObject(mp.searchCatObj(mp.searchCat("Energy"), "Power Plant")));
		assertTrue("Number of Objects does not Match", model.getObjectsOnCanvas().size() == 4);
		controller.addNewObject(new HolonObject(mp.searchCatObj(mp.searchCat("Energy"), "Power Plant")));
		assertTrue("Number of Objects does not Match", model.getObjectsOnCanvas().size() == 5);
		controller.addNewObject(new HolonObject(mp.searchCatObj(mp.searchCat("Building"), "House")));
		assertTrue("ID of the Object does not Match", mp.searchByID(6).getName().equals("House"));
		assertTrue("Type of the Object does not Match", mp.searchByID(6) instanceof HolonObject);
		assertTrue("Number of Objects does not Match", model.getObjectsOnCanvas().size() == 6);
		controller.addNewObject(new HolonObject(mp.searchCatObj(mp.searchCat("Building"), "House")));
		assertTrue("Number of Objects does not Match", model.getObjectsOnCanvas().size() == 7);
		controller.addNewObject(new HolonObject(mp.searchCatObj(mp.searchCat("Building"), "House")));
		assertTrue("Number of Objects does not Match", model.getObjectsOnCanvas().size() == 8);
		controller.addNewObject(new HolonSwitch(mp.searchCatObj(mp.searchCat("Component"), "Switch")));
		assertTrue("ID of the Object does not Match", mp.searchByID(9).getName().equals("Switch"));
		assertTrue("Type of the Object does not Match", mp.searchByID(9) instanceof HolonSwitch);
		assertTrue("Number of Objects does not Match", model.getObjectsOnCanvas().size() == 9);
		controller.addNewObject(new HolonSwitch(mp.searchCatObj(mp.searchCat("Component"), "Switch")));
		assertTrue("Number of Objects does not Match", model.getObjectsOnCanvas().size() == 10);
	}

	/**
	 * Tests deleting Objects.
	 */
	@Test(expected = NullPointerException.class)
	public void testDeletingObject() {
		// Adding Objects on Canvas.. without Coordinates
		controller.addNewObject(new HolonObject(mp.searchCatObj(mp.searchCat("Energy"), "Power Plant")));
		controller.addNewObject(new HolonObject(mp.searchCatObj(mp.searchCat("Energy"), "Power Plant")));
		controller.addNewObject(new HolonObject(mp.searchCatObj(mp.searchCat("Building"), "House")));
		controller.addNewObject(new HolonObject(mp.searchCatObj(mp.searchCat("Energy"), "Power Plant")));
		controller.addNewObject(new HolonObject(mp.searchCatObj(mp.searchCat("Energy"), "Power Plant")));
		controller.addNewObject(new HolonObject(mp.searchCatObj(mp.searchCat("Building"), "House")));
		controller.addNewObject(new HolonSwitch(mp.searchCatObj(mp.searchCat("Component"), "Switch")));
		controller.addNewObject(new HolonObject(mp.searchCatObj(mp.searchCat("Energy"), "Power Plant")));
		controller.addNewObject(new HolonObject(mp.searchCatObj(mp.searchCat("Building"), "House")));
		controller.addNewObject(new HolonSwitch(mp.searchCatObj(mp.searchCat("Component"), "Switch")));
```

```
        controller.deleteObjectOnCanvas(mp.searchByID(4));
        assertTrue("Object:4 was not deleted", mp.searchByID(4) == null);
        assertTrue("Number of Objects does not Match", model.getObjectsOnCanvas().size() == 9);
        controller.deleteObjectOnCanvas(mp.searchByID(5));
        assertTrue("Object:4 was not deleted", mp.searchByID(5) == null);
        assertTrue("Number of Objects does not Match", model.getObjectsOnCanvas().size() == 8);
        controller.deleteObjectOnCanvas(mp.searchByID(6));
        assertTrue("Object:4 was not deleted", mp.searchByID(6) == null);
        assertTrue("Number of Objects does not Match", model.getObjectsOnCanvas().size() == 7);

        // deleting Non-Existant Object -> NullPointerexception
        controller.deleteObjectOnCanvas(mp.searchByID(4));
    }

    /**
     * Tests adding and deleting Edges.
     */
    @Test
    public void testAddingAndDeletingEdges() {
        HolonObject a = new HolonObject("A");
        controller.addNewObject(new HolonObject(a));
        int n = 0;

        // creates vertices A - Z
        for (int i = 2; i < 27; i++) {
            // adds Object on canvas
            HolonObject temp = new HolonObject(adapter.generate(i));
            controller.addNewObject(new HolonObject(temp));
            // connect current vertice with all other vertices
            for (AbstractCpsObject cps : model.getObjectsOnCanvas()) {
                if (!cps.equals(mp.searchByID(i)))
                    controller.addEdgeOnCanvas(new CpsEdge(mp.searchByID(i), cps));
            }

            // test how many connections current vertice got
            assertTrue("Number of Connections does not Match", mp.searchByID(i).getConnectedTo().size() ==
                i - 1);
            // actually just means if its a
            // complete graph -> all vertices connected all other vertices
            n = model.getObjectsOnCanvas().size();
            assertTrue("Number of Edges does not Match", model.getEdgesOnCanvas().size() == (n * (n - 1)) /
                2);
        }
        // same as above
        n = model.getObjectsOnCanvas().size();
        assertTrue("Number of Edges does not Match", model.getEdgesOnCanvas().size() == (n * (n - 1)) / 2);

        // here starts the deleting
        controller.removeEdgesOnCanvas(mp.searchEdge(13, 14));
        assertTrue("Number of Connection of Vertice M does not Match",
                mp.searchByID(13).getConnections().size() == model.getObjectsOnCanvas().size() - 2);
        assertTrue("Edge-M-N was not deleted", mp.searchEdge(13, 14) == null);

        controller.deleteObjectOnCanvas(mp.searchByID(13));
        assertTrue("Object:13 was not deleted", mp.searchByID(13) == null);
        assertTrue("Edge-A-M was not deleted", mp.searchEdge(1, 13) == null);
        assertTrue("Edge-B-M was not deleted", mp.searchEdge(2, 13) == null);
        assertTrue("Edge-C-M was not deleted", mp.searchEdge(3, 13) == null);
        assertTrue("Edge-D-M was not deleted", mp.searchEdge(4, 13) == null);
        assertTrue("Edge-E-M was not deleted", mp.searchEdge(5, 13) == null);
        assertTrue("Edge-F-M was not deleted", mp.searchEdge(6, 13) == null);
        assertTrue("Edge-M-O was not deleted", mp.searchEdge(13, 16) == null);
        assertTrue("Edge-M-P was not deleted", mp.searchEdge(13, 17) == null);
        assertTrue("Edge-M-Q was not deleted", mp.searchEdge(13, 18) == null);
        assertTrue("Edge-M-R was not deleted", mp.searchEdge(13, 19) == null);
        assertTrue("Edge-M-S was not deleted", mp.searchEdge(13, 20) == null);
    }

    /**
     * Test copying, cutting and pasting Objects.
     */
    @Test
    public void testCutCopyPasteObjects() {
        HolonObject a = new HolonObject("A");
        HolonObject b = new HolonObject("B");
        HolonObject c = new HolonObject("C");

        a = new HolonObject(a);
        a.setPosition(1, 1);
```

```
                b = new HolonObject(b);
                b.setPosition(2, 2);
                c = new HolonObject(c);
                c.setPosition(3, 3);
                CpsEdge edge1 = new CpsEdge(a, b);
                CpsEdge edge2 = new CpsEdge(b, c);
                CpsEdge edge3 = new CpsEdge(c, a);
                controller.addNewObject(a);
                controller.addEdgeOnCanvas(edge1);
                controller.addEdgeOnCanvas(edge2);
                controller.addEdgeOnCanvas(edge3);


                assertTrue("Clipboard not empty", model.getClipboradObjects().isEmpty());
                model.getSelectedCpsObjects().add(a);
                model.getSelectedCpsObjects().add(b);
                model.getSelectedCpsObjects().add(c);
                controller.copyObjects();
                assertTrue("Clipboard empty", !model.getClipboradObjects().isEmpty());

                assertTrue("Clipboard empty", !model.getClipboradObjects().isEmpty());

                controller.pasteObjects(new Point(1, 1));

                controller.addNewObject(a);
                controller.addNewObject(b);
                controller.addNewObject(c);
                controller.cutObjects();
                assertTrue("Objects still on canvas", model.getObjectsOnCanvas().isEmpty());

        }
}
```

```
package tests;

import static org.junit.Assert.assertTrue;

import org.junit.Before;
import org.junit.Test;

import classes.AbstractCpsObject;
import classes.HolonObject;
import ui.controller.CanvasController;
import ui.controller.CategoryController;
import ui.controller.MultiPurposeController;
import ui.controller.ObjectController;
import ui.model.Model;

/**
 * Tests for the ObjectController.
 *
 * @author Gruppe14
 */
public class PraktikumHolonsTestObjectController {

        protected PraktikumHolonsAdapter adapter;
        protected Model model;
        protected MultiPurposeController mp;
        protected CategoryController cg;
        protected CanvasController cvs;
        protected ObjectController controller;

        /**
         * Setup for the Tests.
         */
        @Before
        public void setUp() {
                adapter = new PraktikumHolonsAdapter();
                model = new Model();
                mp = new MultiPurposeController(model);
                cg = new CategoryController(model, mp);
                cvs = new CanvasController(model, mp);
                controller = new ObjectController(model, mp);
        }

        /**
         * Tests for the Initial HolonElements.
         */
        @Test
```

```
    public void testInitialHolonElements() {
        assertTrue("Number of Elements does not Match",
                ((HolonObject) mp.searchCatObj(mp.searchCat("Energy"), "Power
                    Plant")).getElements().size() == 1);
        assertTrue("Element does not Match",
                mp.searchEle((HolonObject) mp.searchCatObj(mp.searchCat("Energy"), "Power Plant"),
                    "Power").getEleName()
                            .equals("Power"));
        assertTrue("Element does not Match",
                mp.searchEle((HolonObject) mp.searchCatObj(mp.searchCat("Energy"), "Power Plant"),
                    "Power").getEleName()
                            .equals("Power"));
        assertTrue("Total Energy does not Match",
                mp.searchEle((HolonObject) mp.searchCatObj(mp.searchCat("Building"), "House"), "PC")
                            .getTotalEnergy() == -750);
        assertTrue("Non-Existant Element is Found",
                mp.searchEle((HolonObject) mp.searchCatObj(mp.searchCat("Building"), "House"), "") ==
                    null);
    }

    /**
     * Tests for adding and Deleting in Categories.
     */
    @Test
    public void testAddingAndDeletingInCategory() {
        controller.addNewElementIntoCategoryObject("Building", "House", "A", 1, -10);
        for (int i = 2; i < 27; i++) {
            controller.addNewElementIntoCategoryObject("Building", "House", adapter.generate(i), i, -10);
            // n(n+1) / 2
            assertTrue("Total Energy does not match", ((HolonObject)
                mp.searchCatObj(mp.searchCat("Building"), "House"))
                        .getCurrentEnergy() == -1800 + ((i * (i + 1)) / 2) * -10);
            assertTrue("Number of Elements does not Match",
                    ((HolonObject) mp.searchCatObj(mp.searchCat("Building"),
                        "House")).getElements().size() == 6 + i);
        }

        controller.deleteElementInCategory("Building", "House", "B");
        controller.deleteElementInCategory("Building", "House", "D");
        controller.deleteElementInCategory("Building", "House", "F");
        controller.deleteElementInCategory("Building", "House", "G");
        controller.deleteElementInCategory("Building", "House", "H");
        controller.deleteElementInCategory("Building", "House", "I");
        controller.deleteElementInCategory("Building", "House", "Z");
        controller.deleteElementInCategory("Building", "House", "TV");
        assertTrue("Element:B was Found",
                mp.searchEle((HolonObject) mp.searchCatObj(mp.searchCat("Building"), "House"), "B") ==
                    null);
        assertTrue("Element:D was Found",
                mp.searchEle((HolonObject) mp.searchCatObj(mp.searchCat("Building"), "House"), "D") ==
                    null);
        assertTrue("Element:F was Found",
                mp.searchEle((HolonObject) mp.searchCatObj(mp.searchCat("Building"), "House"), "F") ==
                    null);
        assertTrue("Element:G was Found",
                mp.searchEle((HolonObject) mp.searchCatObj(mp.searchCat("Building"), "House"), "G") ==
                    null);
        assertTrue("Element:H was Found",
                mp.searchEle((HolonObject) mp.searchCatObj(mp.searchCat("Building"), "House"), "H") ==
                    null);
        assertTrue("Element:I was Found",
                mp.searchEle((HolonObject) mp.searchCatObj(mp.searchCat("Building"), "House"), "I") ==
                    null);
        assertTrue("Element:Z was Found",
                mp.searchEle((HolonObject) mp.searchCatObj(mp.searchCat("Building"), "House"), "Z") ==
                    null);
        assertTrue("Element:TV was Found",
                mp.searchEle((HolonObject) mp.searchCatObj(mp.searchCat("Building"), "House"), "TV") ==
                    null);
    }

    /**
     * Tests for Adding and Deleting Objects on the Canvas.
     */
    @Test
    public void testAddingAndDeletingInCanvas() {
        for (int i = 0; i < 100; i++) {
            cvs.addNewObject(new HolonObject(mp.searchCatObj(mp.searchCat("Building"), "House")));
        }
        for (AbstractCpsObject cps : model.getObjectsOnCanvas()) {
```

```
                        for (int i = 0; i < 27; i++) {
                                controller.addNewElementIntoCanvasObject(cps.getID(), adapter.generate(i), 1, -100);
                                assertTrue("Element:" + adapter.generate(i) + " was not Created", mp
                                                .searchEle((HolonObject) mp.searchByID(cps.getID()),
                                                        adapter.generate(i)) != null);
                        }
                        assertTrue("Element:B was not Found", mp
                                        .searchEle((HolonObject) mp.searchByID(cps.getID()), "B") != null);
                        assertTrue("Element:D was not Found", mp
                                        .searchEle((HolonObject) mp.searchByID(cps.getID()), "D") != null);
                        assertTrue("Element:F was not Found", mp
                                        .searchEle((HolonObject) mp.searchByID(cps.getID()), "F") != null);
                        assertTrue("Element:G was not Found", mp
                                        .searchEle((HolonObject) mp.searchByID(cps.getID()), "G") != null);
                        assertTrue("Element:H was not Found", mp
                                        .searchEle((HolonObject) mp.searchByID(cps.getID()), "H") != null);
                        assertTrue("Element:I was not Found", mp
                                        .searchEle((HolonObject) mp.searchByID(cps.getID()), "I") != null);
                        assertTrue("Element:B was not Found", mp
                                        .searchEle((HolonObject) mp.searchByID(cps.getID()), "B") != null);
                }

                for (AbstractCpsObject cps : model.getObjectsOnCanvas()) {
                        int size = model.getSelectedCpsObjects().size();
                        controller.addSelectedObject(cps);
                        assertTrue("Size does not Match", model.getSelectedCpsObjects().size() == size +1);
                }
                for (AbstractCpsObject cps : model.getObjectsOnCanvas()) {
                        System.out.println(model.getSelectedCpsObjects().size());
                        int size = model.getSelectedCpsObjects().size();
                        controller.deleteSelectedObject(cps);
                        assertTrue("Size does not Match", model.getSelectedCpsObjects().size() == size -1);
                        assertTrue("Object was not unselected", !model.getSelectedCpsObjects().contains(cps));

                }
        }



}
```

```
package tests;

import java.awt.Point;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;
import java.util.LinkedList;

import org.junit.Before;
import org.junit.Test;
import classes.Category;
import classes.CpsEdge;

import classes.AbstractCpsObject;

import classes.HolonElement;
import classes.HolonObject;
import classes.HolonSwitch;
import classes.IdCounter;

import static org.junit.Assert.assertTrue;

import ui.controller.CanvasController;
import ui.controller.CategoryController;
import ui.controller.LoadController;
import ui.controller.MultiPurposeController;
import ui.controller.ObjectController;
import ui.controller.StoreController;
import ui.model.Model;
import ui.view.UnitGraph;

/**
 * Tests for LoadAndStoreController.
 *
 * @author Gruppe14
 */
public class PraktikumHolonsTestLoadAndStoreController {
```

```
protected PraktikumHolonsAdapter adapter;
protected Model model;
protected MultiPurposeController mp;
protected CategoryController cg;
protected CanvasController cvs;
protected ObjectController obj;
protected StoreController storeController;
protected LoadController loadController;
protected IdCounter id;
protected String path = System.getProperty("user.home") + "/HolonGUI/Test/";

