Seminator

Dokumentation

Teilnehmer: Tim Farahani, Christoph Fischer, Dominik Grabetz, Elias König, Camillo Krause genannt Barmann, Lukas Mettler, Luca Rutschmann, Anthony Schäufele, Niklas Uhr

Schule: Albert-Einstein-Schule, Ettlingen

Klasse: TGJ 1/2

Fachlehrer: Herr Würz

Fach: Seminarkurs Informatik

Datum: 3. Juli 2017

Inhaltsverzeichnis

1 Einleitung

1.1 Warum ein Digitalsimulator?

Schüler mit dem Profilfach Informatik werden bereits in den ersten Stunden mit logischen Bausteinen konfrontiert und auch einige andere Profilrichtungen arbeiten mit dieser Digitaltechnik, um logische und technische Schaltungen zu verwirklichen.

Standardmäßig wird das kostenpflichtige Programm ISIS (Intelligent Schematic Input System) von Proteus zur Simulation verwendet. Dieses Programm beinhaltet mit über 10.000 Komponenten definitiv mehr, als die im Normalfall benötigten Operationen. Schüler der Digitaltechnik nutzen zu Beginn nur Elementare Grundbausteine und lernen später, je nach Profilfach, ein paar Fortgeschrittene kennen. ISIS ist eindeutig für riesige und sehr komplexe Projekte konzipiert worden und nicht für Schüler, welche mit dieser quasi „unendlichen“ Auswahl nur verwirrt werden.

Um dieses Problem zu beheben, wurden Alternativen gesucht. Natürlich fand man andere Digitalsimulatoren, wie z.B. simulator.io, allerdings sind diese meist umständlich, fehlerhaft oder funktionieren gar nicht erst, weshalb man die Suche am Ende ernüchtert aufgab.

Nach der gescheiterten Suche, beschlossen die Lehrkräfte einen eigenen Digitalsimulator, speziell auf ihre Bedürfnisse zugeschnitten, zu entwerfen. So entstand dieser Seminarkurs, mit Schülern, die hier ihre Chance sehen, ihre Liebe zur Digitaltechnik zu beweisen.

1.2 Zielformulierung

Ziel unseres Seminarkurses ist es, einen Digitalsimulator zu entwerfen, der möglichst einfach und einsteigerfreundlich zu bedienen ist, um neuen Schülern einen einfachen Einstieg zu ermöglichen. Dieser Simulator soll sich nur auf die wichtigsten Bausteine fokussieren und in einer portablen Version vorhanden sein, damit die Schüler auch zuhause damit arbeiten können. Er soll so programmiert werden, dass zukünftige Seminarkursgruppen diesen schnell und ohne große Probleme verstehen und erweitern können.

2 Timeline

In diesem Kapitel dokumentieren und erklären wir unsere Arbeit nach der Zeit, sprich das Kapitel beschreibt unsere Arbeit und Beweggründe von Anfang bis Ende.

2.1 Wo anfangen? Die Entstehung des Designs!

Nun, wir haben unsere Seminarkursgruppe und das gewünschte Thema, doch wir stehen bereits vor dem ersten Problem. Wo sollen wir anfangen? Was sollen wir tun?

Um das erste Problem zu bewältigen, setzen wir uns hin und diskutierten über einen Ansatzpunkt. Es gibt natürlich zahlreiche Möglichkeiten, wir könnten eine Programmiersprache aussuchen, Aufgaben verteilen, die Elemente festlegen etc. Wir entschieden uns allerdings, erst das Design festzulegen und uns dann daran zu orientieren. Um die Innovation möglichst groß zu halten, beschlossen wir, dass jeder sein eigenes individuelles Design vorstellen darf.

Beim nächsten Treffen, erhielten wir einige Skizzen. Wir einigten uns schnell auf ein Design, dass dem von ISIS gar nicht so fern ist, Elemente auf der linken Seite, Fehlermeldungen und Startknopf unten etc. Es ist allerdings deutlich reduziert worden und ermöglicht daher eine bessere Übersicht, sowie Einstieg. (Rechts: Skizze des ersten Designs)

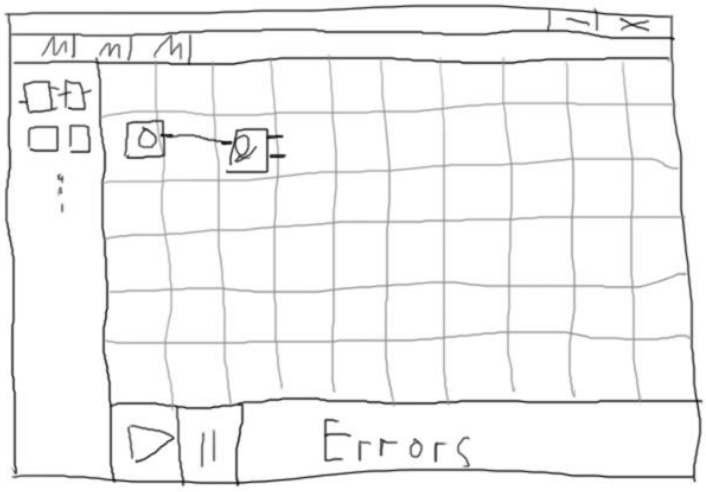


Abbildung 1 Programmaufbau Skizze

Natürlich war dieses erste Konzept noch weit davon entfernt, perfekt zu sein, weshalb wir noch ein paar Änderungen vornahmen. Allerdings wollten wir auch kein festes Design festlegen, denn es soll sich während der Entwicklung stetig Anpassung und verbessert können. Für das Hilfe- & Einstellungsmenü legten wir noch kein Design fest, wie auch, wenn wir noch nicht wissen, welche Optionen es geben wird.

Wir haben uns also für ein Design entschieden, dass dem, von ISIS in der Position der Operationen ähnelt, allerdings reduziert ist, damit sich Schüler schneller zurechtfinden. Nach Erarbeitung des Designs, stellt sich die nächste Frage: Wie bringen wir das Design funktionstüchtig auf den Bildschirm?

2.2 Java - the way to go

Bevor wir unser Design in die Tat umsetzen, müssen wir uns selbstverständlich erst einmal für eine Programmiersprache entscheiden, denn ohne läuft gar nichts.

Das besondere an unserem Seminarkurs war, dass wir bereits sehr talentierte Leute im Umgang mit Programmiersprachen hatten, welche uns daher beraten konnten. Es gab einige Vorschläge aus den zahlreichen Programmiersprachen, wie Python, C#, C++ etc., allerdings schieden einige bei der gewollten Plattformunabhängigkeit bereits aus und andere bei der

verlangten Objektorientierung, denn wir verlangten von Anfang an eine objektorientierte Sprache, diese sind einfach erweiterbar, gut für große Projekte und passen super zum Thema. Letztendlich fiel unsere Auswahl auf die weltweite geschätzte Sprache Java von Oracle, denn diese ist Plattformunabhängig, objektorientiert und wird zudem an der Schule gelernt, was bedeutet, dass die Programmiersprache niemandem unbekannt ist. Des weiteren ist Java heutzutage eine sehr leistungsstarke Sprache im Gegensatz zu anderen Sprachen mit einer eigenen virtuellen Maschine, und ein Digitalsimulator soll selbstverständlich alles an Leistung rausholen.

Abbildung 2 Java

Natürlich wird in diesem Seminarkurs mehr als nur das bisschen Grundwissen benötigt, dass man bis zur 12. Klasse lernt, daher waren einige gefordert sich ein wenig in Java und natürlich auch OOP einzulesen.

2.3 Zeichnen mit JavaFX

Um eine Oberfläche zu realisieren braucht unser Programm ein Framework, eine Art grafisches Grundgerüst für Anwendungen. In Java gibt es davon mehr als genug, was die Auswahl nicht einfacher machte. Für uns kamen jedoch nur die heutigen GUI-Standards in Frage, was unsere Auswahl auf, das 2009 erschienene, JavaFX und, das auf AWT aufbauende, Swing beschränkte. Wir informierten uns bis zum nächsten Treffen über beide Frameworks und entschieden uns wenige Tage später einstimmig für JavaFX.

Im Vergleich zeigte sich, dass die JFX Bibliotheken viele Teile Swings vereinfachen und neue Möglichkeiten bieten. Zudem überzeugte es mit einem moderneren und abgerundeteren Erscheinen als Swing. Desweiteren besitzt JavaFX moderne Features an denen es Swing fehlt, wie z.B. die Möglichkeit die Designs von FX-Elementen per CSS (Cascading Style Sheets) festzulegen oder das von JFX von Beginn an mitgebrachte Multithreading.

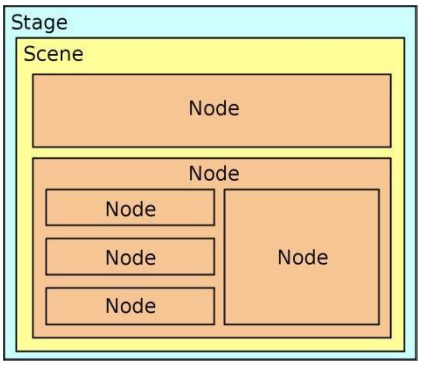


Abbildung 3 JavaFX Fensterstruktur

Um das Erstellen von grafischen Oberflächen mit JFX zu vereinfachen, wurden wir auf den von Oracle veröffentlichten JavaFX Scene Builder aufmerksam. Ein solcher GUI-Builder ermöglicht es, durch Drag & Drop Steuerelemente, Behälter, Layouts, Formen und vieles mehr einfach zu platzieren. Diese so erzeugte GUI wird in einer FXML-Datei gespeichert, deren Inhalt auch per Hand verändert werden kann.

