Universitat Oberta de Catalunya

An open source browser-based software tool for graph drawing and visualisation

Master Thesis

submittet to the Department of IT, Multimedia

and Telecommunications of the

Universitat Oberta de Catalunya

in partial fulfillment of the requirements

for the academic degree of

Master in Free Software

by

Veit-Dieter VOGT

Supervisor / Tutor
Prof. C. Garrigues Olivella

2014

<u>Declaration</u>

Hereby I certify that I have conducted this Master Thesis independently and used no other than the specified resources.

Veit-Dieter Vogt

Table of Contents

Table of Figures

List of Tables

Abstract

One objective of this master thesis was to find and evaluate open source tools or libraries for graph visualisation which are qualified for educational purposes. This tool should be capable for a prospective graph theory course.
The open source libraries have to support graph drawing and visualisation and can run in a browser. First there was a comprehensive search where a lot of libraries were found. Not all of them are under an open source licence, these were filtered off. Secondly these libraries which are written in a computer language which can not run in a browser were separated. These, written in Javascript, which remain subsequently were evaluated to find out which one is the best for this task. The result was that d3.js is that library which has the greatest functional range, best flexibility and could be easily customised.
The second intension was to develop an open source software tool where d3.js was included as graph drawing and visualisation library. The software was written in Javascript so that it can run browser-based. With this tool the students ought to do their exercises and homework. In order not to begin from scratch and develop the entire software, tools and work which are already had been developed and which can be found in the internet was used and integrated. But there was one essential condition to acquit: These tools or software snippets must be under an open source licence and all the licences must be compatible!
Finaly the graph drawing and visualisation editor was presented.

1 Introduction

Graph visualisation is an interesting field of mathematics. With graph visualisation one can show the relationships among the elements of interest. These relationships can be unidirectional or bidiredional, which means that these relations can have one way from one element to an other only, or the elements have interrelations. It is not only of academic interest, but also referes to many aspects of our real-life briefly. With graph visualisation scientists can show the manifold links among users of social networks, clarify the connections of PCs, servers and other network devices among a computer network, or even the internet, illustrate the biological structures of a united cell structure, point up the neural structure of a brain or biological formation. Software dependencies of a complex software meshwork could be elucidated. Graph visualisation can also show the organisational hierarchies in human and animalistic societies, as there are enterprises, residents of a city or an entire nation and of a flock or a swarm. There are many other fields of application in industry and science.
Since Big Data is a buzzword in IT, data visualisation is more necessary than before. Today it is no problem to aquire such an amount of data that humans can not interprete these data without visualisation. Graph visualisation plays a major role in this context.

One intension of this master thesis was to find and evaluate open source tools or libraries for graph visualisation which are qualified for educational purposes, i.e. for teaching a graph theory course. After a comprehensive search these libraries which do not fit the requirements are filtered off. Adjacent the remaining

libraries were evaluated which one is the best for a graph drawing and visualidation tool.
The second goal was to develop a software tool for the intended graph theory course which can run in a browser. The students ought to download this tool from the UOC's virtual campus. With this software tool they will than do their exercises and homework.
In order not to begin from scratch and develop the entire software, tools and work which have already been developed and which can be found in the internet was used and integrated. But there was one essential condition to acquit: These tools or software snippets must be under an open source licence and all the licences must be compatible!

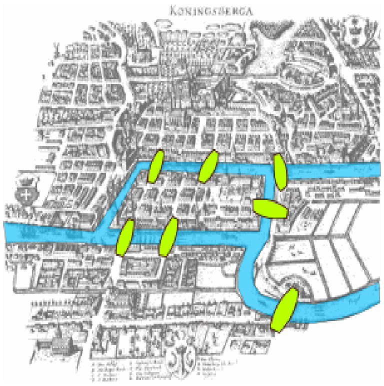
2 Graph Drawing and Visualisation

A Review of the current research

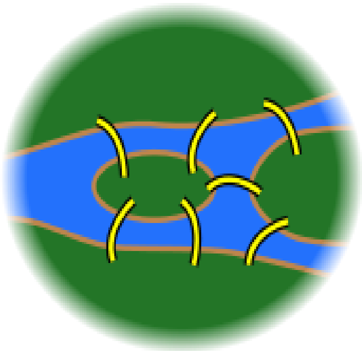
Picture 1: Portrait of Leonard Euler by Emmanuel Handmann 1753