/**
 * Setup for the Tests.
 */
@Before
public void setUp() {
        adapter = new PraktikumHolonsAdapter();
        model = new Model();
        mp = new MultiPurposeController(model);
        cg = new CategoryController(model, mp);
        cvs = new CanvasController(model, mp);
        obj = new ObjectController(model, mp);
        storeController = new StoreController(model);
        loadController = new LoadController(model, cg, cvs, obj, mp);
        // cg.initCategories();
        // obj.initHolonElements();
        File file = new File(path);
        file.mkdirs();
}

/**
 * minimal tests for saving.
 */
@Test
public void testStoreMinimal() {
        try {
                File sav = new File(path + "TestSavMinimal.json");
                storeController.writeSaveFile(sav.getAbsolutePath());
                assertTrue("Save File was not created", new File(path + "TestSavMinimal.json").exists());

                File cat = new File(path + "TestCategoryMinimal.json");
                storeController.writeCategoryFile(cat.getAbsolutePath());
                assertTrue("Category File was not created", new File(path +
                     "TestCategoryMinimal.json").exists());

                File canvas = new File(path + "TestCanvasMinimal.json");
                storeController.writeCanvasFile(canvas.getAbsolutePath());
                assertTrue("Canvas File was not created", new File(path + "TestCanvasMinimal.json").exists());

                assertTrue("Non Existant File exist", !new File(path + "TestDummyMinimal.json").exists());

        } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
}

/**
 * basic tests for saving.
 */
@Test
public void testStoreBasic() {

        for (int i = 1; i <= 26; i++) {
                cg.addNewCategory(adapter.generate(i));
                assertTrue("Number of Categories does not match", model.getCategories().size() == i + 3);
                assertTrue("Category was not created", mp.searchCat(adapter.generate(i)) != null);
                for (int j = 1; j <= 10; j++) {
                        cg.addNewHolonObject(mp.searchCat(adapter.generate(i)), adapter.generate(j),
                                        new ArrayList<HolonElement>(), "");
                        assertTrue("Number of Objects does not match",
                                        mp.searchCat(adapter.generate(i)).getObjects().size() == j);
                        assertTrue("Object was not created",
                             mp.searchCat(adapter.generate(i)).getObjects().get(j - 1) != null);
                }
        }

        // here some Objects were dropped on the canvas
        for (int i = 1; i <= 10; i++) {
```

```
                cvs.addNewObject(new HolonObject(mp.searchCatObj(mp.searchCat("Building"), "House")));
                assertTrue("Size of Objects on Canvas does not match", model.getObjectsOnCanvas().size() == i);
                assertTrue("Object was not added", model.getObjectsOnCanvas().get(i - 1) != null
                                && model.getObjectsOnCanvas().get(i - 1).getObjName().equals("House"));
        }

        HolonSwitch sw = new HolonSwitch(mp.searchCatObj(mp.searchCat("Component"), "Switch"));
        sw.setPosition(77, 88);
        cvs.addNewObject(sw);

        try {
                File sav = new File(path + "TestSavBasic.json");
                storeController.writeSaveFile(sav.getAbsolutePath());
                assertTrue("Save File was not created", new File(path + "TestSavBasic.json").exists());

                File canvas = new File(path + "TestCanvasBasic.json");
                storeController.writeCanvasFile(canvas.getAbsolutePath());
                assertTrue("Canvas File was not created", new File(path + "TestCanvasBasic.json").exists());

                assertTrue("Non Existant File exist", !new File(path + "TestDummyMinimal.json").exists());

        } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
}

/**
 * advanced tests for saving.
 */
@Test
public void testStoreAdvanced() {

        setGraphPoints((HolonObject) mp.searchCatObj(mp.searchCat("Building"), "House"));

        int n = 0;
        for (int i = 0; i < 10; i++) {
                for (int j = 0; j < 10; j++) {
                        HolonObject h = new HolonObject(mp.searchCatObj(mp.searchCat("Building"), "House"));
                        h.setPosition(j * 50, i * 50);
                        cvs.addNewObject(h);
                        setGraphPoints(h);
                        for (AbstractCpsObject cps : model.getObjectsOnCanvas()) {
                                if (!cps.equals(h))
                                        cvs.addEdgeOnCanvas(new CpsEdge(h, cps));
                        }

                        // complete Graph
                        n = model.getObjectsOnCanvas().size();
                        assertTrue("Number of Edges does not Match", model.getEdgesOnCanvas().size() == (n * (n
                            - 1)) / 2);
                }
        }

        try {
                File sav = new File(path + "TestSavAdvanced.json");
                storeController.writeSaveFile(sav.getAbsolutePath());
                assertTrue("Save File was not created", new File(path + "TestSavAdvanced.json").exists());

        } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }

}

/**
 * minimal tests for loading a save file.
 */
@Test
public void testLoadMinimal() {
        try {
                // Category Tests
                Category building = mp.searchCat("Building");
                assertTrue("Number of Categories does not match", model.getCategories().size() == 3);
                assertTrue("TestFile was not found", new File(path + "TestCategoryMinimal.json").exists());
                loadController.readJson(path + "TestCategoryMinimal.json");
                assertTrue("Number of Categories does not match", model.getCategories().size() == 3);
                // Tests if its same instance and if Name is the same
                assertTrue("Same instance of Category:Building", building != mp.searchCat("Building")
```

```
                        && building.getName().equals(mp.searchCat("Building").getName()));

                // Canvas Tests.. basically nothing happens because nothing is on
                // the Canvas
                assertTrue("Number of Objects on Canvas does not match", model.getObjectsOnCanvas().size() ==
                    0);
                assertTrue("TestFile was not found", new File(path + "TestCanvasMinimal.json").exists());
                loadController.readJson(path + "TestCanvasMinimal.json");
                assertTrue("Number of Objects on Canvas does not match", model.getObjectsOnCanvas().size() ==
                    0);

                // Save File tests basically both Test from Above Combined
                building = mp.searchCat("Building");
                assertTrue("Number of Objects in Energy does not Match",
                    mp.searchCat("Energy").getObjects().size() == 1);
                assertTrue("TestFile was not found", new File(path + "TestSavMinimal.json").exists());
                loadController.readJson(path + "TestSavMinimal.json");
                assertTrue("Number of Objects in Energy does not Match",
                    mp.searchCat("Energy").getObjects().size() == 1);
                assertTrue("Number of Objects on Canvas does not match", model.getObjectsOnCanvas().size() ==
                    0);
                assertTrue("Same instance of Category:Building", building != mp.searchCat("Building")
                        && building.getName().equals(mp.searchCat("Building").getName()));

        } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
}

/**
 * basic tests for loading a save file.
 */
@Test
public void testLoadBasic() {
        assertTrue("Non-Existant Category Exists", mp.searchCat("J") == null);
        assertTrue("Non-Existant Category Exists", mp.searchCat("U") == null);
        assertTrue("Non-Existant Category Exists", mp.searchCat("L") == null);
        assertTrue("Non-Existant Category Exists", mp.searchCat("I") == null);
        assertTrue("Non-Existant Category Exists", mp.searchCat("A") == null);
        assertTrue("Non-Existant Category Exists", mp.searchCat("N") == null);

        try {
                assertTrue("Objects on Canvas is not 0", model.getObjectsOnCanvas().size() == 0);
                assertTrue("Canvas File was not found", new File(path + "TestCanvasBasic.json").exists());
                loadController.readJson(path + "TestCanvasBasic.json");

                assertTrue("NUmber of Objects on Canvas does not match", model.getObjectsOnCanvas().size() ==
                    11);
                for (AbstractCpsObject obj : model.getObjectsOnCanvas()) {
                    assertTrue("Not instance of HolonObject", obj instanceof HolonObject || obj instanceof
                        HolonSwitch);

                }

                assertTrue("NUmber of Categories not match", model.getCategories().size() == 3);
                assertTrue("Canvas File was not found", new File(path + "TestSavBasic.json").exists());
                loadController.readJson(path + "TestSavBasic.json");
                assertTrue("NUmber of Categories not match", model.getCategories().size() == 29);
                assertTrue("Existant Category dont Exists", mp.searchCat("J") != null);
                assertTrue("Existant Category dont Exists", mp.searchCat("U") != null);
                assertTrue("Existant Category dont Exists", mp.searchCat("L") != null);
                assertTrue("Existant Category dont Exists", mp.searchCat("I") != null);
                assertTrue("Existant Category dont Exists", mp.searchCat("A") != null);
                assertTrue("Existant Category dont Exists", mp.searchCat("N") != null);
                assertTrue("Existant Object dont Exists", mp.searchCatObj(mp.searchCat("Z"), "A") != null);
                assertTrue("Existant Object dont Exists", mp.searchCatObj(mp.searchCat("Z"), "B") != null);
                assertTrue("Existant Object dont Exists", mp.searchCatObj(mp.searchCat("Z"), "C") != null);
                assertTrue("Existant Object dont Exists", mp.searchCatObj(mp.searchCat("Z"), "D") != null);
                assertTrue("Existant Object dont Exists", mp.searchCatObj(mp.searchCat("Z"), "E") != null);
                assertTrue("Existant Object dont Exists", mp.searchCatObj(mp.searchCat("Z"), "F") != null);

        } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
}

/**
 * advanced tests for loading a save file.
```

```
         */
        @Test
        public void testLoadAdvanced() {

                try {
                        assertTrue("Objects on Canvas is not 0", model.getObjectsOnCanvas().size() == 0);
                        assertTrue("Save File was not found", new File(path + "TestSavAdvanced.json").exists());
                        loadController.readJson(path + "TestSavAdvanced.json");
                        assertTrue("Objects on Canvas is not 100", model.getObjectsOnCanvas().size() == 100);

                        int n = model.getObjectsOnCanvas().size();
                        assertTrue("Number of Edges does not Match", model.getEdgesOnCanvas().size() == (n * (n - 1)) /
                             2);
                        assertTrue("Element has no UnitGraph", !mp
                                        .searchEle((HolonObject) model.getObjectsOnCanvas().get(77),
                                             "Fridge").getGraphPoints().isEmpty());
                        assertTrue("Points in UnitGraph does not Match",
                                        mp.searchEle((HolonObject) model.getObjectsOnCanvas().get(77),
                                             "Fridge").getGraphPoints()
                                                        .size() == 3);

                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
        }

        /**
         * Test for FileNotFound.
         *
         * @throws IOException
         *             FileNotFoundException
         */
        @Test(expected = FileNotFoundException.class)
        public void testLoadException() throws IOException {
                assertTrue("Save File was not found", !new File(path + "TestSavDummy.json").exists());
                loadController.readJson(path + "TestSavDummy.json");
        }

        /**
         * sets the graph points in all elements of an Object.
         *
         * @param obj
         *            the Object
         */
        public void setGraphPoints(HolonObject obj) {
                LinkedList<Point> list = new LinkedList<>();
                list.add(new Point(0, 0));
                list.add(new Point(125, 50));
                list.add(new Point(249, 0));
                UnitGraph u = new UnitGraph(model, null);
                u.repaintWithNewElement(obj.getElements());
                u.fillArrayofValue();
                for (HolonElement ele : obj.getElements()) {
                        ele.setGraphPoints(list);
                }

        }
}
```

```
package tests;

import org.junit.Before;
import org.junit.Test;
import static org.junit.Assert.assertTrue;

import java.util.ArrayList;

import ui.controller.GlobalController;
import ui.model.Model;

/**
 * Test for the GlobalController.
 *
 * @author Gruppe14
 */
public class PraktikumHolonsTestGlobalController {

        protected Model model;
```

```
        protected GlobalController controller;

        /**
         * Setup.
         */
        @Before
        public void setUp() {
                model = new Model();
                controller = new GlobalController(model);

        }

        /**
         * Test for GlobalControls.
         */
        @Test
        public void testGlobalControls() {
                int prevScale = controller.getScale();
                int prevScaleDiv2 = controller.getScaleDiv2();
                int prevNumberSav = controller.getNumbersOfSaves();
                boolean sim = model.getIsSimulation();
                int timer = model.getTimerSpeed();
                int it = model.getCurIteration();

                controller.setScale(100);
                controller.setNumberOfSaves(50);
                controller.setIsSimulation(true);
                controller.setTimerSpeed(2000);
                controller.setCurIteration(10);

                assertTrue("Scale was not changed", controller.getScale() != prevScale);
                assertTrue("ScaleDiv2 was not changed ", model.getScaleDiv2() != prevScaleDiv2);
                assertTrue("Number of Saves was not changed", controller.getNumbersOfSaves() != prevNumberSav);
                assertTrue("Simulation State was not Set", sim != model.getIsSimulation());
                assertTrue("Timer speed was not changed", timer != model.getTimerSpeed());
                assertTrue("Curr Iteration was not Set", it != model.getCurIteration());

        }

}
```

```
package tests;

import org.junit.Before;
import org.junit.Test;
import static org.junit.Assert.*;

import ui.controller.AutoSaveController;
import ui.model.Model;
/**
 * Tests for the AutoSaveController.
 *
 * @author Gruppe14
 */
public class PraktikumHolonsTestAutoSaveController {
        protected Model model;
        protected AutoSaveController controller;


        /**
         * Setup for the test.
         */
        @Before
        public void setUp() {
                model = new Model();
                controller = new AutoSaveController(model);
        }

        /**
         *  kommentar hier hin.
         */
        @Test
        public void testAutoSave() {
                int temp = controller.getAutoSaveNr();
                assertTrue("AutoSave Number not forrect", temp == -1);
                controller.increaseAutoSaveNr();
                assertTrue("AutoSave Number not forrect", controller.getAutoSaveNr() == 0);

                for (int i = 0; i < 20; i++) {
```

```
                        controller.increaseAutoSaveNr();
                }

                assertTrue("Is Allowed not set", controller.allowed());

                for (int i = 0; i < 20; i++) {
                        controller.decreaseAutoSaveNr();;
                }


        }
}
```

```
package tests;

import org.junit.Before;
import org.junit.Test;

import ui.controller.ConsoleController;
import ui.model.Model;
import ui.view.Console;

import java.awt.Color;

/**
 * Tests for the ConsoleController.
 *
 * @author Gruppe14
 */
public class PraktikumHolonsTestConsoleController {

        protected Model model;
        protected ConsoleController controller;

        /**
         * Setup.
         */
        @Before
        public void setUp() {
                model = new Model();
                model.setConsole(new Console());
                controller = new ConsoleController(model);
        }

        /**
         * Test for ConsoleController.
         */
        @Test
        public void testConsoleController() {
                controller.addTextToConsole("Hello World");
                controller.addTextToConsole("Hello World2!", new Color(255, 255, 255), 11, false, true, false);
                controller.clearConsole();

        }
}
```

```
package tests;

import classes.*;
import static org.junit.Assert.assertTrue;

import java.awt.Color;
import java.util.ArrayList;

import org.junit.Test;

/**
 * Test for the Classes.
 *
 * @author Gruppe14
 */
public class PraktikumHolonsTestClasses {

        /**
         * Test for the Categories.
         */
        @Test
        public void testCategory() {
```

```
        Category test1 = new Category("Test1");
        Category test2 = new Category("Test2");
        HolonObject obj1 = new HolonObject("Test Object");
        ArrayList<AbstractCpsObject> arr = new ArrayList<>();

        assertTrue("Name not correct", test1.getName().equals("Test1"));
        assertTrue("Name not correct", test2.getName().equals("Test2"));
        assertTrue("Name should not be the same", !test1.getName().equals(test2.getName()));
        assertTrue("Should be empty", test1.getObjects().size() == 0);
        test1.getObjects().add(obj1);
        assertTrue("Should not be empty", test1.getObjects().size() == 1);
        assertTrue("Should be empty", test2.getObjects().size() == 0);
        test1.setObjects(arr);
        assertTrue("Should be empty", test1.getObjects().size() == 0);
        arr.add(obj1);
        test2.setObjects(arr);
        arr = new ArrayList<>();
        assertTrue("Shoud be empty", arr.isEmpty());
        arr = test2.getObjects();
        assertTrue("Shoud not be empty", !arr.isEmpty());
    }

    /**
     * Test for HolonObject.
     */
    @Test
    public void testHolonObject() {

        HolonObject test1 = new HolonObject("Test1");
        HolonObject test2 = new HolonObject("Test2");
        HolonObject test3 = new HolonObject(test1);
        HolonElement ele = new HolonElement("Element", 1, 10);