Eine grafische Oberfläche in JavaFX ist hierarchisch in 3 Ebenen unterteilt. Die Oberste Ebene ist die Stage, der Rahmen einer Anwendung der den eigentlichen Inhalt enthält. In einer Stage befindet sich eine Scene. Sie enthält die eigentlichen Steuerelemente, den sogenannten Nodes.

2.4 Objekt orientierte Programmierung (3-Schichtarchitektur)

Begriffserklärung: Objekt orientierte Programmierung ist eine Methode der Informatik, welche die Architektur eines Programms so gestaltet, dass seine Objekte dem realen Anwendungsbereich entsprechen.

Bei Objektorientierter Programmierung, wird bewusst auf die funktionale Programmierweise verzichtet, welche sämtliche Methoden, Funktionen, Variablen, etc. in einem einzigen „File“ abspeichert. Es wird danach geschaut Programme in der sogenannten 3-Schicht-Archtitektur zu gestalten. Diese setzt sich aus mehreren Dateien (Files) zusammen, welche auf die erste Schicht, „die Oberfläche“, die zweite Schicht, „Die Steuerung“, oder die dritte Schicht, „Daten“, aufgeteilt werden können. Die Files übernehmen hierbei immer eine ganz spezielle Aufgabe als Teil eines großen Ganzen (Vergleichbar mit einem Zahnrad in einem Getriebe). Bei der Kommunikation wird darauf geachtet, dass die Beziehung stets Unidirektional sind, wobei bei der Beziehung zwischen Oberflächen- und Steuerungsschicht, durchaus auch bidirektionale Beziehungen möglich sind.

Da dieses Programmierprinzip von Herr Würz gewünscht war, haben wir dieses auch verwendet. Der Seminator verfügt genau diese drei Schichten. Er besitzt eine Oberfläche auf der der User seine digitalen Schaltungen entwerfen kann. Dabei werden dauerhaft seine Aktionen auf der Oberfläche von einer Steuerung verarbeitet welche entsprechend auf die Interaktionen reagiert und aus der Daten-Schicht die nötigen Informationen organisiert, um sie wieder an die Oberfläche weiterzugeben.

2.5 Konzept der Elemente

Von Anfang an war die Aufgabenstellung von Herr Würz gegeben, dass das zu entwickelnde Programm einfach zu erweitern sein sollte für zukünftige Seminarkurse. Um diesen Teil des Seminarkurses zu erfüllen, bedienten wir uns des Prinzips der Vererbung aus der objektorientierten Programmierung.

Vererbung bedeutet, laut dem Duden, „die Weitergabe von Erbanlagen von einer Generation an die folgende“. In der Informatik wurde dieses Prinzip ebenfalls angewandt, in dem man Klassen, welche sich in einem oder mehreren Attributwerten gleichen, diese Attributwerte entnimmt und in einer sogenannten Superklasse zusammenfasst. Von dieser Superklasse erben von nun an alle vorherigen Klassen (Subklassen), welchen die gleichen Attribute

entnommen wurden, die Attribute wieder. Dies spart nicht nur Programmcode, sondern ermöglicht es auch einfach neue Klassen zu erschaffen, die über jene gleichen Attribute verfügen, sich jedoch im weiteren Code von ihnen unterscheiden, in dem man sie wieder von der Superklasse erben lässt (Vererbung wird in der von uns verwendeten Sprache Java mit dem Befehl „extends NamederSuperklasse“ eingeleitet). Charakteristisch treten hier die in der objektorientierten Programmierung verwendeten Sichtbarkeiten auf, welche entscheiden ob ein Objekt nachdem es vererbt wurde auch bearbeitet werden kann.

- public (+) // ist Bearbeitbar
- private (-) // ist nicht Bearbeitbar
- protected // ist Bearbeitbar

In unserem Seminarkurs, wurde sich dessen bei der Entwicklung der Digitalen Bausteine bedient, welche sich abgesehen von ihren digitalen Logiken, programmiertechnisch kaum voneinander unterscheiden. Somit wurden von uns gewisse Grundeigenschaften (Grundattribute) festgelegt, über die jedes Element (digitaler Baustein) später verfügen soll. Deswegen findet man in jeder Element_Elementname.java Datei denn Befehl „extends Element“ bei der Initialisierung der Klasse, welcher befiehlt von der Klasse Element die vorher festgelegten Grundattribute zu erben.

Die Grundeigenschaften(siehe Element.java):

Was für eine Element ist es (*public static final String TYPE = "ELEMENT"*

Ihre zugehörige Bauteilgruppe (*protected Group grp*)

Die Anzahl ihre In-/Outputs (*protected int numOutputs, protected int numInputs*)

Ihre Größe auf dem Bildschirm (*protected static double elementWidth = x, protected static double elementHeight = x*)

Jeweils ein Array um die Zustände der In-/Outputs zu speichern (*protected int[] inputs, protected int[] outputs*)

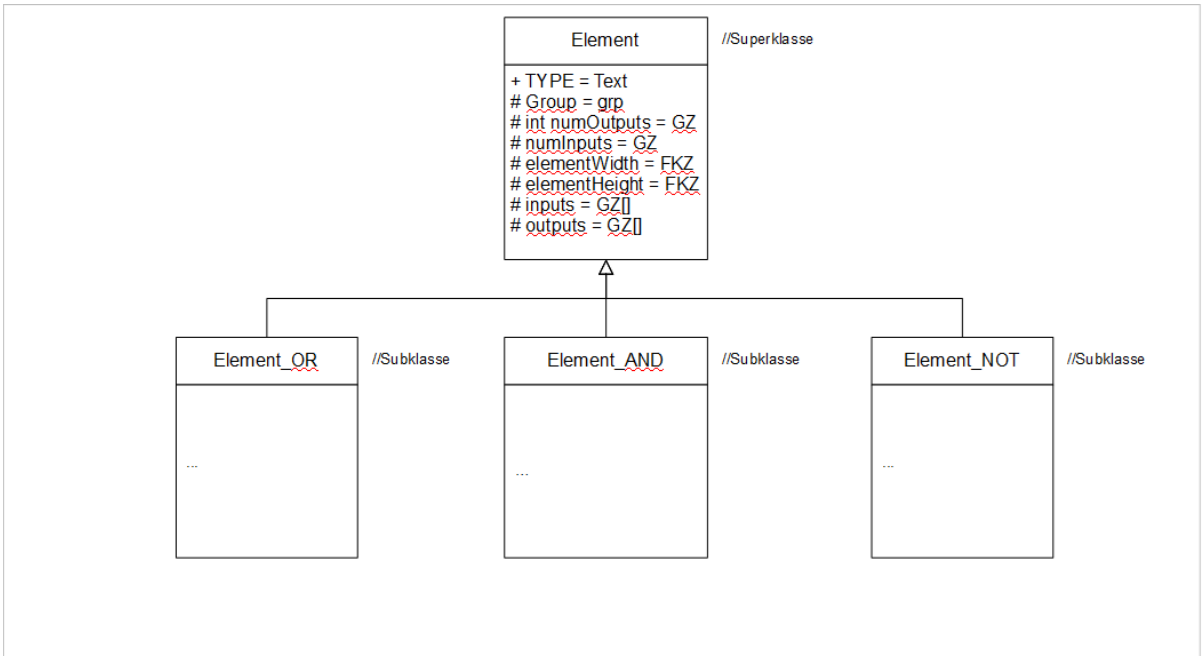


Abbildung 4 Element Vererbung UML

2.6 Blitzschnelle Simulation dank Multithreading

Um die Geschwindigkeit und Funktionalität unseres Programms (speziell der Simulation) zu erhöhen, haben wir uns dazu entschieden, Multithreading einzubauen.

Das Konzept von Multithreading in der Informatik ist relativ simpel. Man hat mehrere „Threads“, also Unterprogramme, welche Zeitgleich verschiedene Operationen ausführen können. Diese laufen auf verschiedenen Prozessorkernen, daher ist es für den Computer einfacher eine große Menge an Daten zu verarbeiten, da die Menge auf die Kerne aufgeteilt wird. Man zerlegt also einen großen, komplexen und daher langsamen Prozess in kleinere Programme, um diese parallel abzuarbeiten, was zu einem deutlich schnelleren Ergebnis führt. Im Gegensatz zum Multitasking, sind die Threads nicht voneinander isoliert, was eine Kommunikation zwischen ihnen ermöglicht.

Multithreading ist nun schon seit einigen Jahren Standard in Computern, dies macht auch Sinn denn, will man z.B. Musik hören und dabei im Internet surfen, muss der Computer sich entscheiden was er nun macht, entweder er spielt einen Laut ab, oder er lädt eine Webadresse. Da ein physischer Prozessorkern nicht Zeitgleich zwei oder mehr Operationen übernehmen kann, ist der Rechner gezwungen schnell zwischen allen Prozessen hin und her zu schalten, was dank enormer Prozessorleistung so gut funktioniert, dass dem Menschen das andauernde abwechseln gar nicht erst auffällt. Dies nennt man Software-Seitiges Multithreading, Aufgaben werden also scheinbar gleichzeitig ausgeführt, in der Realität werden sie jedoch nur extrem schnell hintereinander in einem Prozessorkern abgearbeitet. Doch die fortschreitende Technik kam irgendwann an dem Punkt an, bei

welchem einzelne Prozesse die gesamte Aufmerksamkeit eines Prozessors benötigten. Die Steigerung der Prozessorgeschwindigkeit kam an ihre Grenzen, weshalb Forscher gezwungen waren, „echtes“ Multithreading zu entwickeln. Von „echtem“ oder Hardware Multithreading spricht man, wenn einem Prozessorkern tatsächlich nur einer Aufgabe zugeteilt ist. Verschiedene Kerne übernehmen also verschiedene Prozesse, die dann auch tatsächlich und nicht nur scheinbar gleichzeitig abgearbeitet werden.