Around 1736 some citizen of Königsberg (today Kaliningrad) asked whether it was possible to make a city tour by using every bridge exactly one time only. Königsberg at that time had 7 bridges. The mathematician Leonard Euler (1707 – 1783) attented to this question. Euler gave some thoughts to this problem and established hereby the mathematical section of graph theory. He took a city map of Königsberg, marked the bridges and then he drew an abstracted map of the city and at the end he drew points and lines to finish the abstraction. He figured out, that for a solution every point must have an equal number of incoming and outgoing ways. Which means that the number of connections of a point must be even. But some of the points have odd connections so that his definite answer was that it is not possible to take this city tour.

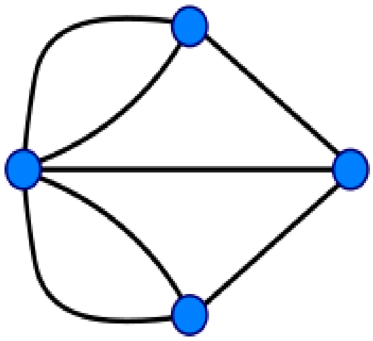
Picture 2: City Map of Königsberg



Picture 3: Marked Bridges



Picture 4: First Step of Abstraction



Picture 5: Second Step of Abstraction

The pictures 2 to 5 illustrates Euler's way of abstraction of the Königsberg bridges problem. (All pictures taken from en.wikipedia.org)
Today these points and lines are called nodes and edges. Graph theory developed in the last centuries to a veritable research field in mathematics. With the increasing number of nodes and edges in a graph it was more and more unreal to understand a graph without visualising them, because humans do understand things better if they can see them. By graph visualisation it is pos-

sible to show the different relationships among the elements of interest. These relationships can be unidirectional or bidirectional, which means that the relations can be one-way or see-saw.
Graph visualisation today is not only of academic interest. With graph visualisation you can show the manifold links among people using social networks, or it could clarify the connections of computers, servers and other devices in a computer network, or the internet in total. Graph visualisation illustrates the biological structures of a united cell structure, points up the neural structure of a brain, or a biological formation. Graph visualisation elucidates software dependencies of a complex software meshwork, or the line drawings of electronic circuits. Graph visualisation can also show the organisational hierarchies in human and animalistic societies, enterprises, residents of a city or an entire nation and of a flock or a swarm.

The research field of graph drawing and visualisation is separated into several section problems:

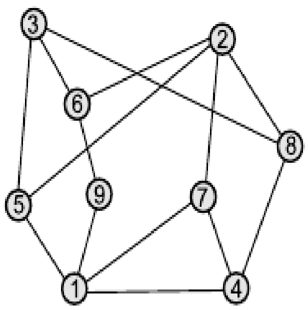
- planarity testing and embedding
- crossings and planarisation
- symmetric graph drawing
- proximity drawing
- tree drawing algorithms
- planar straight-line drawing algorithms
- planar orthogonal and polyline drawing algorithms
- spine and radial drawing

- circular drawing algorithms
- rectangular drawing algorithms
- simultaneous embedding of planar graphs
- force-directed drawing algorithms
- hierarchical drawing algorithms
- three-dimensional drawing algorithms and
- labeling algorithms.

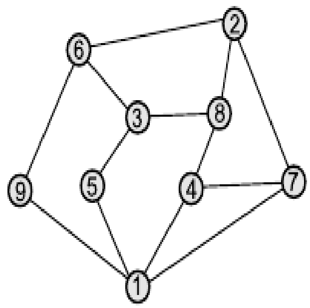
The following review will report the current research of graph drawing and visualisation and will reveal open questions in the sections.