        ArrayList<HolonElement> arr = new ArrayList<>();

        assertTrue("Should be Empty", test1.getElements().isEmpty());
        test1.setElements(arr);
        assertTrue("Should be Empty", test1.getElements().isEmpty());
        arr.add(ele);
        arr.add(ele);
        assertTrue("Should be Empty", test2.getElements().isEmpty());
        assertTrue("Current Energy not corrent", test2.getCurrentEnergy() == 0);
        test2.setElements(arr);
        assertTrue("Should not be Empty", !test2.getElements().isEmpty());
        assertTrue("Current Energy not corrent", test2.getCurrentEnergy() == 20);
        assertTrue("Current Energy not corrent", test2.getCurrentEnergyAtTimeStep(20) == 20);
        String str = test2.toStringElements();
        assertTrue("String not corrent", str.equals("Element, Element"));
        test2.deleteElement(0);
        test2.addElements(ele);
        assertTrue("Current Energy not corrent", test2.getCurrentEnergy() == 20);

        assertTrue("Should be Empty", test3.getElements().isEmpty());
        test3.setElements(test2.copyElements(test2.getElements()));
        assertTrue("Should be Empty", !test3.getElements().isEmpty());
        assertTrue("Should be state 0", test3.getState() == 0);
        test3.setState(1);
        assertTrue("Should be state 1", test3.getState() == 1);
        test3.setState(2);
        assertTrue("Should be state 2", test3.getState() == 2);
        test3.setState(3);
        assertTrue("Should be state 3", test3.getState() == 3);
        test3.setState(4);
        assertTrue("Should be state 4", test3.getState() == 4);
        assertTrue("Element not Found", test3.searchElement("Element") != null);
        test1.setElements(new ArrayList<HolonElement>());
        assertTrue("Not Empty", !test1.checkIfPartiallySupplied(1));
        test3.addElements(new HolonElement("Element2", 1, -10));
        assertTrue("Not Partially Supplied", test3.checkIfPartiallySupplied(1));
        test3.addElements(new HolonElement("Element2", 2, -10));
        assertTrue("Not Partially Supplied", test3.checkIfPartiallySupplied(1));
        test1.addElements(new HolonElement("Element2", 2, -10));
        assertTrue("minSupply < 0", !test1.checkIfPartiallySupplied(1));
        Color color = test3.getColor();
        test3.setColor(new Color(0, 255, 255));
        assertTrue(color.getBlue() != test3.getColor().getBlue());

        test3.addElements(new HolonElement("Element3", 3, 50));
        test3.setState();
        assertTrue("Should be state 3", test3.getState() == 3);
```

```
        test1.setState();
        assertTrue("Should be state 0", test1.getState() == 0);
        test2.setState();

        test2.setBorderColor(color);
        assertTrue("Color not Same", color == test2.getBorderColor());
    }

    /**
     * Test for HolonSwitch.
     */
    @Test
    public void testHolonSwitch() {
        HolonSwitch test1 = new HolonSwitch("Test1");
        HolonSwitch test2 = new HolonSwitch(test1);

        assertTrue("Manuel Mode is on", !test2.getManualMode());
        test2.switchState();
        assertTrue(test2.getState());
        assertTrue(test2.getState(1));
        test2.setManualMode(true);
        test2.switchState();
        test2.switchState();
        assertTrue(test2.getState());
        assertTrue(test2.getState(1));
        assertTrue("Manuel Mode is off", test2.getManualMode());
        assertTrue("ManuelActive is off", test2.getActiveManual());
        test2.switchState();
        assertTrue("ManuelActive is on", !test2.getActiveManual());
        test2.switchState();
        assertTrue("ManuelActive is off", test2.getActiveManual());
        assertTrue(test1.getGraphPoints() != test2.getGraphPoints());
        test2.setGraphPoints(test1.getGraphPoints());
        assertTrue(test1.getGraphPoints() == test2.getGraphPoints());
    }

    /**
     * Test  for CpsEdge.
     */
    @Test
    public void testCpsEdge() {
        CpsNode node1 = new CpsNode("Node1");
        CpsNode node2 = new CpsNode("Node2");
        CpsNode node3 = new CpsNode("Node3");
        CpsEdge edge1 = new CpsEdge(node1, node2, 100);
        CpsEdge edge2 = new CpsEdge(node2, node3);

        assertTrue("Flow not 0", edge1.getFlow() == 0);
        edge1.setFlow(50);
        assertTrue("Capacity not right", edge1.getCapacity() == 100);
        edge2.setFlow(50);
        edge1.setCapacity(200);
        assertTrue("Flow was not changed", edge1.getFlow() == 50);
        assertTrue("Capacity not right", edge1.getCapacity() == 200);
        assertTrue("line broken", edge2.getState());
        edge2.calculateState(false);
        assertTrue("line broken", edge2.getState());
        edge2.setFlow(200);
        edge2.calculateState(false);
        assertTrue("line not broken", !edge2.getState());
        edge1.setCapacity(-1);
        edge1.calculateState(false);
        edge1.setCapacity(500);
        edge1.calculateState(true);
        node1 = (CpsNode) edge1.getB();
        node2 = (CpsNode) edge2.getA();
        assertTrue("Not Same", node1 == node2);
        assertTrue("State not right", edge1.getState());
        edge1.setState(false);
        assertTrue("State not right", !edge1.getState());
        edge2.setTags(new ArrayList<>());
        edge1.setTags(new ArrayList<>());
        assertTrue("Tags not Empty", edge2.getTags().isEmpty());
        edge2.addTag(1);
        assertTrue("Tags not Empty", edge1.getTags().isEmpty());
        assertTrue("Tags Empty", !edge2.getTags().isEmpty());
        edge1.setTags(edge2.getTags());
        assertTrue("Tags Empty", !edge1.getTags().isEmpty());
    }
```

```
        /**
         * Test for HolonElement.
         */
        @Test
        public void testHolonElement() {
                HolonElement ele1 = new HolonElement("TV", 2, -20);
                HolonElement ele2 = new HolonElement("Fridge", 1, -50);
                HolonElement ele3 = new HolonElement(ele2);

                assertTrue("Array not empty", ele1.getEnergyAt()[0] == -20);
                assertTrue("Array not empty", ele1.getEnergyAt()[2] == -20);
                ele1.setEnergyAt(2, -10);
                assertTrue("Array not empty", ele1.getEnergyAt()[2] == -10);
                assertTrue("Name not correct", ele1.getEleName().equals("TV"));
                ele1.setEleName(ele2.getEleName());
                assertTrue("Name not correct", ele1.getEleName().equals("Fridge"));
                assertTrue("Amount not correct", ele2.getAmount() == 1);
                ele2.setAmount(5);
                assertTrue("Amount not correct", ele2.getAmount() == 5);
                assertTrue("Total Energy not Correct", ele2.getTotalEnergy() == ele2.getAmount() * ele2.getEnergy());
                assertTrue("Sign not correct", ele2.getSign() == '-');
                ele3.setSav("CVS");
                assertTrue("SAV not correct", ele3.getSav().equals("CVS"));
        }

        /**
         * Test for Position.
         */
        @Test
        public void testPosition() {
                Position pos1 = new Position(100, 200);
                Position pos2 = new Position();

                assertTrue("Wrong coordinates", pos1.x == 100 && pos1.y == 200);
                assertTrue("Are the Same", pos1.x != pos2.x);
                assertTrue("not (-1,-1)", pos2.x == -1 && pos2.y == -1);
        }

        /**
         * Test SubNet.
         */
        @Test
        public void testSubNet() {
                PraktikumHolonsAdapter adapter = new PraktikumHolonsAdapter();

                ArrayList<HolonObject> obj = new ArrayList<>();
                ArrayList<CpsEdge> edge = new ArrayList<>();
                ArrayList<HolonSwitch> sw = new ArrayList<>();

                assertTrue("Not Empty", obj.isEmpty());
                assertTrue("Not Empty", sw.isEmpty());
                assertTrue("Not Empty", edge.isEmpty());

                for (int i = 1; i < 10; i++) {
                        HolonObject o = new HolonObject(adapter.generate(i));
                        HolonSwitch s = new HolonSwitch(adapter.generate(i));
                        o = new HolonObject(o);
                        s = new HolonSwitch(s);
                        obj.add(o);
                        sw.add(s);
                        edge.add(new CpsEdge(o, s));
                }

                SubNet sub = new SubNet(obj, edge, sw);

                assertTrue("Empty", !sub.getObjects().isEmpty() && sub.getObjects().size() == 9);
                assertTrue("Empty", !sub.getSwitches().isEmpty() && sub.getSwitches().size() == 9);
                assertTrue("Empty", !sub.getEdges().isEmpty() && sub.getEdges().size() == 9);
                assertTrue("Wrong Obj", sub.getObjects().get(5).getObjName().equals("F"));