Natürlich gibt es auch hier einen Haken, denn die verschiedenen Threads müssen möglichst unabhängig und selbstständig arbeiten können. Wenn ein Thread A z.B. dauerhaft die neuesten Daten eines anderen Threads B benötigt, muss A eventuell seine Arbeit für eine lange Zeit aufgeben, bis B fertig ist, was A sehr ineffizient macht.

Programmierer müssen daher oft darauf achten, dass ihr Code so sauber geschrieben ist, dass Aufgaben optimal erledigt werden können. Es bringt also nichts, ein Programm wild auf mehrere Threads zu verteilen und zu hoffen, dass die Leistung steigt, solange die Threads keine sinnvoll gewählten Aufgaben besitzen.

Das benutzen von Multithreading ist sehr einfach in Java, denn im Gegensatz zu vielen anderen Sprachen, bietet Java dafür eine vorgefertigte Klasse an. Die Klasse „Thread“ ermöglicht es, per Vererbung, Klassen zu erstellen die auf einem anderen Prozessorkern, sofern Verfügbar, laufen und daher parallel abgearbeitet werden können. Man erstellt eine Klasse mit dem gewünschten Programm, macht diese zu einer Kindesklasse von „Thread“ und startet diese. Mehr muss man nicht machen, denn die vorgefertigte Klasse von Java übernimmt den Rest, wie z.B. die Auswahl des Prozessorkerns. Um diese einzelnen Threads zu starten, wird allerdings ein Hauptprogramm benötigt das nach dem Start, oder zum gegebenen Zeitpunkt, die jeweiligen Thread-Klassen startet (siehe Bild).

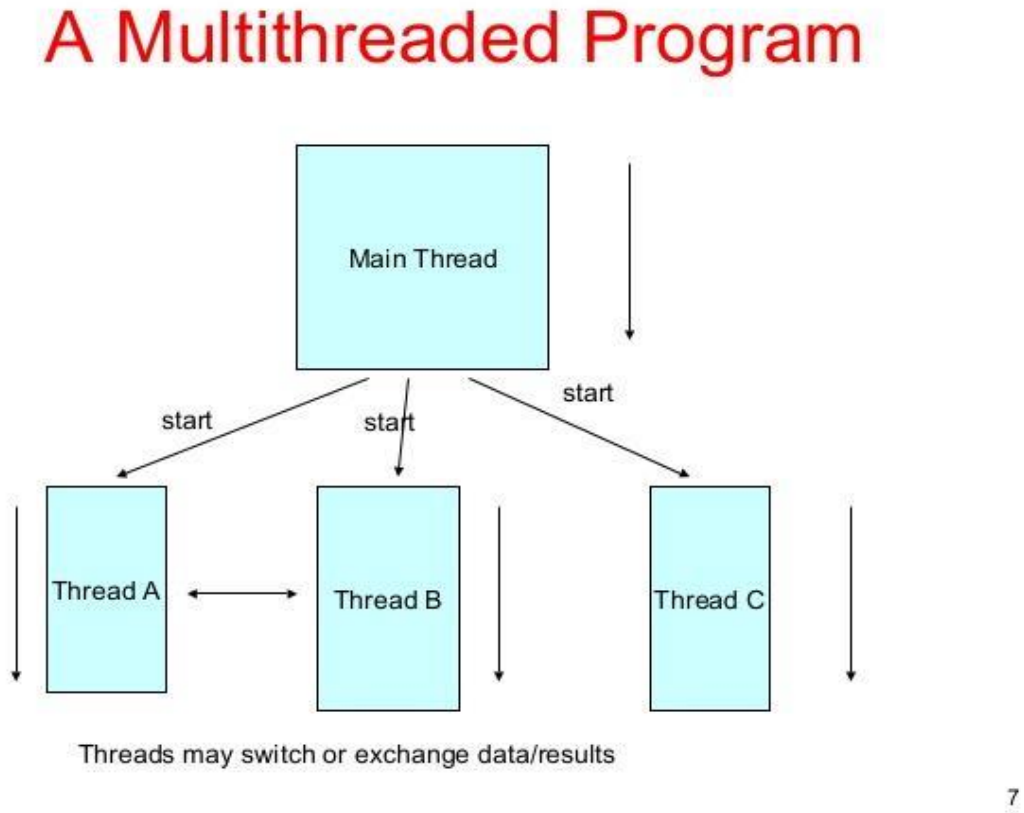


Abbildung 5 Multithreading

Auch wir wollen auf diesen Zug aufspringen, denn Multithreading bietet nicht nur einen enormen Schub in der Geschwindigkeit, sondern löst auch einige Probleme.

Wie in „2.3 Zeichnen mit JavaFX“ erwähnt, benutzt JFX von Natur aus Multithreading, dies hat den Vorteil, dass wenn man in unserem Programm ein Element platziert, dieses in einem anderen Thread gezeichnet wird, während der Rest des Programms noch voll funktionsfähig ist und somit auf weitere Interaktionen reagieren kann. Das selbe Prinzip haben wir bei unserem Pathfinder verwendet, wenn man eine Verbindung legen möchte wird der Weg in einem unabhängigen parallelen Thread errechnet, damit die Oberfläche aktiv bleibt.

Bei der Simulation wurde Multithreading erst richtig essentiell, denn die Rechenleistung, die benötigt wird, um alle Elemente zu simulieren, zu zeichnen und zu aktualisieren ist enorm groß. Das Knifflige dabei ist, dass die GUI dabei ansprechbar bleiben muss, um z.B. bei einem Thumbswitch die auszugebende Zahl zu ändern, oder gar die Simulation zu stoppen. Würden wir unsere Simulation bei so einem Fall kurz unterbrechen, wäre das Resultat eine Verzögerung in der Zeit, weshalb der Takt sehr wahrscheinlich nicht mehr eingehalten werden kann. Um dieses Problem zu beheben, haben wir einen Thread aufgesetzt, welcher nach einem Takt, z.B. alle 20 Millisekunden, die Elemente und Verbindungen updatet, während die GUI natürlich unabhängig in ihrem eigenem Thread läuft. Dadurch wurde nicht

nur unser Problem behoben, sondern wir haben auch einen starken Geschwindigkeits Schub dazugewonnen.

Des Weiteren, lässt sich Multithreading natürlich auch auf jede beliebige unabhängige Aufgabe anwenden, wie z.B. das Übernehmen von Einstellungen oder das Speichern und Laden einer Datei. Die Zerlegung des Hauptprogramms in viele kleine Threads steigert also die allgemeine Performance.

2.7 A* Pathfinding, schön & effizient

Das Pathfinding (das Aufspüren des kürzesten Weges) ist eine Methode, welche bei den Verbindungen zwischen Elementen ihre Anwendung findet. Bei allen erhältlichen Digitalsimulationen ist ein Pathfinder implementiert, denn der Endnutzer will ja nicht für jede Verbindung die er zwischen zwei Bausteinen erstellen will, händisch den Pfad, welchem die Verbindungslinie folgen soll, bestimmen.

So war auch uns klar: wir brauchen einen Pathfinder. Die Suche nach einem passenden Pathfinder war nicht lange. So stellt sich schnell heraus, dass der A* Pathfinder der effizienteste von allen ist. Und das Beste ist: Er findet immer den kürzesten Weg.

2.7.1 Funktionsweise

Bevor der Pathfinder seine Arbeit erledigen kann, müssen Hindernisse erkannt und abstrakt dargestellt werden. Es gibt verschiedene Arten von Hindernissen, so müssen anderen Bausteinen und anderen Verbindungen stets ausgewichen werden.

2.7.1.1 TileCode

Die Aufgabe der Klasse „TileCode“ ist es, alle Hindernisse zu abstrahieren. Letztendlich speichert die TileCode Klasse alle Elemente und alle Verbindungen in einem 2-Dimensionalen Array.

Ein Element sieht abstrakt dargestellt wie folgt aus: (Links: normal, Mitte & Rechts: abstrakt)

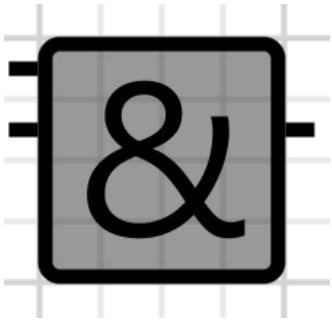


Abbildung 7 Element

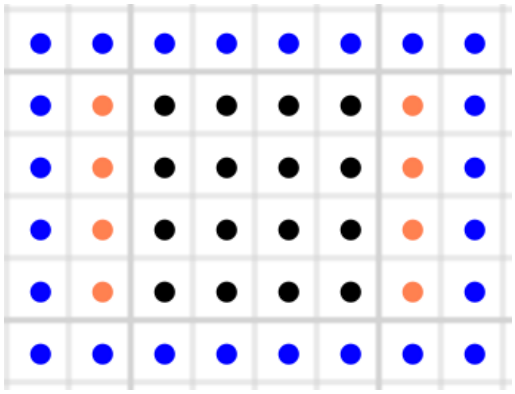


Abbildung 6 Element abstrakt dargestellt (Farbpunkte)

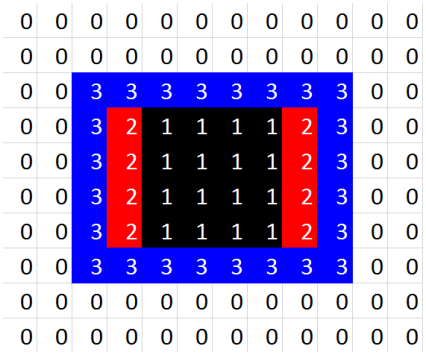


Abbildung 8Element abstrakt dargestellt (Array)

Weshalb die verschiedenen Farben?