        }

}
```

A.4.3.2 Code Coverage

Zur Ermittlung des Code Coverage haben wir den Eclipse-Plugin EclEmma benutzt. Dieses Plugin ermittelt über Tests, wieviel Coverage unser Code erreicht. Da unser Projekt über das Model-View-Controler-Pattern aufgebaut ist, konnte keine vollständiger Coverage erreicht werden. Lediglich nur der Controller und das Model konnte geprüft werden. Das View konnte nicht über Code Coverage getestet werden, da dieses vom User-Input abhängig ist.

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
praktikum-holons	36,9 %	9.897	16.959	26.856
src	36,9 %	9.897	16.959	26.856
ui.view	5,3 %	810	14.339	15.149
ui.controller	58,5 %	2.763	1.963	4.726
SimulationManager.java	0,0 %	0	1.270	1.270
Control.java	0,0 %	0	531	531
ObjectController.java	83,5 %	208	41	249
LoadController.java	94,6 %	678	39	717
CanvasController.java	92,5 %	467	38	505
CategoryController.java	89,8 %	292	33	325
MultiPurposeController.java	98,1 %	204	4	208
StoreController.java	99,5 %	770	4	774
AutoSaveController.java	96,1 %	73	3	76
ConsoleController.java	100,0 %	28	0	28
GlobalController.java	100,0 %	43	0	43
tests	94,3 %	5.032	305	5.337
API	0,0 %	0	150	150
classes	90,2 %	1.097	119	1.216
ui.model	70,9 %	195	80	275
exceptions	0,0 %	0	3	3

Test Summary

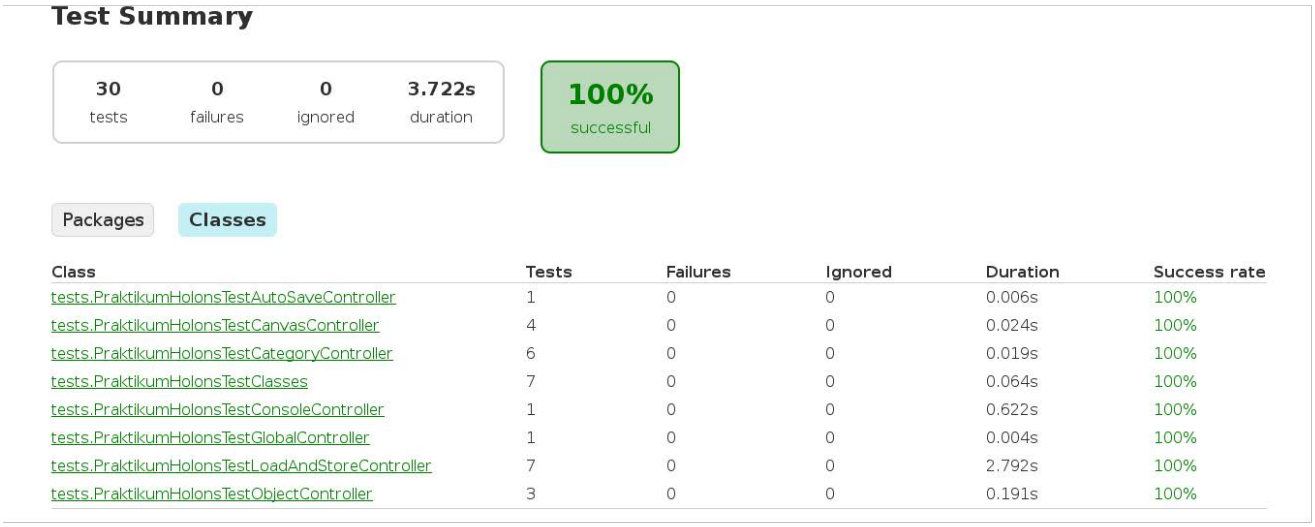
30	0	0	3.722s
tests	failures	ignored	duration

100% successful

Packages | Classes

Package	Tests	Failures	Ignored	Duration	Success rate
tests	30	0	0	3.722s	100%

Generated by Gradle 2.2.1 at 27.09.2016 18:58:34



Test Summary

30	0	0	3.722s
tests	failures	ignored	duration

100% successful

Packages **Classes**

Class	Tests	Failures	Ignored	Duration	Success rate
tests.PraktikumHolonsTestAutoSaveController	1	0	0	0.006s	100%
tests.PraktikumHolonsTestCanvasController	4	0	0	0.024s	100%
tests.PraktikumHolonsTestCategoryController	6	0	0	0.019s	100%
tests.PraktikumHolonsTestClasses	7	0	0	0.064s	100%
tests.PraktikumHolonsTestConsoleController	1	0	0	0.622s	100%
tests.PraktikumHolonsTestGlobalController	1	0	0	0.004s	100%
tests.PraktikumHolonsTestLoadAndStoreController	7	0	0	2.792s	100%
tests.PraktikumHolonsTestObjectController	3	0	0	0.191s	100%

A.4.3.3 Konsequenzen

Wie oben vorher beschrieben schlug immer der gesamte Build fehl, wenn ein automatisierter Test fehlschlug. Somit wurde der Fehler immer unverzüglich, ohne Erstellung weiterer User Stories, vom jeweiligen Entwickler behoben werden.

A.5 User Stories

ID	1
Name	CPS-Objekte per Drag and Drop platzieren
Beschreibung	Als Benutzer kann ich CPS-Objekte vom Seitenpanel (linke Spalte) per Drag and Drop auf der Modellierungsfläche (Canvas) platzieren.
Akzeptanzkrieterium	Das ausgewählte Objekt befindet sich an der gewünschten Stelle (x und y Koordinaten werden benutzt). Sollte es sich dabei um eine unerlaubte Stelle handeln (weil sich z.B dort bereits ein anderes befindet), wird das Objekt nicht platziert.
Geschätzter Aufwand (Story Points)	2
Entwickler	Kevin
Umgesetzt in Iteration	3
Tatsächlicher Aufwand (Std.)	7
Velocity (Std./Story Point)	7/2 = 3,5
Bemerkungen	

ID	2
Name	Zeichnen von Verbindungen
Beschreibung	Als Benutzer kann ich, nachdem ich das CPS-Objekt „Verbindung“ ausgewählt habe, durch Klicken auf den Start- und Endpunkt in der Modellierungsfläche (Canvas), eine Verbindung zwischen den zwei Punkten zeichnen.
Akzeptanzkrieterium	Die Verbindung wird zwischen den beiden Punkten korrekt dargestellt. Handelt es sich bei einem oder bei beiden Punkten um ein anderes CPS-Objekt, wird eine Verbindung zwischen ihnen gezeichnet.
Geschätzter Aufwand (Story Points)	5
Entwickler	Kevin
Umgesetzt in Iteration	4
Tatsächlicher Aufwand (Std.)	4
Velocity (Std./Story Point)	4/5 = 0,8

Bemerkungen	Es wurde mit mehr Komplikationen gerechnet

ID	3
Name	Maximale Kapazität von Verbindungen global editieren
Beschreibung	Als Benutzer kann ich im Menü die Option “Edit Connection Properties” auswählen und dann die Kapazität für alle existierenden und alle neu erstellten Verbindungen editieren
Akzeptanzkrieterium	Wird die Kapazität für alle existierenden Verbindungen geändert, wird dies auf der Modellierungsfläche (Canvas) ersichtlich und die Verbindungen verhalten sich auch dementsprechend. Wird die Kapazität für neu erstellte Verbindungen geändert, haben alle neu erstellten Verbindungen genau diese Kapazität
Geschätzter Aufwand (Story Points)	1
Entwickler	Dominik
Umgesetzt in Iteration	8
Tatsächlicher Aufwand (Std.)	3
Velocity (Std./Story Point)	3
Bemerkungen	

ID	4
Name	Erlangen von Information über einzelne Objekte
Beschreibung	Als Benutzer kann ich alle wichtigen Informationen über ein CPS-Objekt auf der Modellierungsfläche (Canvas) per Mausklick auf das Objekt in einem Seitenpanel anzeigen lassen.
Akzeptanzkrieterium	Die Informationen werden korrekt angezeigt. Das heißt, die Informationen entsprechen den echten Daten über das Objekt.
Geschätzter Aufwand (Story Points)	2
Entwickler	Edgardo, Jessey

Umgesetzt in Iteration	6
Tatsächlicher Aufwand (Std.)	6
Velocity (Std./Story Point)	3
Bemerkungen	Die gezeigte Information wird mehrmals verändert, sowie verschiedene Informationen für Single- und Multi-Selection.

ID	5
Name	Producer/Consumer hinzufügen/löschen