- Schwarze Punkte: Sie signalisieren dem Pathfinder, dass durch diese Zellen keine Verbindung verlegt werden kann.
- Orangene Punkte: Durch diese Zellen darf der Pathfinder nur Verbindungen legen, wenn es keine andere Möglichkeit gibt.
- Blaue Punkte: Durch diese Zellen legt ein Pathfinder nur Verbindungen, wenn sonst einem größeren Umweg gefolgt werden müsste.
- Rote Punkte: Diese Punkte haben die gleiche Funktion wie die blauen Punkte

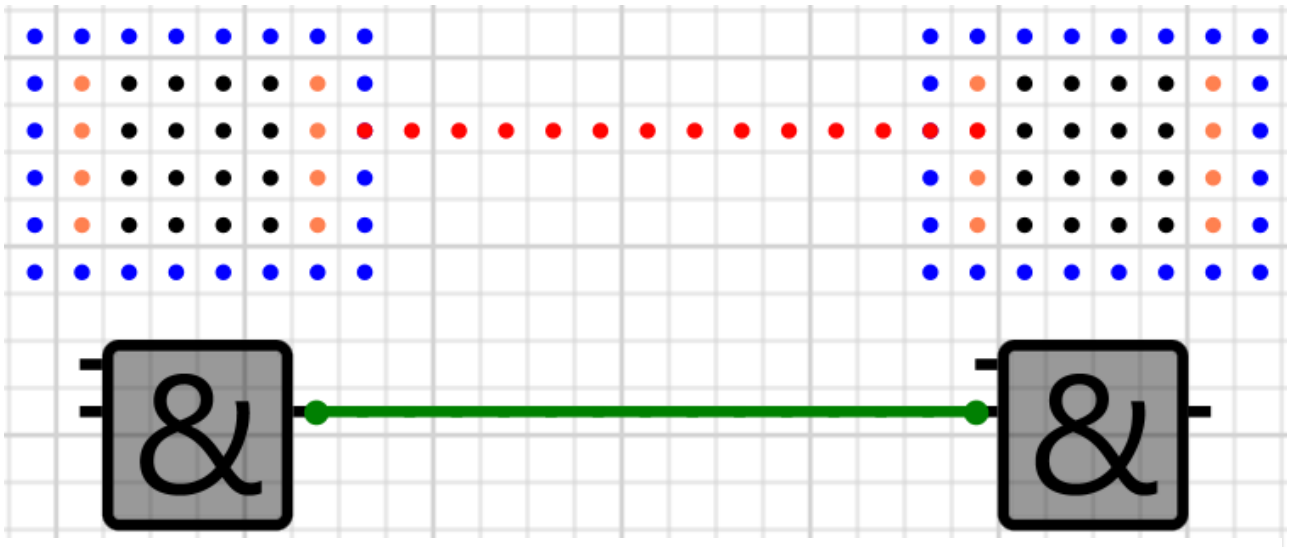


Abbildung 9 Normal & abstrakt

Durch diese Kriterien wird dem Pathfinder eine Art Intelligenz verliehen. Er findet nicht nur den kürzesten Weg, sondern folgt dabei auch noch bestimmten Kriterien.

2.7.1.2 A PathFinder*

Der A* PathFinder findet immer den günstigsten (kürzesten) Weg. So werden dem PathFinder eine Start-, eine Endkoordinate und der wie in 2.7.1.1 gezeigt, generierte TileCode übergeben. Aus diesen Informationen ermittelt der A* Algorithmus dann selbstständig den Weg:

Beginnend bei der Startkoordinate werden alle umliegenden (vier) Felder „aufgedeckt“. Beim Aufdecken werden die Kosten für dieses Feld berechnet: Kosten = (G_{cost} + H_{cost} + A_{cost})

- G_{cost}: Der Abstand (Manhattan-Distance) zu der Anfangskoordinate
- H_{cost}: Der Abstand (Manhattan-Distance) zu der Endkoordinate
- A_{cost}: Zusätzliche Kosten (werden aus dem TileCode-Array berechnet)

Diese vier „aufgedeckten“ Felder werden in einer Liste (openList) gespeichert. Diese Liste wird dann sortiert nach der Höhe der oben berechneten Kosten (Felder mit geringen Kosten stehen dann am Listenanfang und Felder mit hohen Kosten am Listenende). Dann wird das Feld mit den geringsten Kosten genommen und die umliegenden Felder werden „aufgedeckt“ und dieser Prozess wird solange wiederholt bis:

1. Die openList leer ist ➔ Es gibt keinen Pfad zum Zielfeld, oder
2. Ein „aufzudeckendendes“ Feld das Zielfeld ist ➔ kürzester Pfad wurde gefunden

Beispiel:

Der kürzeste Weg zwischen diesen beiden Elementen soll gefunden werden (links):

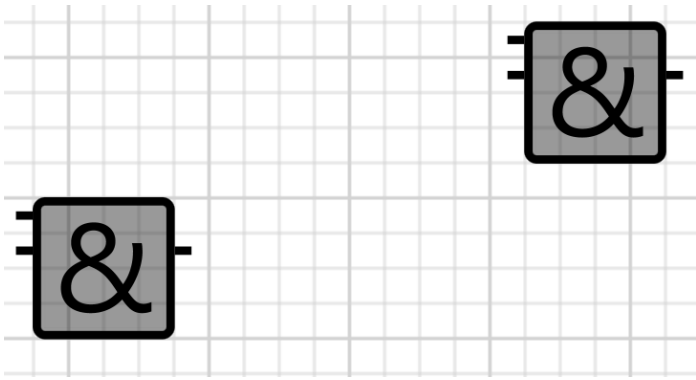


Abbildung 11 Elemente normal

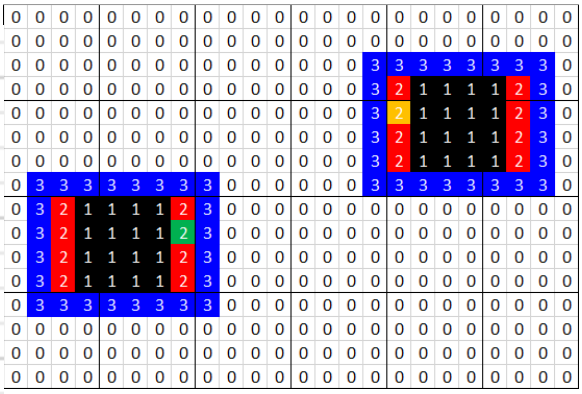


Abbildung 12 Elemente abstrakt

Das TileCode-Array sieht dann wie auf dem rechten Bild aus. Das grüne Feld ist der Startpunkt und das goldene der Endpunkt. Nachdem die Kosten für alle Felder berechnet wurden sieht das wie folgt aus:

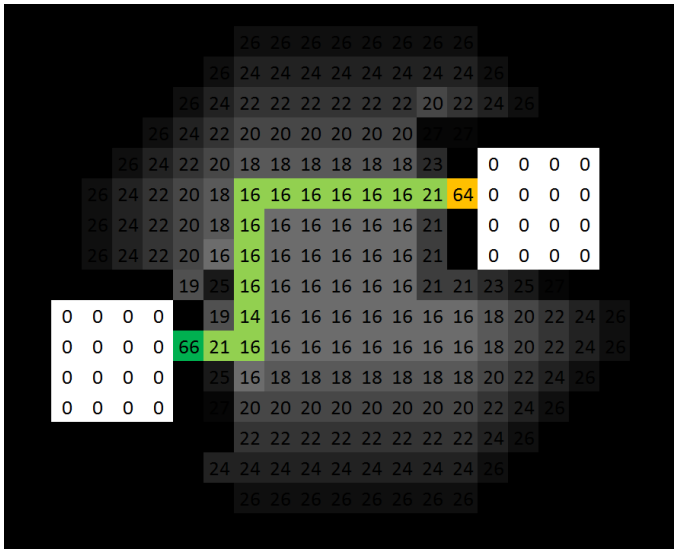


Abbildung 13 Pathfinding Lichtfeld

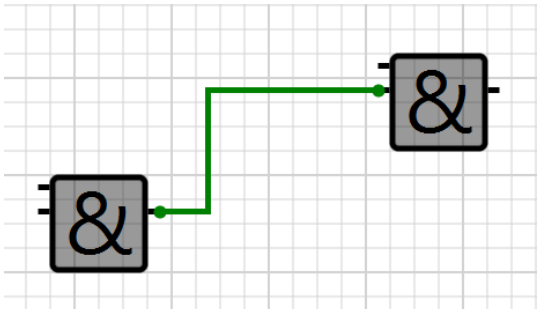


Abbildung 10 Pathfinding normal

In der linken Abbildung stehen die berechneten Kosten. Die dunklen Felder haben hohe Kosten und die hellen Felder haben niedrige Kosten. Die grünen Felder stellen dem optimalen Pfad dar. Das rechte Bild zeigt, wie die Verbindung dann letztendlich von dem Seminator gezeichnet wird.