Beschreibung	Als Benutzer kann ich auf dem Seitenpanel in dem mir die Informationen über ein Objekt angezeigt werden, „Producer“ bzw. „Consumer“ mit gewünschtem Namen, Anzahl und Werten zu einem CPS-Gebäude hinzufügen.
Akzeptanzkrieterium	Die Änderungen werden gespeichert und bei der nächsten Informationsabfrage wieder angezeigt.
Geschätzter Aufwand (Story Points)	3
Entwickler	Dominik, Edgardo, Jessey
Umgesetzt in Iteration	4
Tatsächlicher Aufwand (Std.)	3 ; 12,5
Velocity (Std./Story Point)	5,17
Bemerkungen	

ID	6
Name	Ansicht Mode (Modellierung und Simulation)
Beschreibung	Als Benutzer kann ich durch das Klicken auf die entsprechenden Buttons zwischen den Ansichten „Modellierung“ und „Simulation“ wechseln
Akzeptanzkrieterium	Nach dem Klicken auf „Simulation“ wird die Simulationsansicht angezeigt. Nach dem Klicken auf „Modellierung“ wird die Modellierungsansicht angezeigt.
Geschätzter Aufwand (Story Points)	2

Entwickler	Kevin
Umgesetzt in Iteration	10
Tatsächlicher Aufwand (Std.)	2
Velocity (Std./Story Point)	1
Bemerkungen	Weniger Komplikationen als Erwartet

ID	7
Name	Speichern als Datei/Laden einer Datei
Beschreibung	Als Benutzer kann ich den aktuellen Zustand in Form einer Datei speichern. Des Weiteren kann ich auch erstellte Dateien Laden.
Akzeptanzkrieterium	Der Zustand wird korrekt gespeichert und beim Laden auch wieder genauso dargestellt bzw. in das System übernommen.
Geschätzter Aufwand (Story Points)	4
Entwickler	Julian
Umgesetzt in Iteration	6
Tatsächlicher Aufwand (Std.)	20,5
Velocity (Std./Story Point)	5,13
Bemerkungen	

ID	8
Name	Kategorie hinzufügen/löschen
Beschreibung	Als Benutzer kann ich durch das Klicken auf den Button “+ Category“ bzw. “- Category” eine Neue Kategorie mit gewünschtem Namen hinzufügen bzw. löschen
Akzeptanzkrieterium	Die Kategorie wird hinzugefügt und unter den anderen angezeigt. Sollte der Name bereits vergeben sein, wird der Benutzer aufgefordert einen anderen zu vergeben.

Geschätzter Aufwand (Story Points)	3
Entwickler	Dominik
Umgesetzt in Iteration	2
Tatsächlicher Aufwand (Std.)	7,5 Std.
Velocity (Std./Story Point)	7,5/3 = 2,5
Bemerkungen	

ID	9
Name	Objekte zu Kategorien hinzufügen/ löschen
Beschreibung	Als Benutzer kann ich innerhalb einer Kategorie durch das Klicken auf den Button “+ <Kategoriename>“ bzw. “- <Kategoriename>”ein Objekt mit gewünschtem Namen und Default Werten zu einer Kategorie hinzufügen bzw. löschen
Akzeptanzkrieterium	Das Neue Objekt wird mit den gegebenen Werten zu der Kategorie hinzugefügt und angezeigt
Geschätzter Aufwand (Story Points)	4
Entwickler	Jessey, Dominik
Umgesetzt in Iteration	4 bzw. 2
Tatsächlicher Aufwand (Std.)	9,5
Velocity (Std./Story Point)	9,5/4 = 2,3 75
Bemerkungen	

ID	10
Name	Editieren von Eigenschaften von canvas objekten.
Beschreibung	Als Benutzer kann ich Kategorien, Objekte in Kategorien, Objekte auf der Modellierungsfläche, sowie die einzelnen „Producer“ und „Consumer“ innerhalb der Objekte sowohl löschen, als auch jede Eigenschaft editieren.

Akzeptanzkrieterium	Jegliche Änderung am aktuellen Zustand wird übernommen und entsprechend angezeigt.
Geschätzter Aufwand (Story Points)	4
Entwickler	Edgerado
Umgesetzt in Iteration	6
Tatsächlicher Aufwand (Std.)	Edgardo: 9 property table 9 element table Kevin: 2 element table Jessey: 30 min Verbindungen eigenschaften
Velocity (Std./Story Point)	
Bemerkungen	

ID	11
Name	Energie Verhalten von Elementen und Status von Switches in bestimmten Zeitintervall als Graph zeichnen
Beschreibung	Durch das Klicken auf ein HolonElements auf dem Seitenpanel bzw. Switch auf den Canvas, kann man auf der Zeichenfläche einen Graph zeichnen, der den Verbrauch bzw. Produktion (in %) für HolonElements abhängig vom Maximal Verbrauch bzw. Produktion und den Status (On/Off) für Switches zum bestimmten Zeitpunkt angibt
Akzeptanzkrieterium	Der Graph wird gespeichert und beim nächsten Klicken auf das Element wieder genau so angezeigt
Geschätzter Aufwand (Story Points)	6
Entwickler	Kevin
Umgesetzt in Iteration	6 bzw. 7
Tatsächlicher Aufwand (Std.)	15
Velocity (Std./Story Point)	15/6 = 2,5
Bemerkungen	

ID	12
Name	Holon Elemente ein/aus-schalten

Beschreibung	Durch das Klicken auf die Checkbox eines Elements, kann man es an bzw. ausschalten
Akzeptanzkrieterium	Ein ausgeschaltetes Element wird bei der Gesamtenergieberechnung nicht berücksichtigt. Ansonsten ist das Gegenteil der Fall
Geschätzter Aufwand (Story Points)	2
Entwickler	Jessey
Umgesetzt in Iteration	6
Tatsächlicher Aufwand (Std.)	3
Velocity (Std./Story Point)	3/2 = 1,5
Bemerkungen	

ID	13
Name	Editieren von Objekten im Kategorien
Beschreibung	Als Benutzer kann ich Objekte in Kategorien editieren.
Akzeptanzkrieterium	Jegliche Änderung am aktuellen Zustand wird übernommen und entsprechend angezeigt.
Geschätzter Aufwand (Story Points)	2
Entwickler	Edgardo
Umgesetzt in Iteration	8
Tatsächlicher Aufwand (Std.)	10
Velocity (Std./Story Point)	10/2 = 5
Bemerkungen	

ID	14
Name	Markieren von mehreren Objekten auf der Canvas
Beschreibung	Als Benutzer kann ich Objekte auf der Canvas markieren und als gruppe verschieben
Akzeptanzkrieterium	Alle ausgewählten Objekte sind makiert
Geschätzter Aufwand (Story Points)	4