2.8 JSON der XML-Loader, simples Speichen/Laden

2.8.1 Problemstellung

Eine der wohl wichtigsten Funktionen unseres Digitalsimulators stellt das Laden und Speichern von Projekten dar. Um dies umzusetzen benötigt man eine Möglichkeit ein aktives Projekt zu erfassen und so zu formatieren, dass es in eine Datei geschrieben werden kann, ohne dabei Verluste zu erleiden. Das Prinzip von Laden und Speichern ist recht einfach, allerdings bestehen unsere Projekte aus sehr komplexen und gegenseitig abhängigen Objekte, welche man nicht einfach in eine Datei speichern kann, was die Realisierung dieser Funktion stark erschwert.

2.8.2 Lösung

Da man keine komplexen Objekte speichern kann, sollen diese in primitive Datentypen zerlegt werden (z.B. Ganzzahl, Text), denn diese lassen sich ohne große Probleme und Aufwand speichern. Dieses Prinzip lässt sich sehr gut auf unsere Objekte übertragen. Man kann z.B. einen Baustein in seine „Einzelteile“ zerlegen, welche letztendlich nur aus einfachen Datentypen bestehen, z.B. Position (Fließkommazahl), Anzahl der Eingänge (Ganzzahl) oder eventuell sogar eine gespeicherte Wertetabelle (Ganzzahlen oder Text).

2.8.3 JSON als XML-Loader

Um das obige Prinzip einfach umzusetzen, haben wir beschlossen JSON als XML-Loader zu benutzen. XML ist eine Sprache zum Beschreiben von Zuständen und Objekten und ähnelt daher etwas der Auszeichnungssprache HTML, allerdings schreibt XML keine Datentypen vor, was es ermöglicht, den Inhalt von jeglichen, durch Text und Zahlen darstellbaren, Datentypen zu Speichern.

```
<ePosX>2123.0</ePosX>
<ePosX>2480.0</ePosX>
<ePosX>2417.0</ePosX>
<ePosY>2228.0</ePosY>
<ePosY>2249.0</ePosY>
<ePosY>2081.0</ePosY>
```

Abbildung 14 XML

```
<array>
    <item>0</item>
    <item>5</item>
    <item>3</item>
    <item>7</item>
</array>
```

Abbildung 15 XML

Das obige Bild (links) zeigt, wie die Position unserer Elemente gespeichert wird. „ePosX“ steht beispielsweise für „Element Position auf X-Achse“ und hinter dieser Variable verbirgt sich eine gewöhnliche Fließkommazahl. Wir können diese also in ein XML Dokument schreiben, mit dem jeweiligen Namen kennzeichnen, und beim Laden muss man lediglich wieder den Namen der Variable suchen und den Inhalt Laden. Dieses Prinzip ist auf alle primitiven Datentypen anwendbar und auch selbst auf ein paar komplexe wie z.B. Arrays,

denn XML erlaubt auch die Benutzung einer Hierarchie (Bild rechts), was es ermöglicht mehrere Daten zusammenzufassen.

JSON nimmt uns die Arbeit ab, alle Variablen „per Hand“ abzuspeichern und später durch iterieren wieder zu laden, was uns nicht nur eine Menge Arbeit erspart, es erhöht auch die Performance und beugt Fehler vor, da JSON eine von Experten (Oracle) betreute Software ist.

2.8.4 Benutzung von JSON

Um ein Objekt mit JSON abzuspeichern, muss man in der zugehörigen Klasse die zu speichernden Werte festlegen und mittels einer Notation Kennzeichen (siehe Bild). Meist werden dabei die Getter & Setter der jeweiligen Variable gekennzeichnet. Anschließend kann man das Objekt mit einem, aus der JSON-Klasse, erzeugten Objekt abspeichern und auch wieder Laden.

```
@XmlElement
public double[] getePosX() {
    return ePosX;
}
```

Abbildung 16 JSON

Da wir in unserem Projekten allerdings mehrere (unterschiedliche) Objekte speichern müssen, allerdings nur eine einzige gespeicherte Datei wünschen, legten wir eine Klasse (hier trifft „Schablone“ sehr gut zu) an, in welche wir sämtliche primitive Datentypen unserer Objekte speichern. Diese Klasse lässt sich dann mit JSON als eine einzelne Datei abspeichern und laden. Die Dateiendung haben wir natürlich nicht bei „.xml gelassen“ sondern auf „.dgs“ geändert, damit man die gespeicherten Dateien eindeutig erkennt und zuordnen kann.

2.9 Properties

Der Seminator verfügt natürlich ebenfalls über ein Einstellungsfenster, im Englischen auch Properties. Dort kann man wie üblich in Programmen, ein paar Sachen anpassen

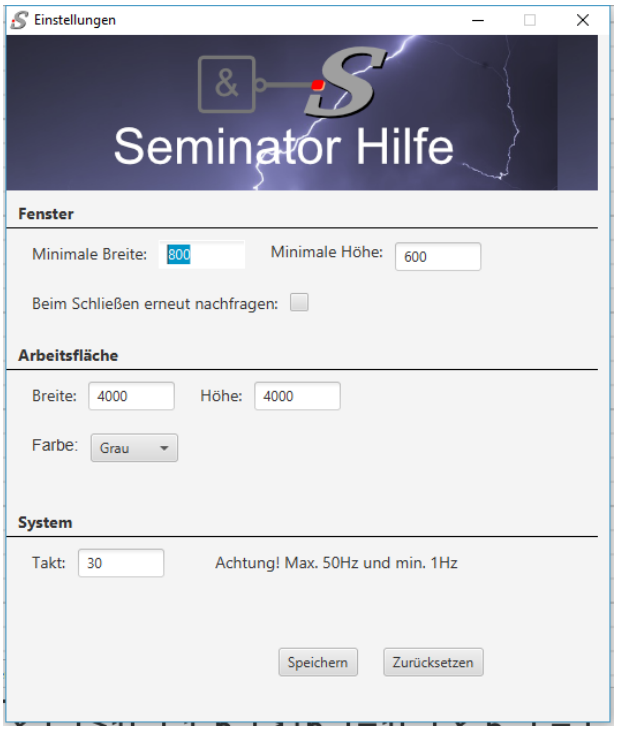


Abbildung 17 Seminator Einstellungen

wie Beispielsweise die Fenstergröße.

Die Einstellungen sind oben links zu finden unter Bearbeiten und dann Einstellungen. Auch über die Tastenkombination Strg + P (= Properties) aufrufbar. Es öffnet sich ein weiteres Fenster mit Einstellungsmöglichkeiten in den Bereichen Fenster, Arbeitsfläche und System (siehe Abbildung 1).

Somit ist es im Fenster Menüpunkt möglich, die Minimale Breite und Minimale Höhe einzustellen, die Standardhaft auf 800x600 gestellt sind. Auch kann man ein Häkchen setzen, wenn man möchte, dass dies beim Schließen nochmals gefragt werden soll.

Bei der Arbeitsoberfläche ist ebenfalls die Gesamthöhe und -breite in Pixel einstellbar. Dabei beträgt der Standardwert 4000x4000. Als kleine Einstellung am Rande, kann man auch das Hintergrundgitter umfärben, in die Farben blau, grün, rot und schwarz. Standarthaft ist hier die Farbe grau gewählt.

Als letztes ist es auch möglich, den Takt des Systems zu ändern, welcher anfangs auf 30Hz steht. Dabei liegt die minimale Einstellungsmöglichkeit bei 1Hz und die maximale bei 50Hz, was nochmals als Notiz nebendran steht.

Eine kleine und letzte Einstellungsmöglichkeit, ist noch bei den Bausteinen zu finden. Und zwar kann man dort mit einem Rechtsklick auf das Einstellungsfenster navigieren und die Anzahl der Inputs wechseln. Diese Option ist bei allen Bausteinen da, jedoch wird eine Änderung nur bei den Bausteinen angenommen, bei welchen die Anzahl auch Sinn macht. Deshalb würde beispielsweise eine Änderung der Inputs bei einer 7- Segmentanzeige nichts bringen.

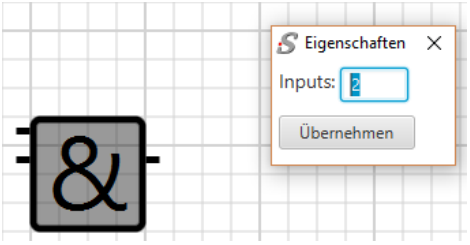


Abbildung 19 Element inputs (2)

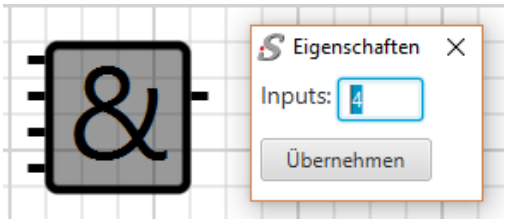


Abbildung 18 Element inputs (4)

3 Bedienung

3.1 Wo ist was?

3.1.1 Altes Design

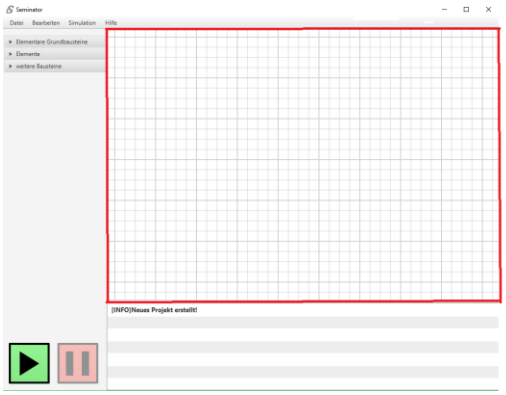


Abbildung 20 Benutzeroberfläche (alt)

Um den Seminator bedienen zu können, wäre es ganz nützlich, wenn man sich kurz ansieht, wo was zu finden ist.

Das Herzstück des Seminators ist natürlich seine Simulationsoberfläche, welche man, durch scrollen mit dem Mausrad, raus zoomen oder auch ranzoomen kann.

Um die Simulation zu starten, gibt es unten links in der Ecke einen grünen „Play-Button“ und einen roten „Pause-Button“, mit denen man die Simulation starten und wieder anhalten kann.

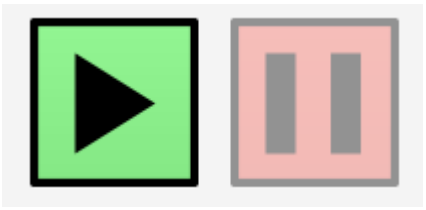


Abbildung 21 Simulation Start Stop

Zu einem Digitalsimulation gehört auch ein „Errorlog“, in dem aufgetreten Fehler während der Simulation aber auch sonstige Mitteilungen, wie zum Beispiel die Nachricht dass ein neues Projekt erstellt wurde.

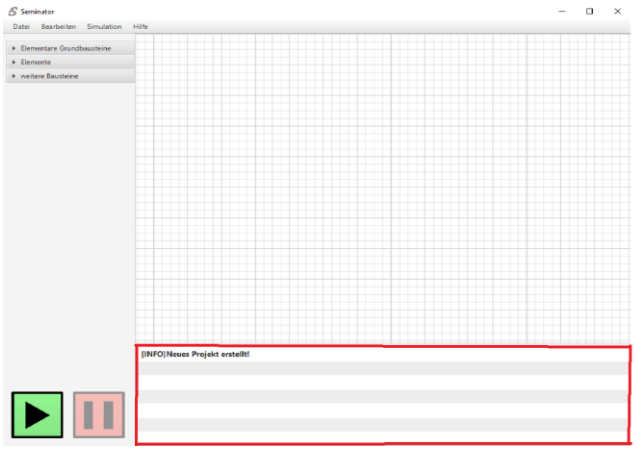


Abbildung 22 Errorlog

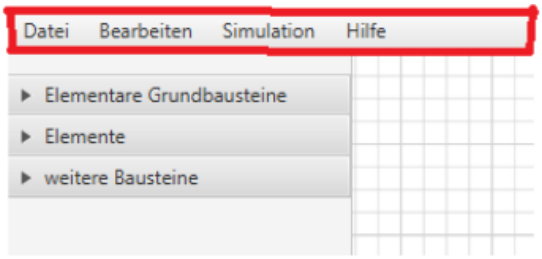
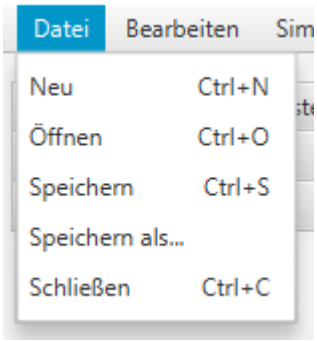


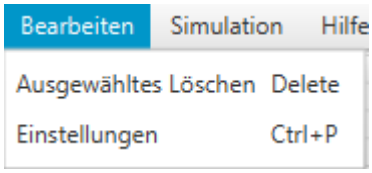
Abbildung 23 Menü

In der Obersten Leiste befindet sich vier unverzichtbare Tools für einen Digitalsimulator.



Mit dem Button „Datei“ ist es möglich, ein neues Projekt zu erstellen als auch ein bereits bestehendes Projekt zu öffnen. Um bestehende Projekte zu besitzen, ist es natürlich auch möglich sie zu Speichern . Um diese Vorgänge zu beschleunigen, sind Abkürzungen neben den jeweiligen Funktionen angegeben.

Abbildung 24 Datei



Unter dem Button „Bearbeiten“, verbergen sich zwei wichtige Optionen. Zum einen kann man einen Ausgewählten Bereich alles Löschen, um zu vermeiden dass man alles einzeln löschen muss.

Abbildung 25 Bearbeiten

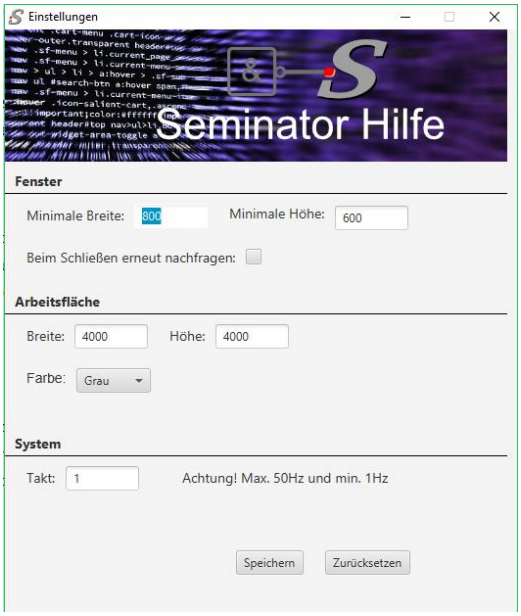
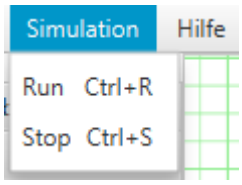


Abbildung 26 Einstellungen

Als zweite Funktion öffnet sich ein Einstellungsfenster.

In diesem Fenster ist es möglich, eine Reihe von Einstellungen für den Digitalsimulator festzulegen. So kann man die Breite und Höhe des Gesamten Fenstern in der verkleinerten Form dauerhaft festlegen. Darunter kann man die Breite und Höhe der gesamten Arbeitsfläche festlegen, und deren Hintergrundfarbe. Ganz unten stellt man den Takt ein, mit der die Simulation laufen soll. Um die Einstellungen zu Speichern ist auch ein Button vorhanden, und zum zurücksetzen auf die Standardeinstellungen auch.



Unter dem Button Simulation, befindet sich nochmals die Möglichkeit, die Simulation zu starten und zu stoppen. Hier sind auch wieder Abkürzungen angegeben, um diesen Vorgang zu beschleunigen.

Abbildung 27 Simulation

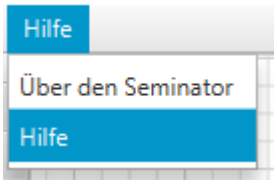


Abbildung 28 Hilfe

Unter der Funktion Hilfe, verbirgt sich zum einen unter „ Über den Seminator“ die Namen der Entwickler, als auch der Rahmen dieses Projekts. Beim Auswählen der Funktion Hilfe öffnet sich ein Fenster in dem alles nochmal erklärt ist damit jeder Nutzer einfach und unkompliziert nachlesen kann, falls etwas unklar sein sollte.

3.1.2 Veränderungen zur Alpha 0.2

Um das Programm angenehmer, für den Nutzer zu gestalten und auch um ein paar Problemen aus dem Weg zu gehen, haben wir uns entscheiden, das Design zu verändern.

Die Auswahl der Bausteine ist nun nicht länger an der Linken, Seite sondern wurde nach unten verschoben. Außerdem kann man jetzt nur noch zwischen Elementaren Grundbausteinen und weiteren Bausteine wechseln. Der Error-log öffnet sich, mit einem Click auf „Console“.

Abbildung 29 Programmaufbau Alpha 0.2

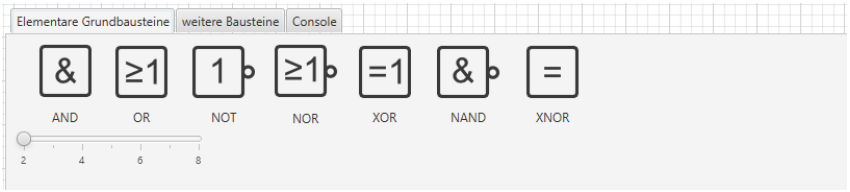


Abbildung 30 Elemente Alpha 0.2

3.2 Die Elemente

Ohne Bausteine, den sogenannten Elementen, macht ein Digitalsimulator allgemein wenig Sinn. Im Seminator können eine Reihe an Bausteinen auf der Arbeitsfläche platziert, und verbunden werden. Diese sind am unteren Bildschirmrand, neben den Play- und Pause-Buttons, zu finden. Die Elemente sind in zwei Kategorien aufgeteilt. Während unter dem Reiter „Elementare Grundbausteine“ die „logic gates“ AND, OR, NOT, NOR, XOR, NOR, NAND und XNOR zu finden sind, kann über die Auswahl von „weitere Bausteine“ eine ganze Reihe sonstiger nützlicher Bausteine erzeugt werden. Neben einem steuerbaren Signal und einer LED, sind auch verschiedene 7-Segmentanzeigen, ein Thumbswitch, eine Clock, welche einen einstellbaren Takt ausgibt, ein Volladierer, verschiedene Speicherbausteine und ein verschiebbares Textfeld erzeugbar.