Entwickler	Kevin, Edgardo
Umgesetzt in Iteration	8
Tatsächlicher Aufwand (Std.)	Kevin: 5 + Edgardo:7,5 = 15,5
Velocity (Std./Story Point)	15,/4 = 3.875

Bemerkungen	

ID	15
Name	Autosaves erzeugen und laden
Beschreibung	Als Benutzer kann ich durch Buttons Änderungen auf der Canvas rückgängig machen bzw. wiederherstellen
Akzeptanzkrieterium	Bei Undo: Canvas befindet sich in dem Zustand vor exakt einer Änderung Bei Redo: Canvas befindet sich in exakt dem Zustand vor dem rückgängig machen
Geschätzter Aufwand (Story Points)	3
Entwickler	Jessey
Umgesetzt in Iteration	8
Tatsächlicher Aufwand (Std.)	6
Velocity (Std./Story Point)	6/3 = 2
Bemerkungen	

ID	16
Name	Markieren von mehreren HolonElementen auf der Tabelle
Beschreibung	Als Benutzer kann ich mehrere HolonElement markieren und als Gruppe löschen und Graph (Produktion und Verbrauch) ändern
Akzeptanzkrieterium	Alle augewählten HolonElement sind markiert und werden richtig geändert
Geschätzter Aufwand (Story Points)	4
Entwickler	Edgardo
Umgesetzt in Iteration	8
Tatsächlicher Aufwand (Std.)	16
Velocity (Std./Story Point)	16/4 = 4
Bemerkungen	

ID	17
Name	Cut/Copy/Paste von CpsObjekten auf der Canvas

Beschreibung	Object auf der Canvas auschneiden, kopieren und einfügen
Akzeptanzkrieterium	Die eingefügten Objekte sind exakte Kopien von ausgeschnittenen/kopierten Elementen (abgesehen von der ID)
Geschätzter Aufwand (Story Points)	3
Entwickler	Kevin
Umgesetzt in Iteration	10
Tatsächlicher Aufwand (Std.)	9
Velocity (Std./Story Point)	3
Bemerkungen	

ID	18
Name	Suchen von Objekten und Ersetzen der Namen
Beschreibung	Als Benutzer hat man die Möglichkeit Objekte auf dem Canvas über den Namen zu Suchen. Dabei können alle Gleichzeitig Gefunden Werden oder alle einzeln
Akzeptanzkrieterium	Single Suche: Forward und Backward jedes einzelne Object wird gefunden und markeirt Suche nach Alle: Alle Objete mit dem Namen werden Markiert Replace All: Alle Objekte bekommen den neuen Namen und werden Markiert Single replace: markierte Objekte bekommen neuen Namen
Geschätzter Aufwand (Story Points)	3
Entwickler	Julian
Umgesetzt in Iteration	10
Tatsächlicher Aufwand (Std.)	10
Velocity (Std./Story Point)	10/3 = 3,33
Bemerkungen	

ID	19
Name	Start von Simulation (Zeitpunkt, Geschwindigkeit und Algorithm)

Beschreibung	Durch das Klicken auf den Simulationsbutton wird der Simulationsmodus aktiviert, aufgrund dessen sich das Verhalten des Netzwerks dementsprechend verändert. Der User kann auch die Geschwindigkeit der Simulation ändern
Akzeptanzkrieterium	Der Button Für Simulation bleibt ausgewählt und Verbindungen die kaputt gehen, bleiben kaputt.
Geschätzter Aufwand (Story Points)	4
Entwickler	Dominik
Umgesetzt in Iteration	10
Tatsächlicher Aufwand (Std.)	7
Velocity (Std./Story Point)	1,75
Bemerkungen	

ID	20
Name	View ändern (Größe von Objekten)
Beschreibung	Die Größe der Objekte auf der Canvas verändern
Akzeptanzkrieterium	Die Gößе der Objekte entspricht der vom User angegebene Größe
Geschätzter Aufwand (Story Points)	1
Entwickler	Kevin
Umgesetzt in Iteration	10
Tatsächlicher Aufwand (Std.)	1
Velocity (Std./Story Point)	1
Bemerkungen	

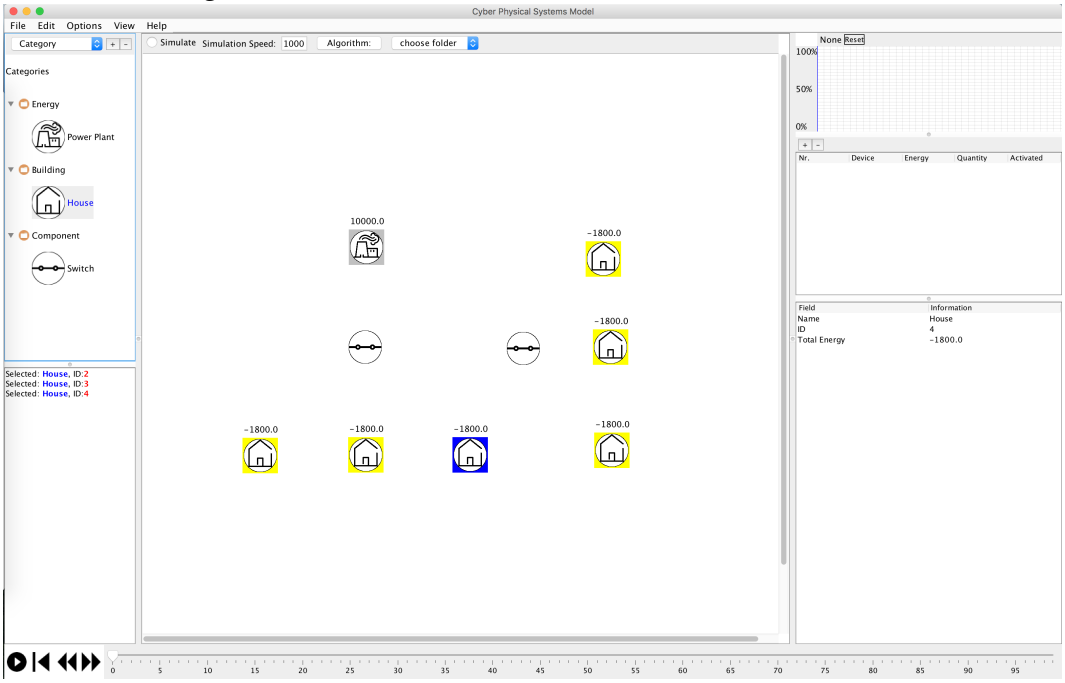
ID	21
Name	Edit-Mode von HolonElement
Beschreibung	Als Benutzer kann ich HolonElemente editieren (Name, Verbrauch oder Produktion, Status (On/Off) und Menge)
Akzeptanzkrieterium	HolonElement wird richtig aktualisiert, sowie Einfluss beim Total Energy (HolonObject) verändert.
Geschätzter Aufwand (Story Points)	2
Entwickler	Edgardo
Umgesetzt in Iteration	8

Tatsächlicher Aufwand (Std.)	7
Velocity (Std./Story Point)	7/2 = 3,5
Bemerkungen	

A.6 Short Story

Wir wollen durch ein Beispiel die verschiedenen Funktionalitäten unseres Programms zeigen. Wir werden ein fiktives Szenario erzeugen. Das Szenario wird aus drei kleinen Dörfer bestehen. Jedes Dorf wird Energie aus 2 verschiedene Kraftwerke bekommen und mittels Schaltern geregelt.

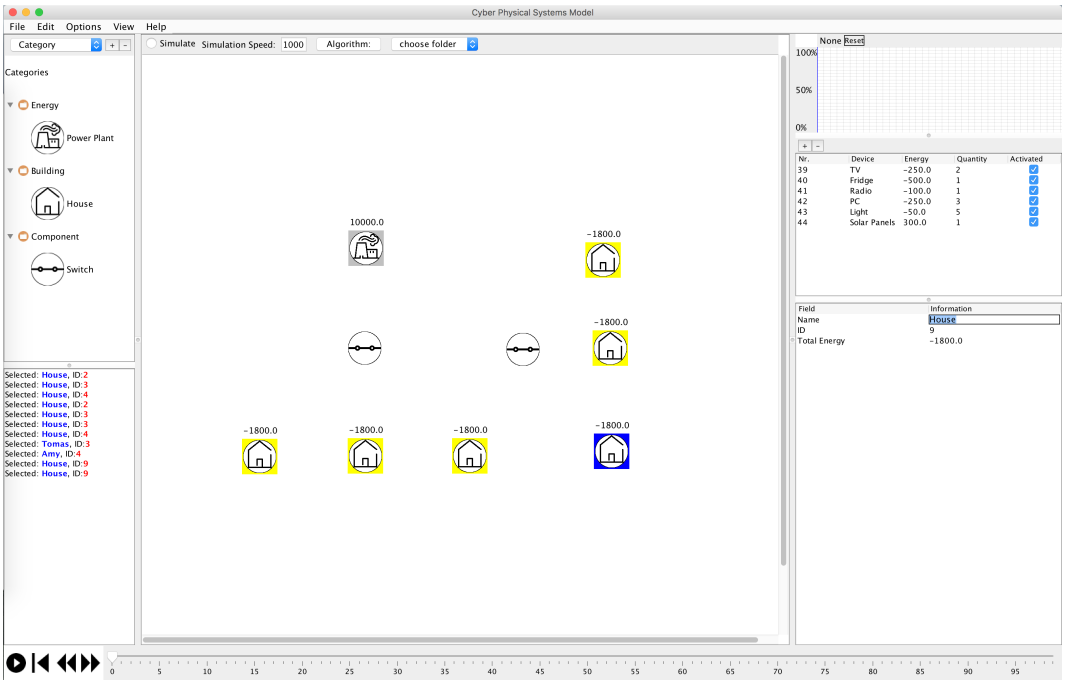
1. Schritt: (Erstellung von Standard Elementen)



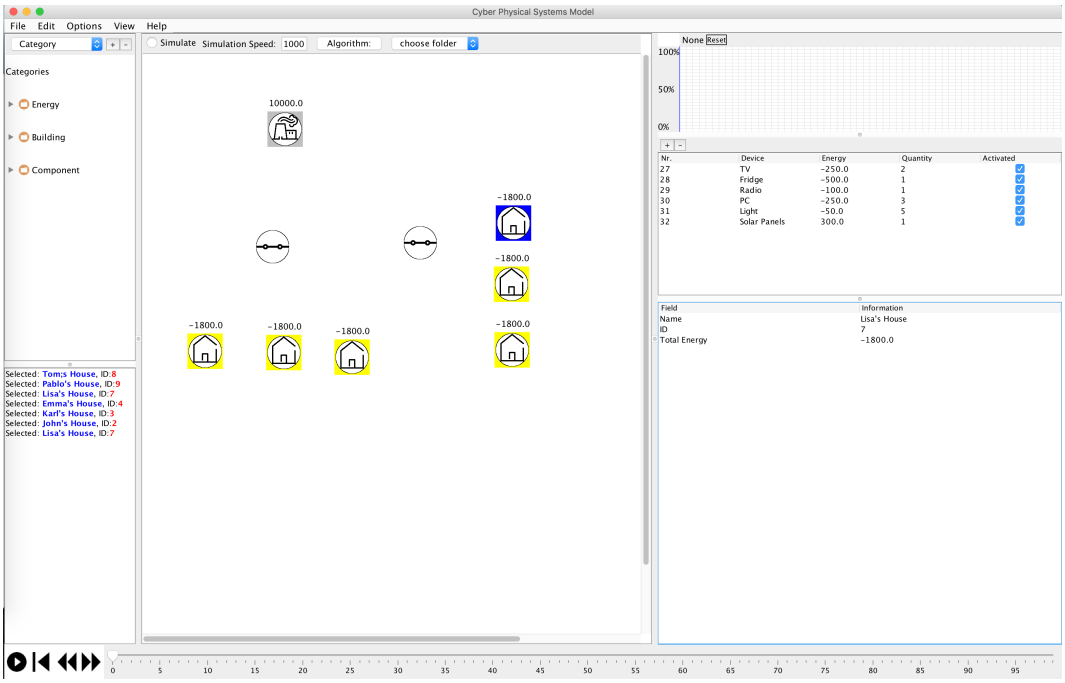
Screenshot 1

Am Anfang wird der Benutzer Standard-“CpsObject” und -“Categories” zur Verfügung haben. Diese sind die Folgenden: “Energy”-Kategorie enthält “Power Plant”(s) , “Building”-Kategorie enthält “House”(s) und “Component”-Kategorie enthält “Switch”(es).
Im unseren Beispiel werden werden sechs Häuser, zwei Schaltern und ein Kraftwerk modelliert, die in zwei “Dörferünterteilt werden. Jedes enthält 3 Häuser und ein Schalter, und beide “Dörfer” bekommen Strom von dem gleichen Kraftwerk (siehe Screenshot 1)

2.Schritt: (Editieren von Namen)



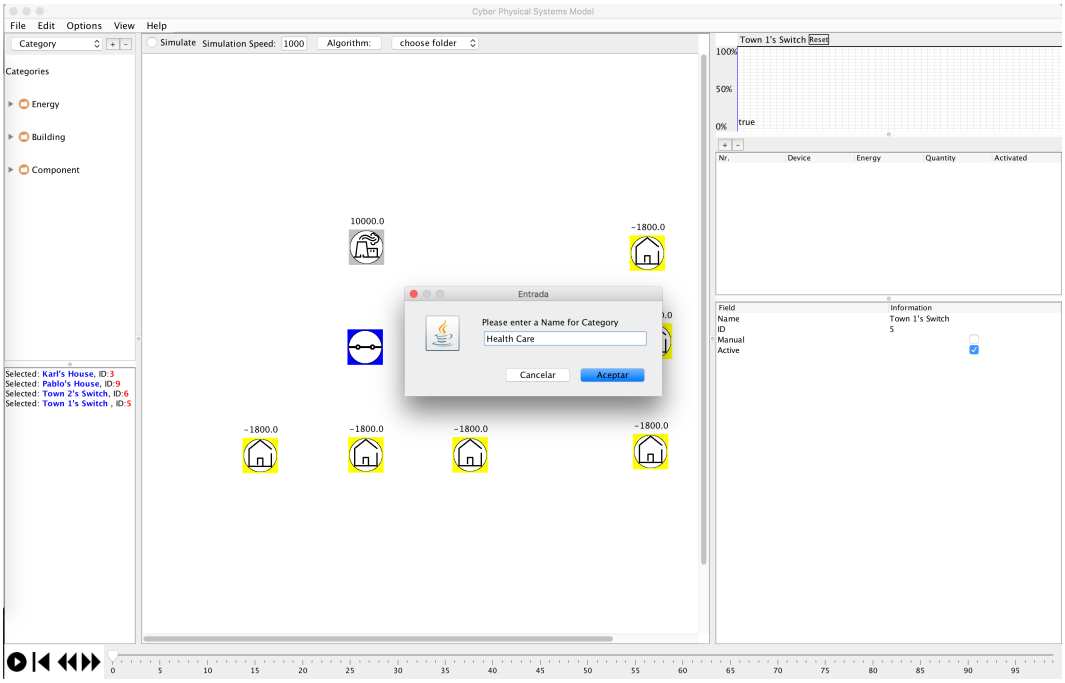
Screenshot 2



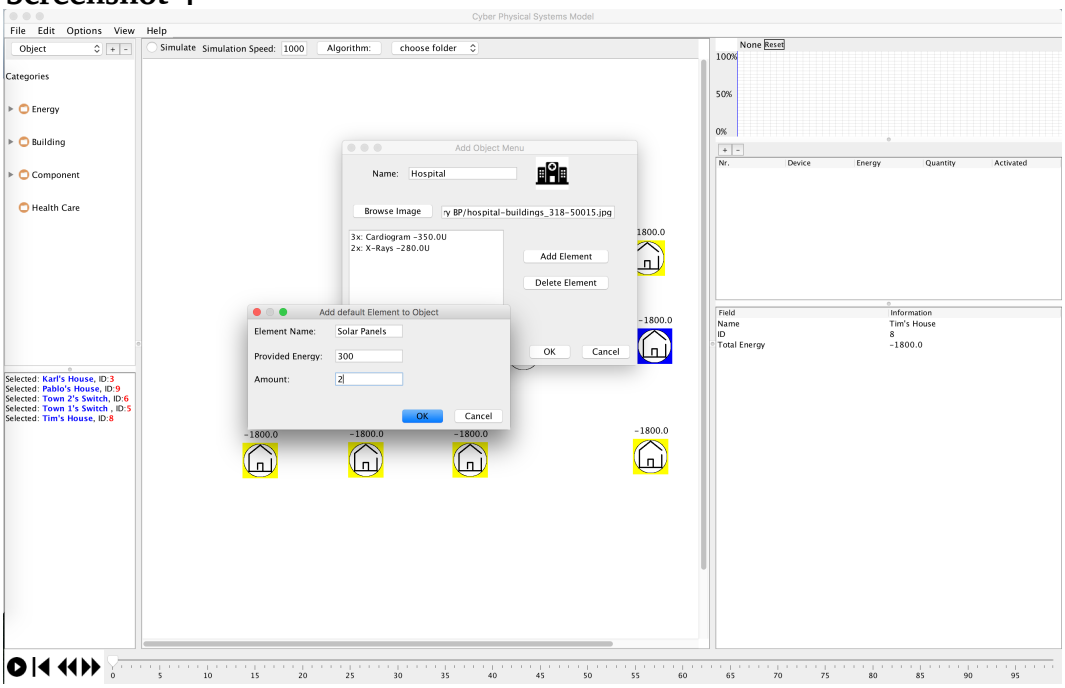
Screenshot 3

Man hat die Möglichkeit die Namen der verschiedenen “CpsObject”(s) zu verändern.
Zum Beispiel benennen wir das Haus mit ID 2 “John’s House”, das Haus mit ID 3 “Karl’s House”, das Haus mit ID 4 “Emma’s House”, das Haus mit ID 7 “Lisa’s House”, das Haus mit ID 8 “Tom’s House” und das Haus mit ID 9 “Pablo’s House”. Gleicherweise benennen wir die zwei Schalter auf “Town 1’s Switch” und “Town 2’s Switch” um. (siehe Screenshot 2).

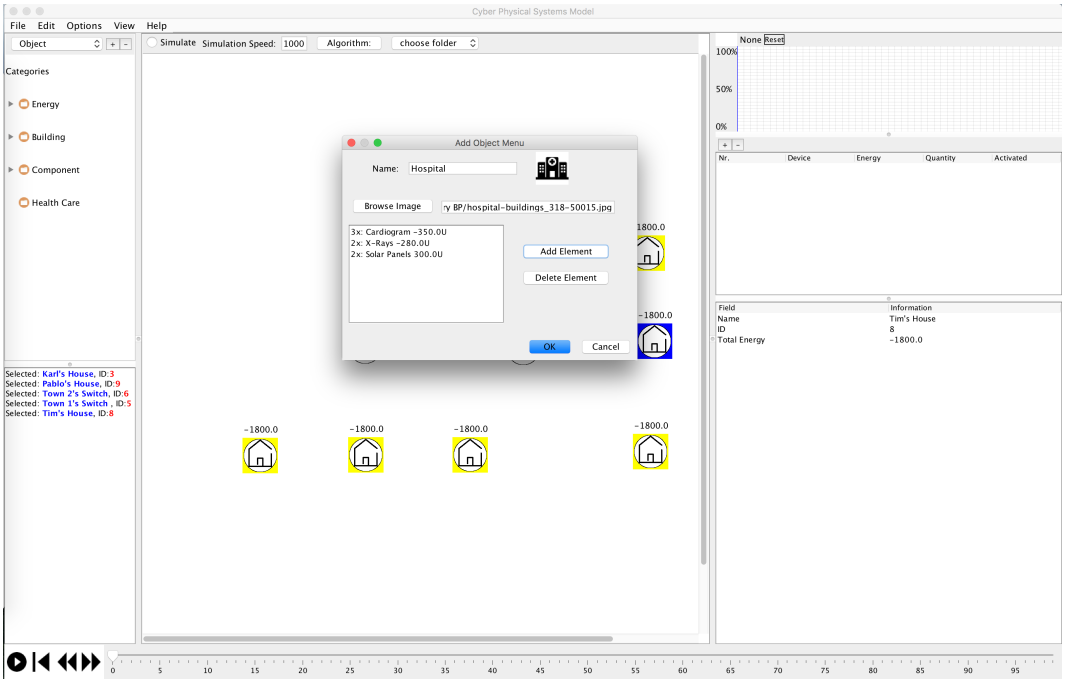
3. Schritt: (Neue “Categorie” und “HolonObject” erstellen)



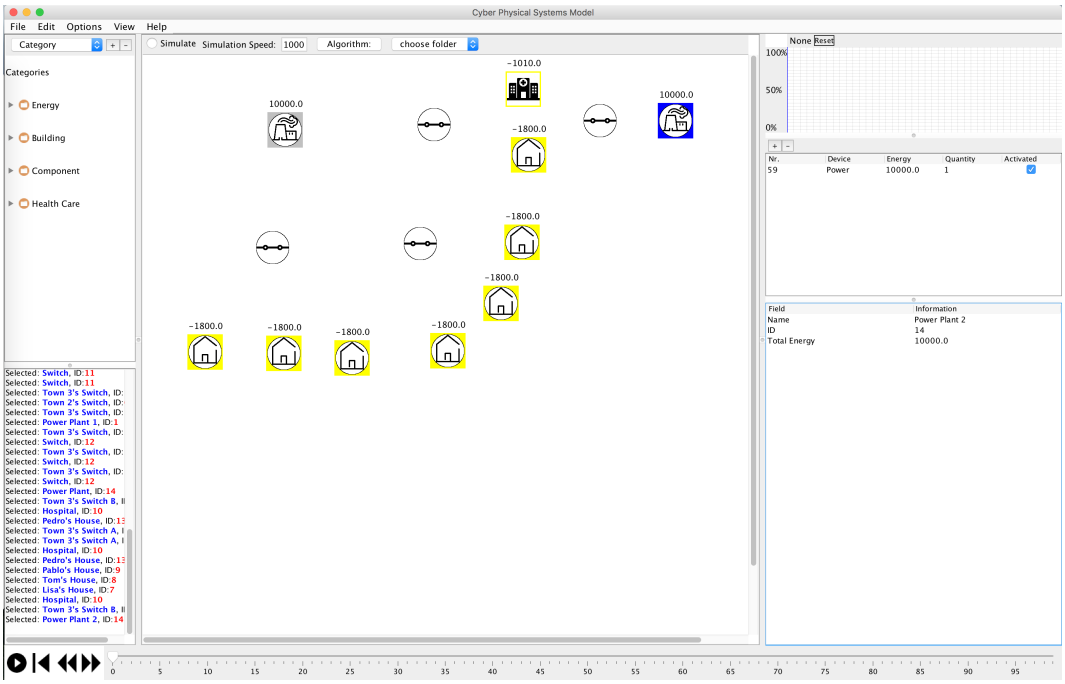
Screenshot 4



Screenshot 5



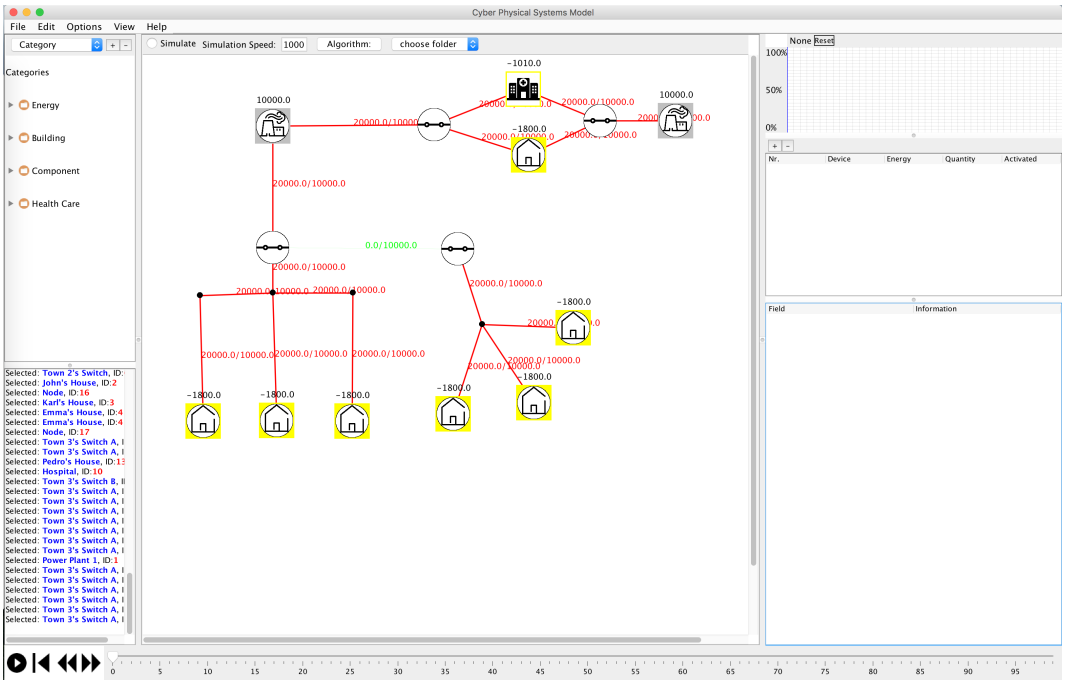
Screenshot 6



Screenshot 7

Man hat die Möglichkeit neue “Categorie” bzw. “HolonObject” zu erzeugen. Die neuen “HolonObject” haben die Möglichkeit vordefinierte “HolonElement” zu enthalten.
Beispielsweise definieren wir eine weitere “Category” “Health Care” (siehe Screenshot 4), welche das “HolonObject” “Hospital” hat (siehe Screenshot 5), dass die folgenden “HolonElement”(s) enthält: 3 “Cardiogram”(s) mit einem Verbrauch von jeweils350 Einheiten, 2 “X-Rays”(s) mit einem Verbrauch von jeweils 280 Einheiten und 2 “Solar Panel”(s) mit einer Produktion von jeweils 300 Einheiten (siehe Screenshot 6). In unserem Beispiel wird ein neues Dorf hinzugefügt, das eine Krankenhaus und ein weiteres Haus (“Pedro’s House”) enthält. Dieses Dorf wird gleichseitig von zwei Kraftwerken versorgt (“Power Plant 1” und “Power Plant 2”) (siehe Screenshot 7).

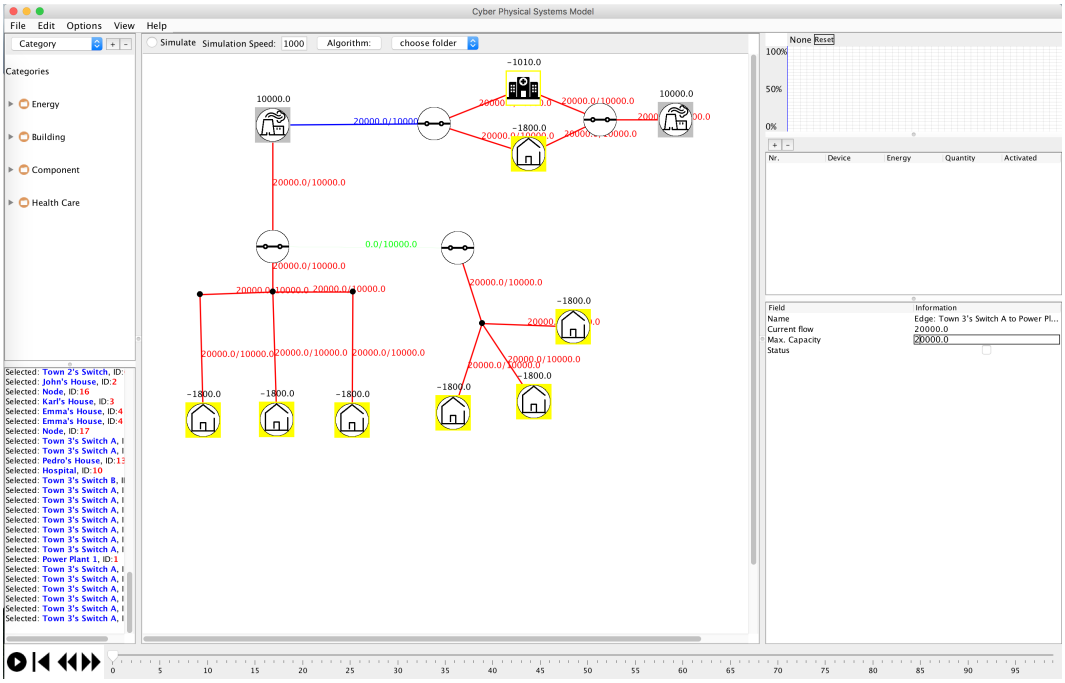
4. Schritt: (Verbindungen zwischen “CpsObject”(s))



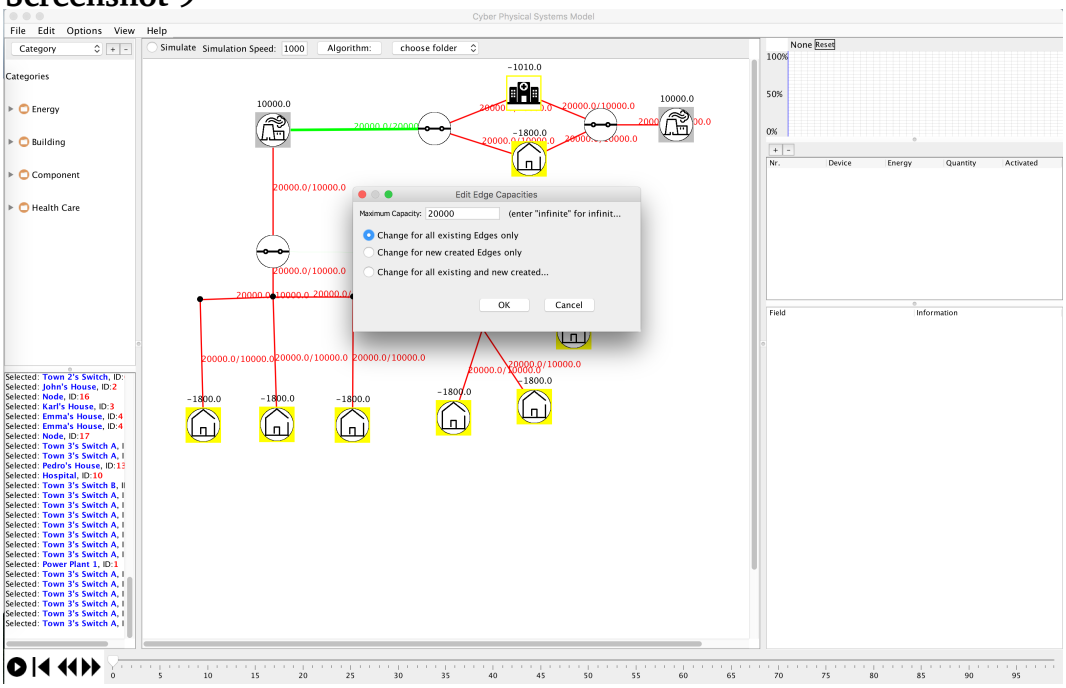
Screenshot 8

Als Leitungen hat der Benutzer Linien zur Verfügung. Man kann beliebige “CpsObject”(s) miteinander verknüpfen, sowie mit Knoten. Die Knoten werden automatisch erstellt, sobald eine Linie kein reelles “CpsObject” als Ziel erkennt.
In unseren Beispiel wir das “Power Plant 1” mit 2 Schaltern verknüpft (“Town 1’s Switch” und “Town 3’s Switch A”). “Town 1’s Switch” wird mit 3 Häusern (“John’s House”, “Karl’s House” und “Emma’s House”) und einem weiteren Schalter (“Town 2’s Switch”) verknüpft. “Town 2’s Switch” wird mit 3 weiteren Häusern (“Lisa’s House”, “Tom’s House” und “Pablo’s House”) verknüpft. Andererseits wird “Town 3’s Switch A” mit dem “Hospital” und “Pedro’s House” verknüpft. Beide werden gleichzeitig durch “Town 3’s Switch B” mit “Power Plant 2” verknüpft (siehe Screenshot 8).

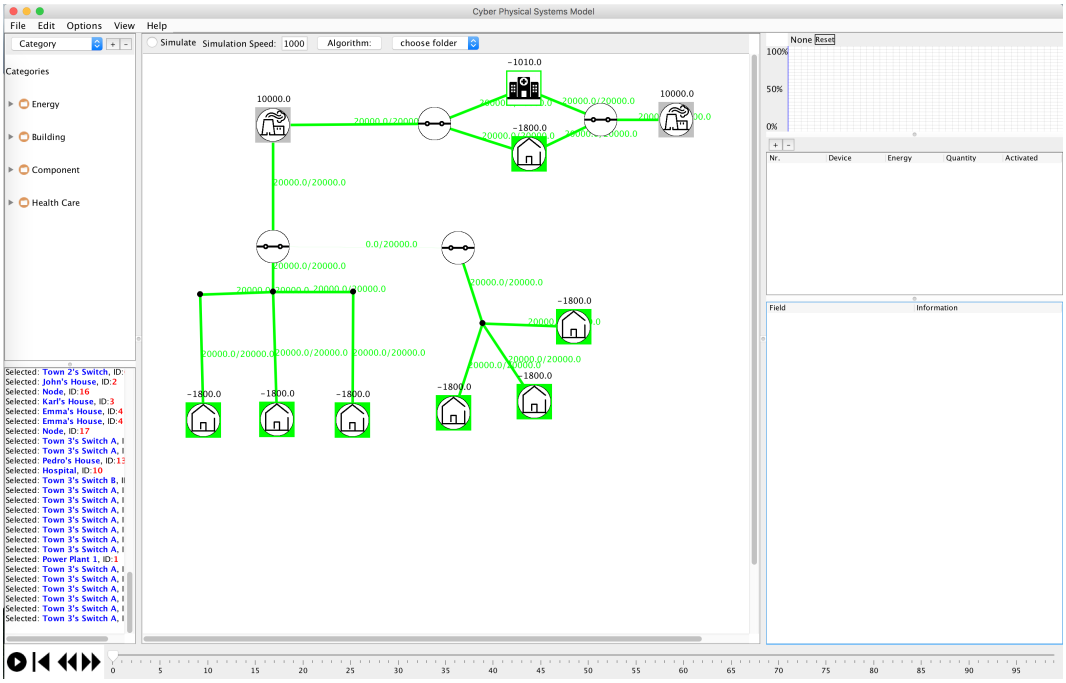
5. Schritt: (Editieren von maximalem Stromfluss bei “CpsEdges”)



Screenshot 9



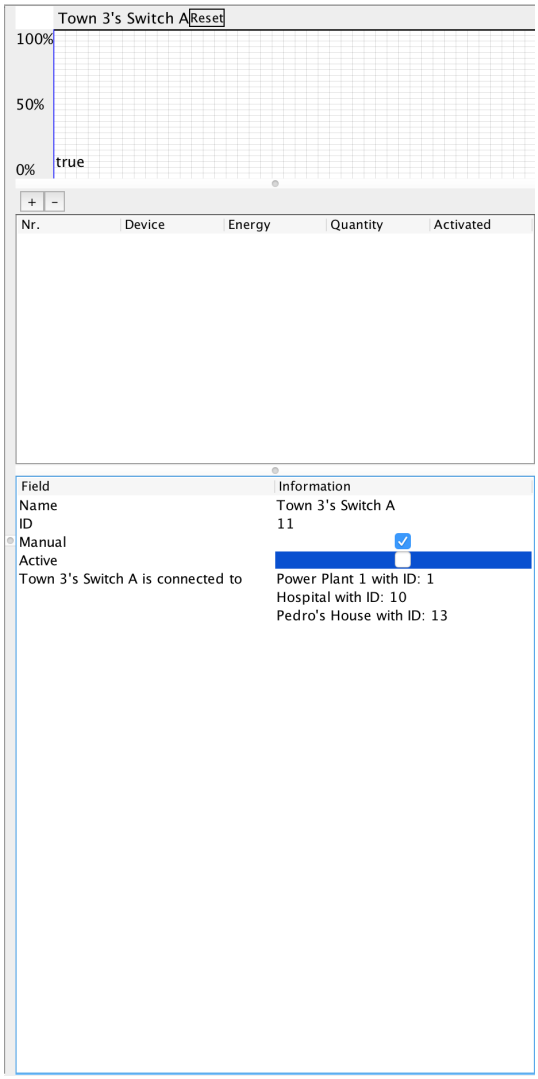
Screenshot 10



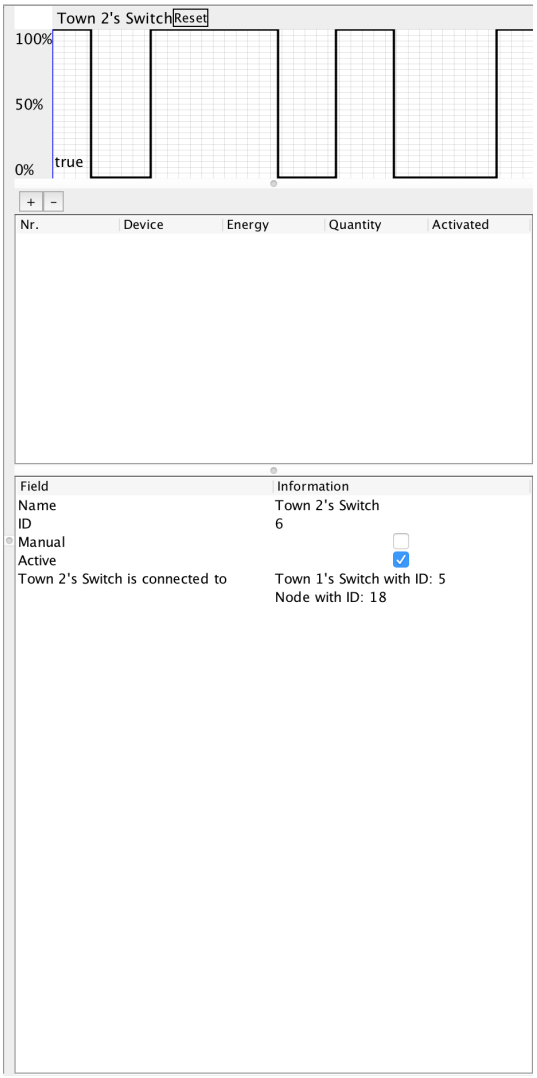
Screenshot 11

Im Schritt 4 kann man sehen, dass der Zustand der Verbindungen rot ist. Das heißt, dass der Stromfluss höher als die Kapazität der Verbindungen ist. Als Lösung hat man folgende Möglichkeiten: jeder Verbindung manuell editieren (siehe Screenshot 9) oder alle Verbindungen mit dem gleichen Wert global editieren (siehe Screenshot 10).
Wir werden alle Verbindungen auf 20000 “Units” global setzen, sodass der Stromfluss (kleiner oder) gleich der Kapazität ist (siehe Screenshot 11).

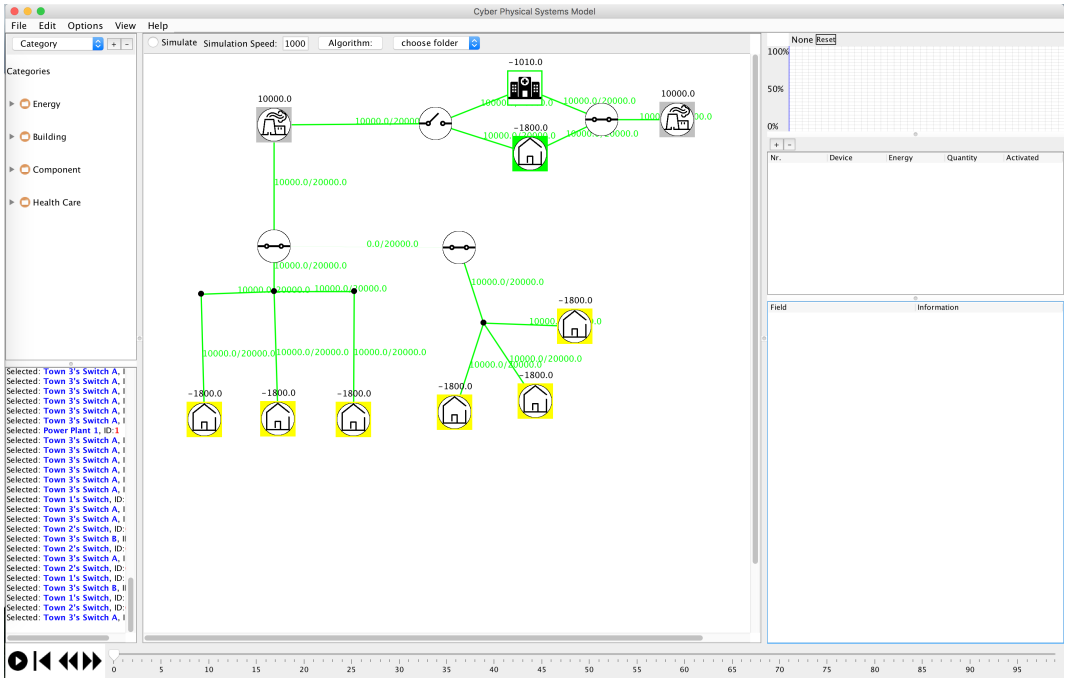
6. Schritt: (Editieren vom Modus des “HolonSwitch”(es))



Screenshot 12



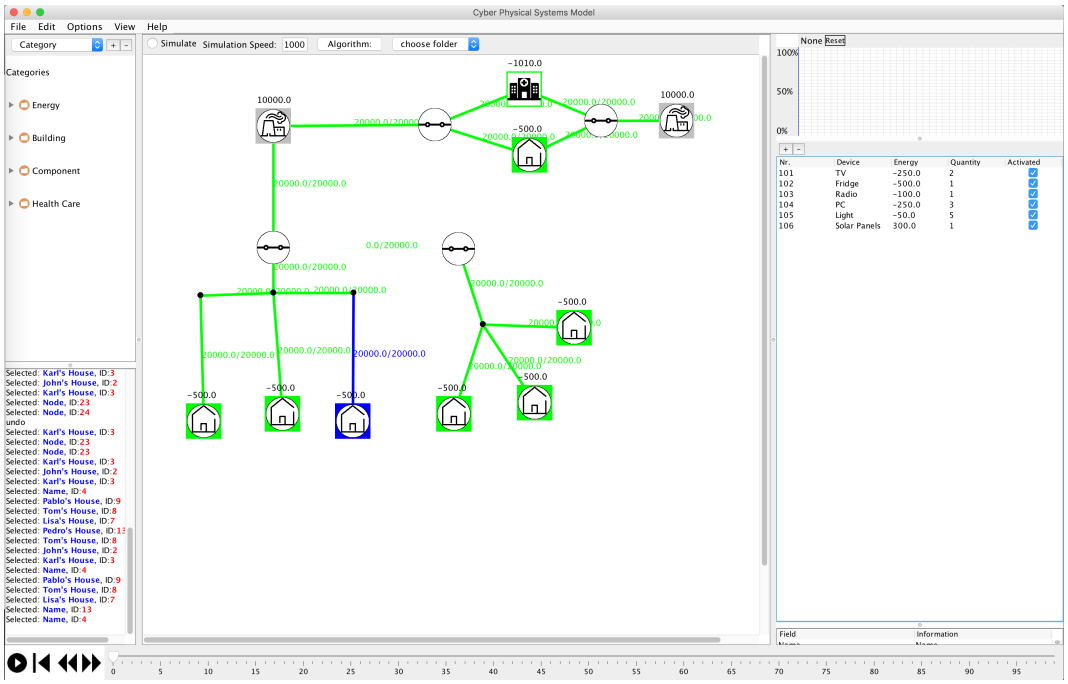
Screenshot 13



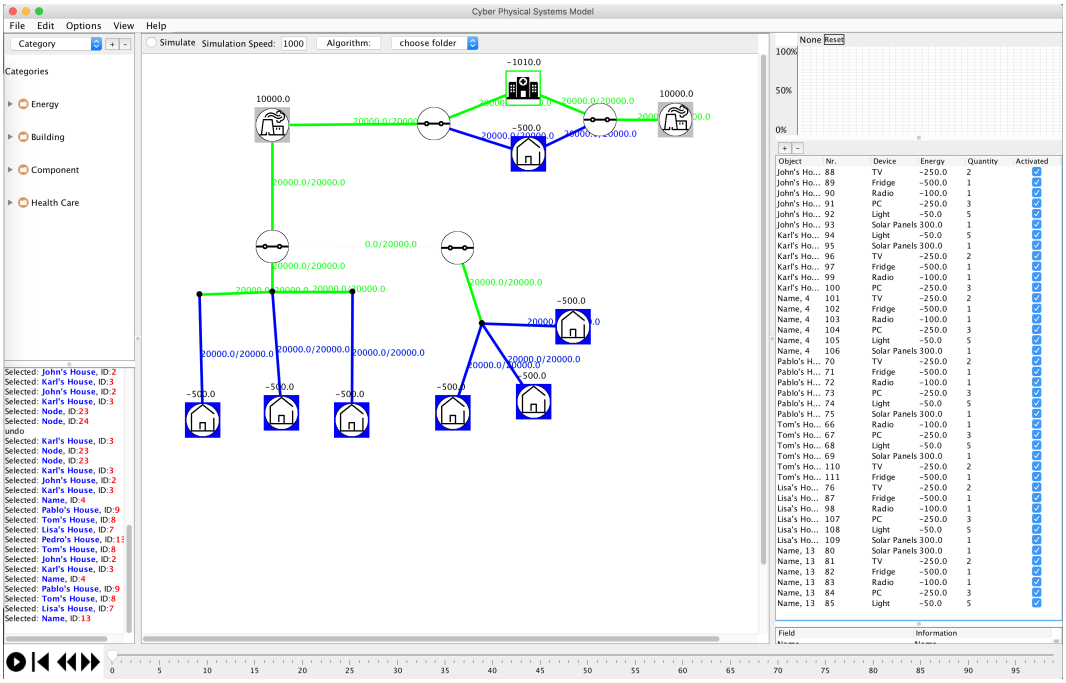
Screenshot 14

Das Problem vom maximalem Stromfluss und Kapazität, die man im Schritt 5 gehabt hat, könnte man in einer anderen Weise lösen. Der Benutzer hat die Möglichkeit die Schalter entweder manuell oder automatisch zu manipulieren. Den Modus kann man unter dem Feld “Mode” ändern (siehe Screenshot 12 unten). Manuell heißt, Doppel auf den Schalter zu klicken oder die “Checkbox” unter dem Feld “Active” zu markieren um den Zustand zu ändern. Automatisch bedeutet, dass das Verhalten des Schalters mittels des oberen Graph beschrieben wird (siehe Screenshot 13 oben).
Im unserem Beispiel werden wir “Town 1’s Switch” ,“Town 2’s Switch” und “Town 3’s Switch B” als automatisch und “Town 3’s Switch A” als manuell mit den Anfangswert “open” modelliert (siehe Screenshot 14).

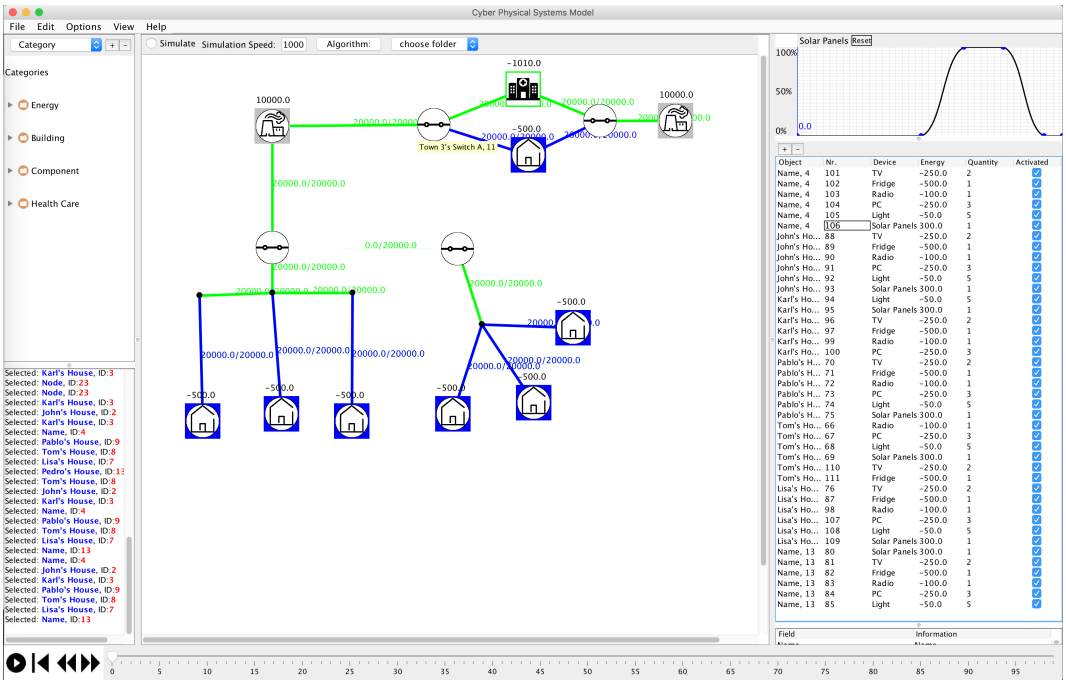
7. Schritt: (Editieren von “HolonElement”(s) mittels “Single-” oder “Multi-Selection”)



Screenshot 15



Screenshot 16



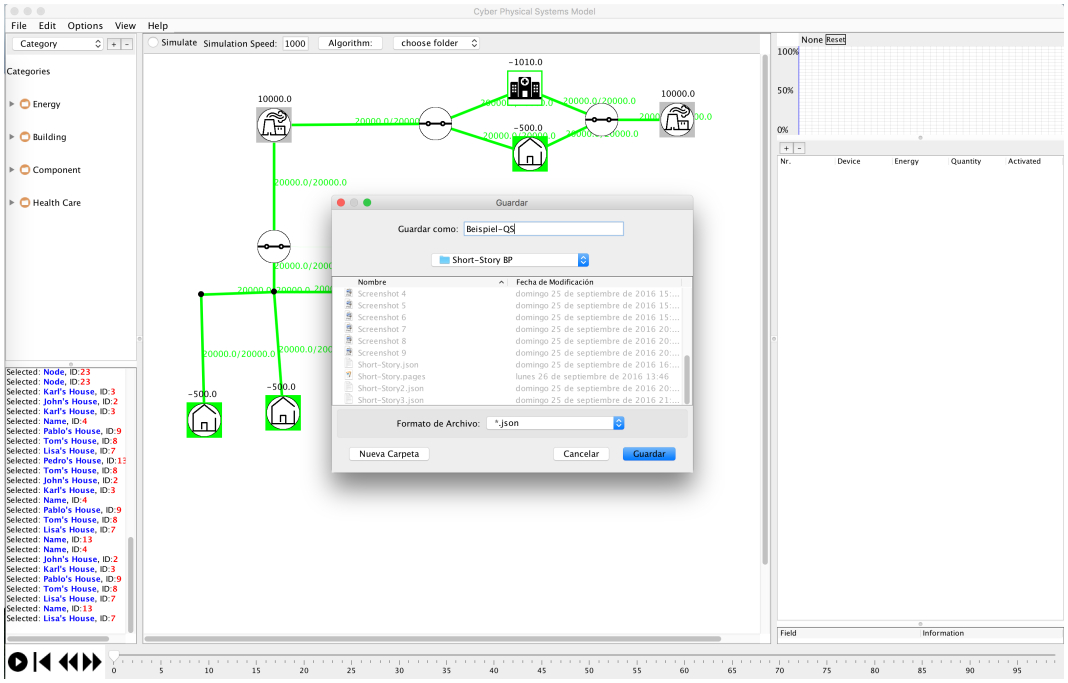
Screenshot 17

Der Benutzer hat ebenso ein Tool, um die “HolonElement”(s) editieren zu können. Felder, die editiertbar sind, sind der Name, der Strom (positiv entspricht Verbrauch und negativ entspricht Produktion), die Menge, der Zustand und den Stromfluss jedes “HolonElement”(s) über einen Zeitraum. Edit-Tool unterteilt sich in 4 verschiedene Zustände:

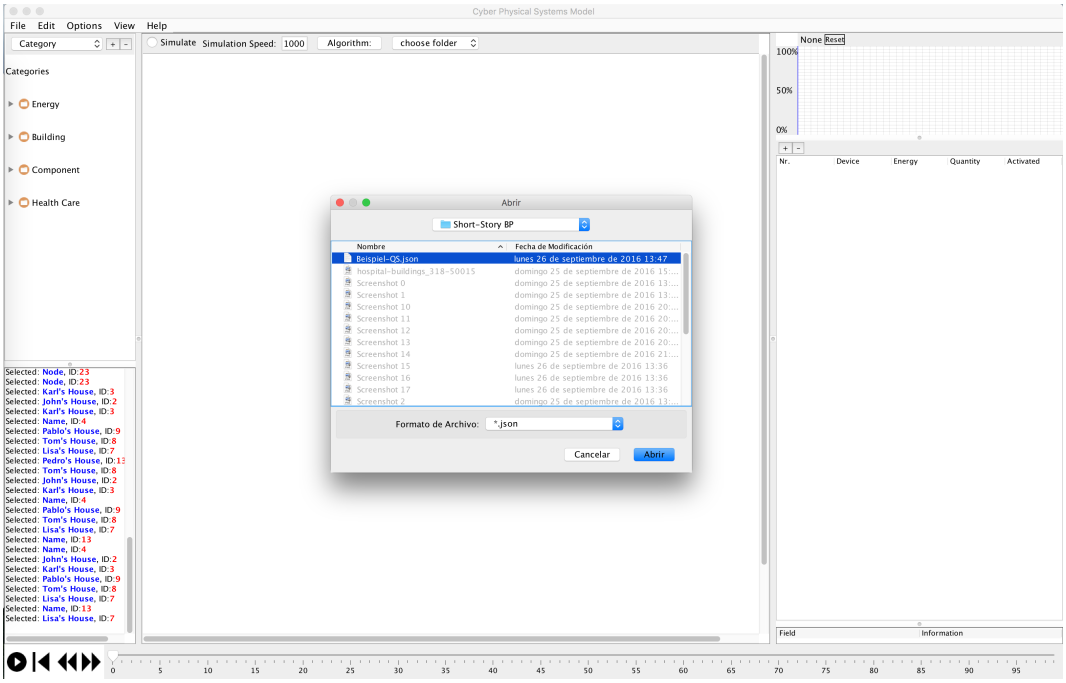
1. Single-Selektion von “HolonObject” - Single-Selektion von “HolonElement”
2. Single-Selektion von “HolonObject - Multi-Selektion von “HolonElement”
3. Multi-Selektion von “HolonObject” - Single-Selektion von “HolonElement”
4. Multi-Selektion von “HolonObject” - Multi-Selektion von “HolonElement”

Die Tabelle in der Mitte der rechten Spalte wird entsprechend angepasst. Die Tabelle wird folgende Felder besitzen: “Nr.” (ID), “Device” (Name), “Energy” (positiver oder negativer Strom), “Quantity” (Menge) und “Activated” (Zustand) beim Single-Selektion von “HolonObject” (siehe Screenshot 15) und beim Multi-Selektion von “HolonObject”: “Object” (“HolonObject”, zu dem das “HolonElement” gehört), “Nr.” (ID), “Device” (Name), “Energy” (positiver oder negativer Strom), “Quantity” (Menge) und “Activated” (Zustand) (siehe Screenshot 16).