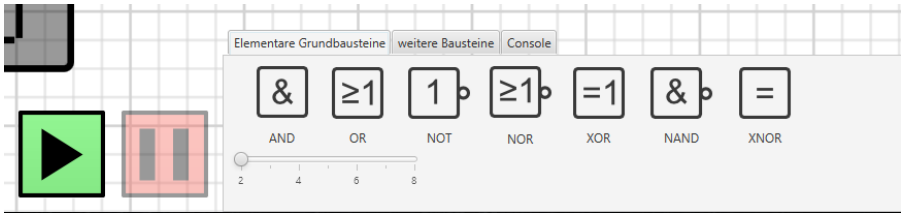


Abbildung 31 Grundbausteine

Per Mausklick kann einfach zwischen den Reitern hin und her gewechselt werden. Ebenso können Elemente mit einem Linksklick ausgewählt werden. Sie müssen nicht per Drag & Drop einzeln auf die Arbeitsfläche gezogen werden, sondern können beliebig oft per erneutem Linksklick an der Stelle des Mauszeigers platziert werden. Um die Auswahl aufzuheben, muss das Element einfach erneut angeklickt, oder ein anderes ausgewählt werden. Alternativ, kann auch die Escape-Taste die Auswahl aufheben .

Elementare Grundbausteine können mit verschieden vielen Eingängen erzeugt werden. Die Anzahl der Eingänge, kann mit einem Slider unterhalb der Elemente festgelegt, jedoch auch später noch geändert werden.

Einzelne, weitere Elemente, wie z.B. das Textfeld, erzeugen zuerst eine Meldung bevor sie auf der Arbeitsfläche erscheinen. Hier kann der Benutzende eine weitere Einstellung vornehmen, wie z.B. das Festlegen des angezeigten Textes.

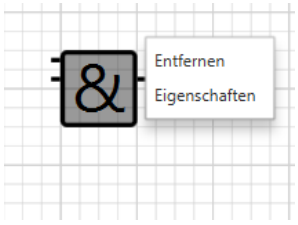


Abbildung 32 Element-Eigenschaften

Sind Objekte einmal auf der Oberfläche erzeugt, können sie mit einem Linksklick verschoben werden. Wird hier bei gedrückter Maus Entfernen gedrückt, wird der Baustein gelöscht, mitsamt all seinen Verbindungen. Alternativ kann mit einem Rechtsklick auf ein Element der Baustein entweder verändert, über den Menüpunkt „Eigenschaften“, oder entfernt werden.

Bausteine können miteinander verbunden werden und so ganze Schaltungen ergeben. Dies geschieht mittels zwei Linksklinken. Durch Auswahl eines Ausgangs, wird eine transparente Linie erzeugt, die dem Mauszeiger folgt. Wird nun ein Eingang angeklickt, erzeugt der Seminator eine Verbindung. Sie können mit Rechtsklick auch entfernt, und zudem zurückgesetzt werden, denn es ist möglich Fixpunkte auf einer Verbindung zu erzeugen, die Verschoben werden können, und so die Verbindung verschieben. An die Fixpunkte können weitere Verbindungen gelegt werden können.

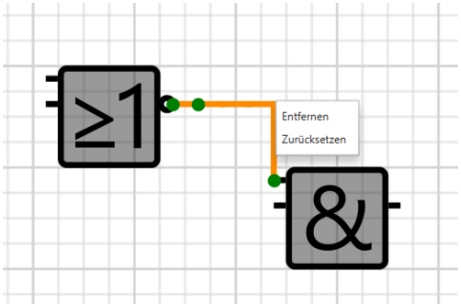


Abbildung 33 Verbindungen

4 Weiterentwicklung

4.1 Ein Element designen und erstellen

Elemente bilden bei einem Digitalsimulator den Hauptbestandteil der Anwendung und wie diese genau diese auf die Oberfläche des Benutzers kommen, behandelt dieser Beitrag.

Beim Erstellen von einem Element, ist erstmals wichtig, dass dieses Element ein Referenzattribut besitzt zu der Oberklasse **Element.java,** um davon die Haupteigenschaften eines Elements zu erben. Dazu zählen beispielsweise die Eingänge und Ausgänge, die von der Oberklasse übergeben werden.

So stehen im Konstruktor des zu erstellenden Elements die Attribute: pX und pY für die Position und pInputs für die Anzahl, der Eingänge. Diese Attribute sind essenziell für die Erstellung eines Element Objekts.

```
//Constructor
public Element_AND(double pX, double pY, int pInputs, NodeGestures dNodeGestures){ //Baustein zeichnen
```

Abbildung 34 Konstruktor Element

Daraufhin folgt ein Array für die Anzahl der Ausgänge. Von Element zu Element, kann optional dann darunter die Anzahl eingetragen werden.

```
pX = pX-elementWidth/2;
pY = pY-elementHeight/2;
```

Abbildung 35 X und Y Koordinaten

Damit der Mauszeiger beim Klicken im Mittelpunkt ist, werden die nebenstehende Operationen beigefügt .Dabei wird von den Koordinaten X und Y die Elementhöhe und weite geteilt durch 2 abgezogen.

Wenn folgendes alles gemacht wurde, kommt man auch schon zum designen/ zeichnen des Elements, so wie es später auch nach dem setzten auf der Oberfläche zu sehen sein wird. Dabei nutzt man die von Java vorgefertigten Operationen. In diesem Fall, ist es immer ein Rechteck, dass die Form eines Elements bildet. Somit wird die Operation, wie in folgender Abbildung verwendet.

```
rec = Draw.drawRectangle(pX, pY, elementWidth, elementHeight, 10, 10, Color.BLACK, Properties.getElementOpacity(), 5);
```

Abbildung 36 Rechteck erstellen

X und Y, die Höhe und Breite, die Stärke der Rundung und die Farbe als Attribute enthält.

Damit wäre der äußere Rahmen des Elements geschaffen, jedoch wäre es jetzt nur ein leeres Rechteck mit Einängen. Da nach der Formelsammlung jedes Element ein Symbol im inneren des Rechtecks enthält, um diese zu unterschieden, fügen wir mit einer weiteren Zeichenfunktion eine Bezeichnung hinzu.

Eine Bezeichnung oder auch Label im englischen, wird ganz einfach mit der Operation auf folgender Abbildung dargestellt. Dabei gibt man dieser Bezeichnung einen Namen und gibt an wo sich dieser Text befinden soll, wie der Text lautet, welche Farbe er haben soll, … und die Größe der Schrift.

```
lbl = Draw.drawLabel((pX+2), (pY - 17), "=" , Color.BLACK, false, 75);
lbl2 = Draw.drawLabel((pX+40), (pY - 15), "1" , Color.BLACK, false, 75);
```

Abbildung 37 Labels erstellen

Soweit wäre das Element auf der Oberfläche schon fast vollständig, jedoch fehlen uns noch die Ausgänge, die nicht bei jedem Element notwendig sind, aber bei den meisten benötigt werden.

Auch diese werden ganz einfach mit einer Operation gezeichnet und stellen somit kein Hexenwerk dar (siehe folgende Abbildung).

```
outputLines.add(Draw.drawLine((pX + 100), (pY + 29.5), (pX + 90), (pY + 29.5), Color.BLACK, 5));
```

Abbildung 38 In- und Ouputs zeichnen

Zuletzt fehlt nur noch die Funktion des Elements. Diese ist natürlich von Baustein zu Baustein verschieden, unterscheidet sich aber nur vom Inhalt. Eine Element-Klasse, wie schon oben beschrieben, ist eine Unterklasse und erbt von einer Oberklasse, in der alle benötigten Funktionen drin stecken. So muss man bei der Funktion eines Bausteins nur die abstrakte Methode **update()** überschrieben werden und mit dem Inhalt für den Baustein seiner Wahl befüllt werden. Am Beispiel von der 7 Segment Anzeige (siehe beistehende Abbildung), werden mehrere If-Abfragen getätigt um die Richtigen Linien zu kennzeichnen.

```
@Override
public void update(){



    if(inputs[0] == 1){
            lineSeg0.setStroke(Color.RED);
            }else if (inputs[0] == 0){
               lineSeg0.setStroke(Color.GREY);
            }
    if(inputs[1] == 1){
            lineSeg1.setStroke(Color.RED);
            }else if (inputs[1] == 0){
               lineSeg1.setStroke(Color.GREY);
            }
```

Abbildung 6

4.2 Kurzer Überblick im Quellcode

Da in einem großen und komplexen Programm sehr viele Funktionen und Klassen benötigt werden, folgt hier ein kurzer Überblick über den Quellcode und eine Erläuterung, wo man was findet. Zunächst ist anzumerken, dass, um den Quellcode übersichtlich zu halten, die Klassen in verschiedene Packages unterteilt wurden, sodass man sich ganz einfach im Code zurechtfindet. Diese sind:

- connection: sorgt für die logische Verbindung zwischen den Elementen, die miteinander verbunden wurden
- digitsim: enthält den Controller, der alles miteinander verbindet, zudem den Quellcode der den Thread ermöglicht und noch die Hauptklasse.
- digitsim.data: hier werden die im Programmlauf gezeigten Bilder der Elemente gespeichert.
- element: in diesem Package befindet sich der Quellcode für jedes einzelne Element.
- general: beinhaltet die Voreinstellungen im Einstellungsfenster und ermöglicht das Speichern und Laden von Einstellungen, zudem werden hier die Kabel auf der Arbeitsfläche erzeugt.
- gestures: erlaubt die Interagierung der Maus, zoom und erstellt das karierte Muster auf der Arbeitsfläche.
- help: enthält die Hilfefunktion des Digitalsimulators und ruft die zugehörige html Datei auf.

- help.Bilder: in diesem Package befinden sich dieBilder, die in der Hilfefunktion genutzt werden.
- pathFinder: die enthaltenen Klassen erlauben die Berechnung des besten Weges für die Kabel, sodass diese nicht unnötig lang werden und dabei nicht kreuz und quer verlaufen.
- properties: hier wird das Einstellungsfenster und die Funktionen für die Funktionsfähigkeit der gewählten Einstellungen gespeichert.
- splashScreen: dies enthält den Ladebildschirm und sorgt für dessen Aufruf.
- stylesheets: beinhaltet die Markierung und Hervorhebung des gerade ausgewählten im Elementmenü.
- toolbox: gestattet Funktionen wie das Zeichne der Bauteile, den Error Handler sowie Funktionen, die das Laden und Speichern ermöglichen.

5 Protokolle

Protokoll

30.09.2016

Programmiersprache

Vorschläge:

- Java (FX/Swing) für Oberfläche und Programmierung
-

Oberfläche

Fragen:

- Wer macht die Oberfläche?
- Wer programmiert die Oberfläche?
- Wie machen wir die Oberfläche(3-Schichtarchitektur)

Nachteile an simulator.io:

- Kabel ziehen nur mit Moduswechsel möglich
- Kabel löschen sehr umständlich(Kabel muss nachgefahren werden)

Nächstes Treffen:

- am 07.10.2016 ab 13 Uhr!!!!

Protokoll

13.10.2016

- Programmstruktur gut dokumentieren, damit ausbaufähig für nachfolgende Gruppen
- Einigung auf Java als Programmiersprache und JavaFX für die Oberfläche

Grundziel:

Programmierung eines Digitalsimulators der die elementaren Grundbausteine der Digitaltechnik simulieren kann und erweiterbar in seinen Simulationsmöglichkeiten ist!

Nächstes Treffen:

- am 14.10.2016 ab 13 Uhr

Ziele bis zum nächsten Mal:

- Jeder entwirft seine Vorstellung für die Oberfläche und präsentiert diesen Entwurf beim nächsten Treffen

Protokoll 19.10.2016

- Oberfläche gefällt Herr Würz
- Bausteine durch Mausklick auswählen, durch weiteren Mausklick setzen

Dokumentation:

1. Seminarkurs kurz erklären
2. Programmvorstellung (Aussehen, Möglichkeiten)
3. Entwicklungsumgebung (Java, …)
4. Strukturen erklären (OOP, Probleme -> Lösung)
5. Tätigkeitserklärung
6. Eigenständigkeitserklärung

Nächstes Treffen:

- Spätestens Freitag nach den Herbstferien

Ziele bis zum nächsten Mal:

- Bauteile auf Oberfläche setzbar
- Kabelverlegen

Protokoll 11.11.2016

Nächstes Treffen:

- Am 25.11.2016

Ziele bis zum nächsten Mal:

- Instruktion aller
- Kabelverbindung

Protokoll

25.11.2016

Nächstes Treffen:

- Am 09.12.2016 um 13 Uhr vor Herr Würz Büro

Ziele bis zum nächsten Mal:

- Start der Dokumentation!!! -> Keine Hilfefunktion
- Hilfefunktion:
  - Camillo
  - Luca
- Pathfinder optimieren
- Möglichkeit Ein- und Ausgänge zu „verneinen“
  (Not, NAND, NOR, etc. wird dann unnötig)

Herr Würz ist grandios begeistert!!!

Eine Dokumentation von allen ist genehmigt von Herr Würz

Aktuelle Meinung ist bei Herr Würz 13-144 Notenpunkte für alle!!! ?Wieso keine 15?

Dokumentation:

1. Formulierung des Seminarkurs (Ziel, Möglichkeiten)
2. Arbeitsphasen (Ideen, Entwicklung, Details, etc.)
3. Code- & Funktionenerklärung an einzelnen Beispielen

Protokoll

16.12.2016

Vorschlag anstatt Libre Office Word Online zu verwenden.

- Einstimmigkeit für Word Online

Nächstes Treffen:

- Am 13.01.2017 oder 20.01.2017

Ziele bis zum nächsten Mal:

- Verbindungen verschiebbar
- Doku erweitern

Tim Faharani früher gegangen!

Protokoll

20.01.2017

Nächstes Treffen:

- Am 17.02.2017

Ziele bis zum nächsten Mal:

- Dokumentation langsam fertigstellen

Protokoll 17.02.2017

Nächstes Treffen:

- Am 10.03.2017

Ziele bis zum nächsten Mal:

- Laden von gespeicherten Dateien
- Absturzfehler finden (CPU-Auslastung zum Teil bei 35 %!!)
- Doku erweitern

Protokoll 10.03.2017

Nächstes Treffen:

- Am 24.03.2017

Ziele bis zum nächsten Mal:

- Zustandsdefinierung(am Kabel)(Einweisung von Elias & Tim)

Protokoll 24.03.2017

Nächstes Treffen:

- Am 28.04.2017

Ziele bis zum nächsten Mal:

- Halbaddierer
- Volladdierer

- Ram ?
- Comperator
- Tastaturfunktionen(Strg, Entf, etc.)

Wunsch von Herr Würz:

- Wenn der Raster der Elemente offen ist, werden die anderen Raster(Erweiterte Bausteine, etc.) an den unteren Rand verschoben.

Protokoll 28.04.2017

Nächstes Treffen:

- Am 12.05.2017

Ziele bis zum nächsten Mal:

- Subcircuit (als letztes großes Ziel)
- Comperator
- Halbaddierer
- Neue Oberfläche (Design von Millo)

6 Abschluss

6.1 Offene Probleme

Bei unserem Projekt gibt es wie beschrieben drei Arten der Verbindung: Eine Verbindung die zwei Ein-Ausgänge von zwei Elementen speichert, eine die eine Verbindung (+ XY-Koordinaten) und ein Ein-Ausgang eines Elementes speichern, sowie eine Verbindung die 2 Verbindungen speichert (mit den jeweiligen XY-Koordinaten). Also Verbindungen zwischen Elementen, zwischen bestehenden Verbindungen und die Kombination.

Erstere stellt überhaupt kein Problem dar, man speichert einfach die jeweiligen Elemente, sowie die zugehörigen Ein-Ausgänge, somit kann man diese einfach laden und erstellen. Verbindungen zwischen Elementen stellen also keinerlei Problem dar!

Komplizierter wird es bei den anderen zwei Typen. Sie werden normal gespeichert, also das jeweilige Element mit Ein-Ausgang oder die jeweilige Verbindungen mit den Koordinaten,

das Problem erfolgt dann beim Laden, denn versucht man eine Verbindung zu laden, welche mit einer anderen Verbindung verbunden ist, welche wiederum von einer nicht geladen Verbindung abhängt, findet man sich in einer endlos Schleife wieder, mehrere Verbindungen können aufgrund fehlender Verbindungen nicht geladen werden, da allerdings jede von den anderen Abhängt ist es unmöglich diese zu laden.

Eine mögliche Lösung wäre, nicht Verbindungen direkt mit Verbindungen zu verbinden, sondern „Knotenpunkte“ mit (unsichtbaren) Ein-Ausgängen einzubauen, diese könnten dann wie Elemente geladen werden.

Leider haben wir nicht genügend Zeit, diese Lösung umzusetzen, da man nicht nur die Knettonpunkte umschreiben müsste, sondern auch alle Verbindungs-Klassen, sowie den Rest des Programms (Oberfläche, Steuerung, Simulation…) darauf anpassen müsste.

6.2 Fazit

Im Laufe dieses Seminarkurses gab es viele unerwartete Probleme. Für die meisten fanden wir eine Lösung und haben somit trotzdem unsere anfangs gesetzten Ziele weit übertroffen. Auch bei den implementierten Bausteinen sind wir über die Grundbausteine (unser erstes Ziel) hinweggekommen und haben einige komplexere Elemente erstellt. Was das Speichern und Laden betrifft, hatten wir viele Probleme, haben es jedoch trotzdem geschafft diese Funktion mit einigen Einschränkungen zu implementieren (siehe Kapitel 5.1). Während unserer Arbeit am Seminarkurs mussten wir uns viel neues Wissen aneignen und haben deshalb auch viel für zukünftige Projekte dazugelernt.

Alles in Allem haben wir einen funktionalen Digitalsimulator geschaffen, der für Einsteiger eine gute Alternative zu dem meist verwendeten Programm „ISIS“ ist, und auch für zukünftige Seminarkurse Raum zum Weiterentwickeln bietet.

7 Quellenverzeichnis

7.1 Internet (Bilder / Internetseiten)

Alle nicht aufgelisteten Abbildungen wurden selbst „aufgenommen“

8 Eidesstattliche Erklärung

Wir erklären, dass wir die Arbeit selbstständig angefertigt und nur die angegebenen Hilfsmittel benutzt haben. Alle Stellen, die dem Wortlaut oder dem Sinn nach deren Werken, gegebenenfalls auch elektronischen Medien, entnommen sind, sind von uns durch Angabe der Quelle als Entlehnung kenntlich gemacht. Dies gilt auch für Zeitungen, Skizzen, Bilder und andere visuelle Darstellungen.

Ettlingen, 3. Juli 2017