Im unserem Beispiel werden wir alle 7 Häuser gleich editieren. Alle “HolonElement”(s) werden den gleichen Graph (betrachten wir als einen Tag) besitzen, außer dem Kühlschrank (ganze Zeit an) und die Solarmodule (produzieren Energie nur tagsüber) (siehe Screenshot 17). (zu Beachten: der Graph enthält allgemeine Einheiten. Im unserem Beispiel betrachten wir den Graph als einen Tag - also 24 Stunden)

8. Schritt: (Speichern und Laden)



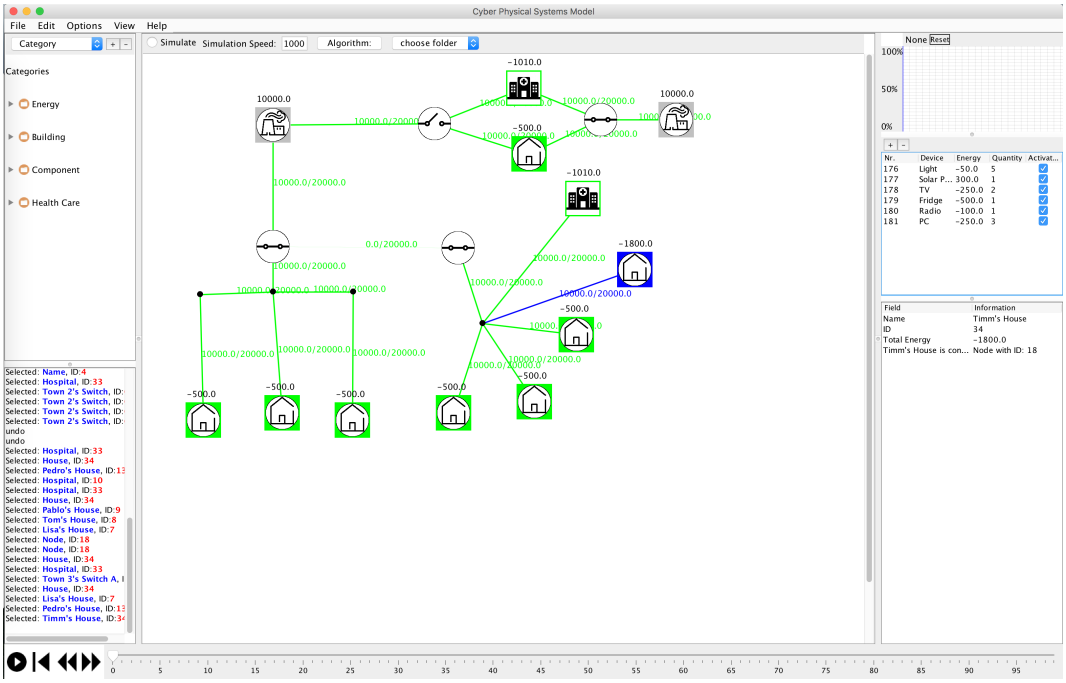
Screenshot 18



Screenshot 19

Der Benutzer hat immer die Möglichkeit seinen Fortschritt zu speichern und in der Zukunft fortzusetzen. Unter dem Fenster “File” findet man die “New”- (Neustart), “Save”- (Speichern vom Zustand) und “Open”-Taste (Laden eines gespeicherten Zustandes).// Im unserem Beispiel werden wir das Program unter dem Namen “Beispiel-QS” speichern (siehe Screenshot 18) und sofort wieder laden (siehe Screenshot 19).

9. Schritt: (Tastatur-Shortcuts)

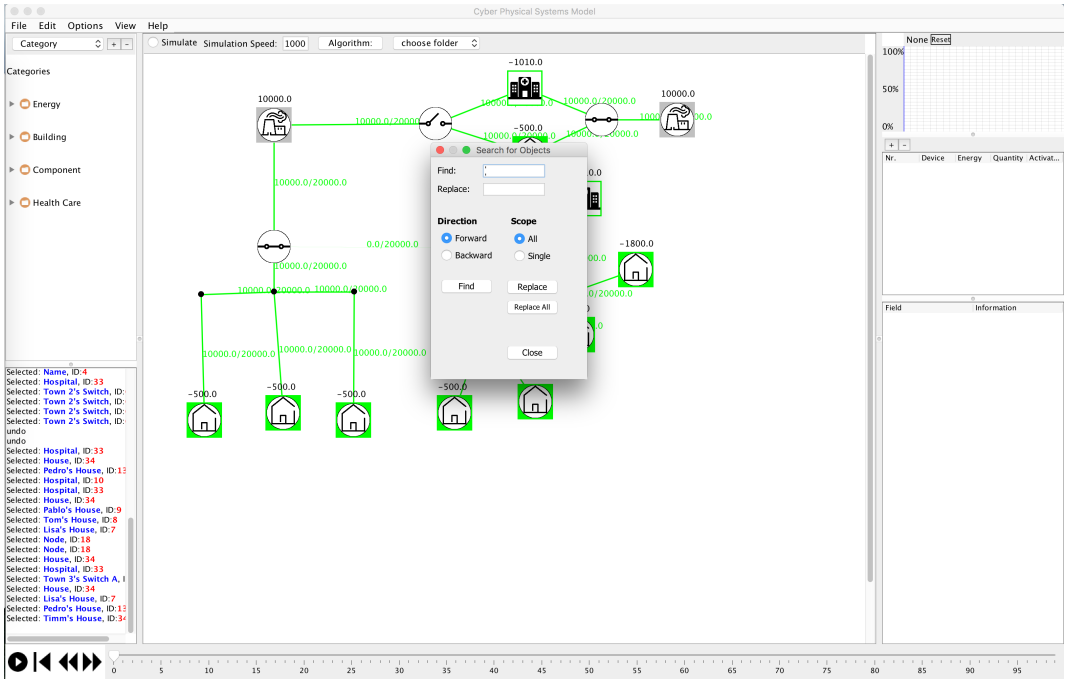


Screenshot 20

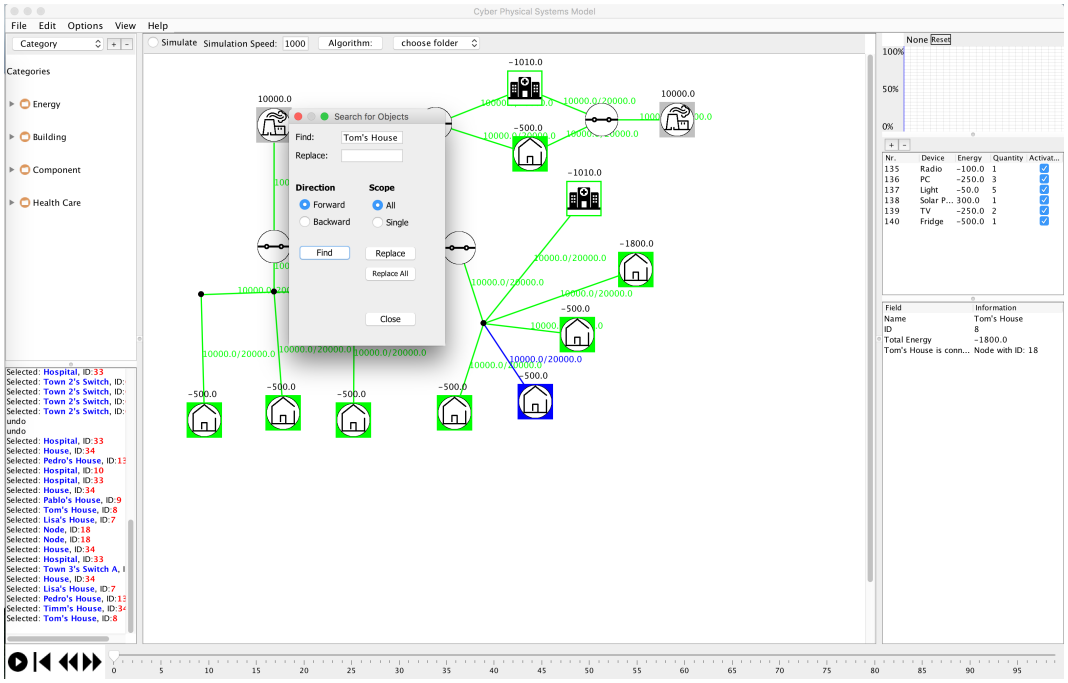
Der Benutzer hat immer die standard Tastatur-Shortcuts zur Verfügung : “Ctrl + X” (Cut), “Ctrl + C” (Copy), “Ctrl + V” (Paste), “Ctrl + F” (Suchen), “Ctrl + Z” (Undo) und “Ctrl +

Y” (Redo).
Im unserem Beispiel werden wir das Krankenhaus und das Haus von Pedro kopieren und bei “Town 2” einfügen. Das Haus von Pedro werden wir auf “Timm’s House” (siehe Screenshot 20).

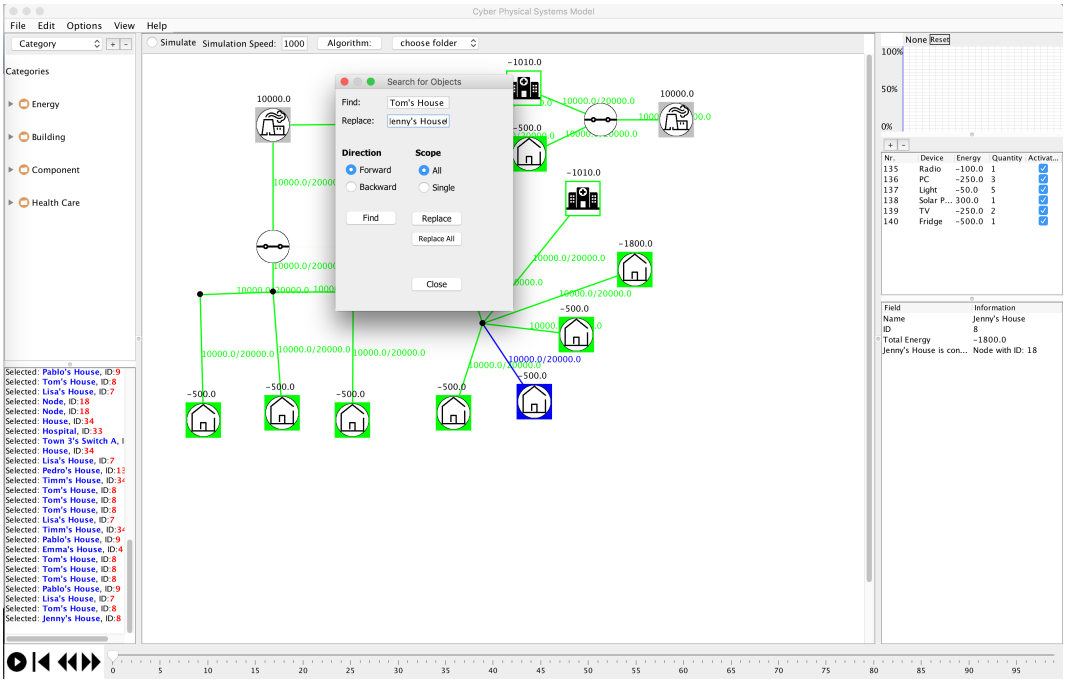
10. Schritt: (Suchen/Ersetzen)



Screenshot 21



Screenshot 22



Screenshot 23

Der Benutzer kann entweder mit dem Tastatur-Shortcut (“Ctrl +F”) oder unter “Edit->Find/Replace” das Find bzw. Ersetzen Fenster aufmachen (siehe Screenshot 21).
Im unseren Beispiel werden wir mittels dem “Find/Replace”-Fenster das Haus von Tom suchen (siehe Screenshot 22) und nachher zu“Jenny’s House” ändern (siehe Screenshot 23).

Damit sind wir mit dem Modellierung-Modus fertig. Als Fazit können wir sagen, dass der Benutzer folgende Tools:

1. Objekte in Canvas per Drag and Drop einfügen
2. Objekte editieren
3. Objekte selbst definieren
4. Verbindungen zwischen Objekten erstellen
5. Stromfluss der Verbindungen manipulieren
6. Schaltern manipulieren
7. Geräte im Objekten manipulieren
8. Speichern und Laden von Fortschritt
9. Tastatur-Shortcuts
10. Suche und ersetzen von Namen

zur Verfügung hat.

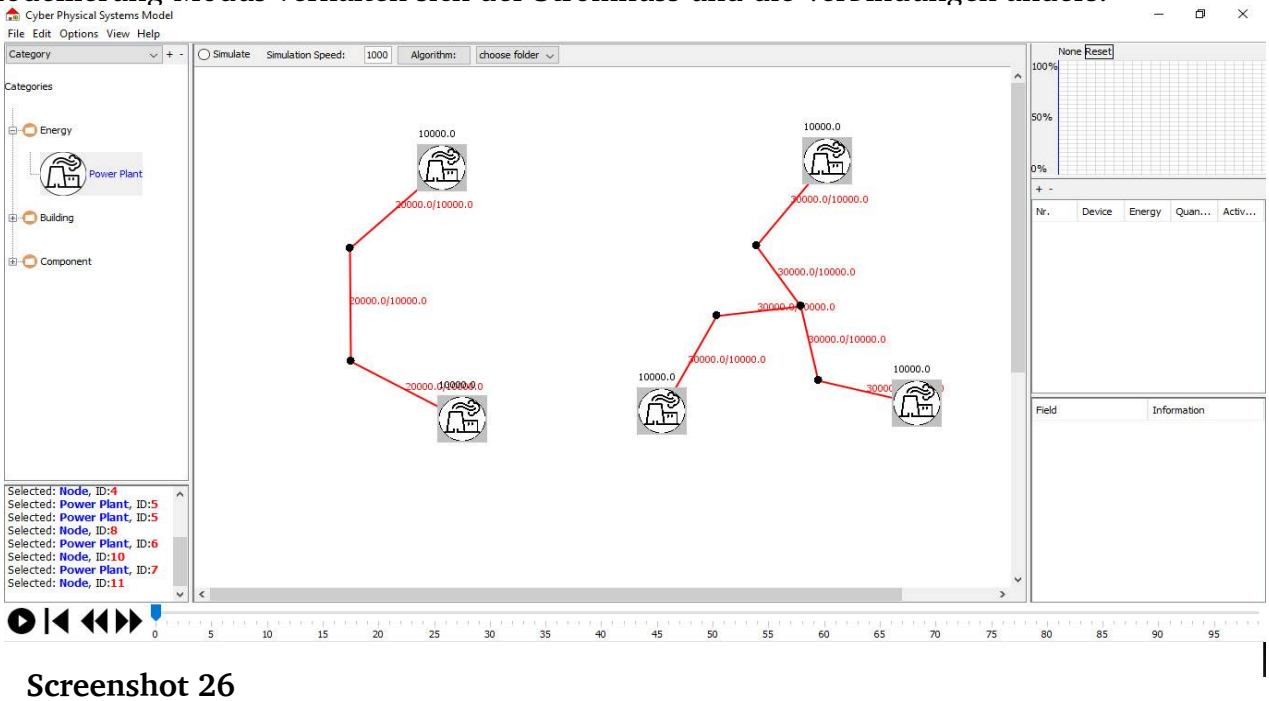
Der Simulation-Modus wird dem Benutzer auch jeder Zeit zur Verfügung stehen, womit der Benutzer seine Modellierung testen kann.

Screenshot 24

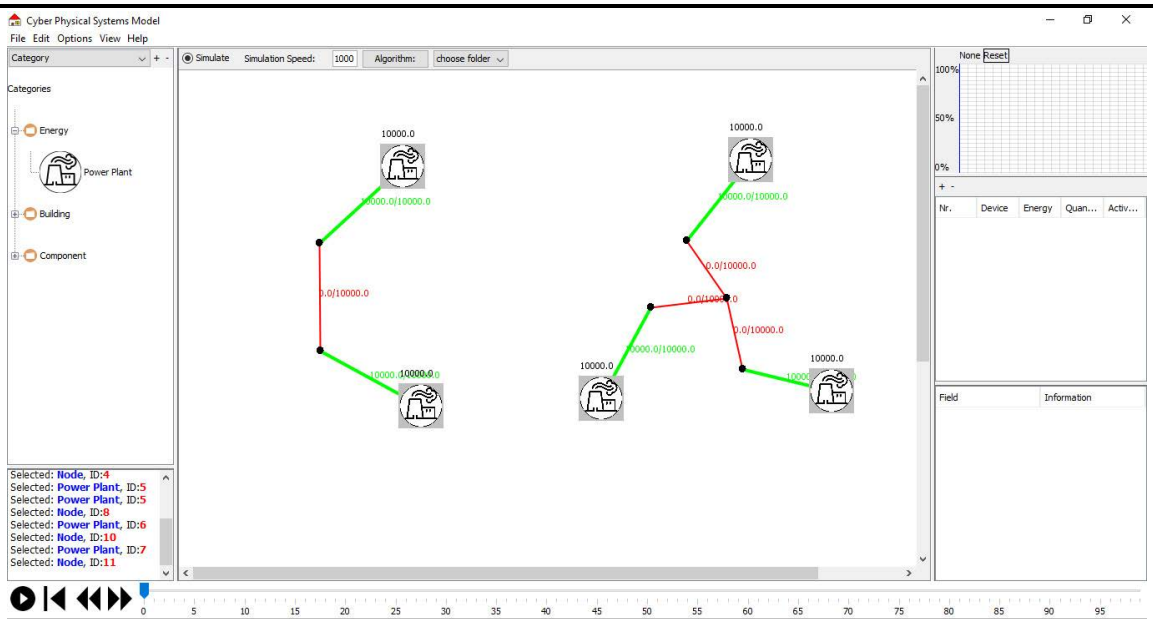
Screenshot 25

Die Simulation besteht aus einem Time Panel (siehe Screenshot 24) und einer Simulation-Leiste (siehe Screenshot 25).
Das Time Panel enthält eine Zeitlinie mit 100 Zeitschritten, eine “Play”-Taste, welche die Simulation startet, eine “Reset”-Taste, welche die Simulation neu startet, eine “Forward”-Taste, welche einen Zeitschritt weiter rückt, und eine “Backwards”-Taste, welche ein Zeitschritt zurück rückt.
Die Simulation-Leiste besteht aus einer “Radio”-Taste (“Simulation”), welche die Simulation an- oder ausschaltet, ein Feld für die Simulationsgeschwindigkeit, eine “Algorithm”-Taste, welche ein Fenster öffnet, um neue Algorithmen zu importieren, und eine “Drop-Down”-Taste, welche alle importierten Algorithmen enthält.

Eine weitere Eigenschaft des Simulation-Modus ist sein Verhalten. Im Gegensatz zu dem Modellierung-Modus verhalten sich der Stromfluss und die Verbindungen anders.



Screenshot 26



Screenshot 27

Im Modellierung-Modus hat der Stromfluss keine “Geschwindigkeit”, dass heißt, jede Verbindung wird unabhängig berechnet (als die Summe alle Stromquellen). Im Gegensatz dazu, wird der Stromfluss im Simulation-Modus so berechnet, dass sobald eine Verbindung bricht, kein Strom mehr durchfließt.
Als Beispiel werden wir zwei Szenarios kreieren. Das erste Szenario besteht aus 2 Kraftwerken, die durch 3 “CpsEdges” verbunden sind. Das zweite Szenario besteht aus 3 Kraftwerken, die durch jeweils 2 “CpsEdges” zu einen Mittelpunkt verbunden sind. Im Modellierung-Modus werden alle Verbindungen rot (kaputt) für beide Szenarios gezeigt (siehe Screenshot 26 - links Szenario 1 und rechts Szenario 2).
Im Simulation-Modus werden nur einige Verbindungen rot (kaputt) gezeigt (siehe Screenshot 27 - links Szenario 1 und rechts Szenario 2).
Für das Szenario 1 werden zuerst die direkten Verbindungen berücksichtigen (Also Verbindungen zwischen Kraftwerk 1 und Knoten 1 und Kraftwerk 2 und Knoten 2). Im nächsten Schritt wird die Berechnung der dritten Verbindung durchgeführt (zu beachten: die Kraftwerke produzieren jeder 10000 Einheiten Strom und die maximale Kapazität der dritten Verbindung entspricht 10000 Einheiten), welche eine Störung bzw. Bruch der Verbindung impliziert.
Das Szenario 2 wird sich ähnlich verhalten, aber mit dem einen Unterschied, dass die Störung im 3. Schritt stattfinden wird. Die Verbindungen zwischen Mittelpunkt und Knoten werden im 2. Schritt berechnet. Danach im Schritt 3 werden alle Werte addiert (10000 Einheiten pro Kraft-werk = 30000 Einheiten) und weiter geleitet. Daraus folgt eine vielfache Störung der Verbin-dungen.