Planarity testing and embedding

The characterisation of a planar graph was written in the 1930s [Kur30], but it took a long time since 1970 [HT74] to find a linear-time solution. Why are planar graphs so interesting? They have several interesting properties. Planar graphs are sparce and 4-colorable [AH77]. Their inner structure can be described very easy and the less crossings a graph has the better is its readability for humans. Therefore the testing of planarity of a graph is an interesting research field. There are some algorithms for testing the planarity. All linear-time algorithms fall into two categories. One category is called cycle based and the other one vertex addition algorithm. The cylce based technique is that a cycle splits the graph into two sections, an inner and an outer section. The technique of the vertex addition algorithms is to beginn with smaller planar graphs and than adding vertices to build the final graph [HT08]. Depth First Search (DFS) is a technique which is common to all planarity testing methodes. It is a special method to vistit all vertices of a graph in specified order. Recognising a planar graph is a main problem. Is a graph certified as non-planar it is called a Kuratowski subgraph isolation [CMS08]. Scientists have developed dynamic algorithms to determine the planarity of a graph (Lempel-Even-Cederbaum [LEC67], Shih-Hsu [SH99] and Boyer-Myrvold [BM04]). Computing planar embeddings means that vertices and edges are added or deleted to construct the final graph [DBTV01].



Picture 6: Non-planar Graph



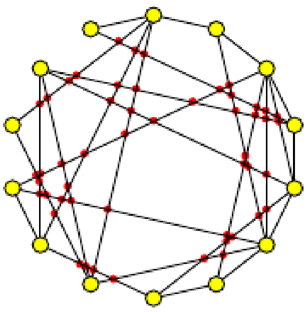
Picture 7: Planar Graph

In planarity testing there are some constraints in simultaneous planarity testing, clustered planarity testing and decomposition-based planarity testing.

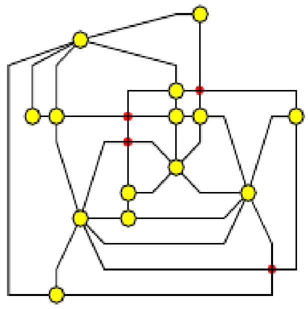
Crossings and planarisation

Another problem is that the bigger the graph and the more crossings it has the lesser is the readability. Crossing minimisation is an optimisation problem in graph drawing. This problem was first examined by Turan during World War II. There are some algorithms which try to solve the problem, but it is still an open problem. Crossing minimisation is as well of commercial interest. In VLSI (very large scale integration) design it is necessary to have as less crossings as possible. More wire crossings imply more costs of the chip. The planarisation method [BTT84] is the current approach to the crossing minimisation problem.

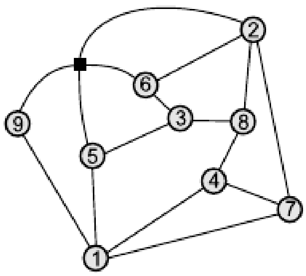
This method consists of two steps. The first step is to draw a planar subgraph which has as many edges as possible. All edges which are not contained in this drawing will then be inserted in the next step. Whenever a crossing is produced, a dummy vertex will be inserted to eliminate the crossing. After all edges have been inserted a planar drawing algorithm will than compute the layout. At the end the dummy vertices will be deleted.



Picture 8: Graph with many crossings



Picture 9: Graph with less crossings

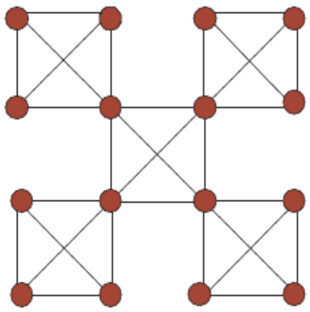


Picture 10: Planarised Graph

One of the open problems in crossing minimisation is the determination of the crossing numbers of the final graph [PT00]. Another problem is the approximability of crossing minimisation.

Symmetric graph drawing

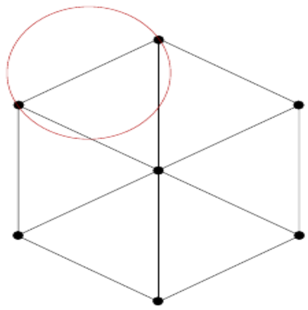
Drawing a graph symmetrically is in most cases preferred by humans over a planar graph drawing [KK89]. Therefore symmetry graph drawing is another important section in graph drawing and is based on fundamental aesthetic criteria. The goal is it to find besides the trivial symmetry a non-trivial symmetry of a graph [CLY01]. Symmetries are related to the automorphisms of a graph. A symmetry can be rotational or reflective or a combination of both. An automorphism group of a graph defines these combinatorial symmetries. But not every automorphism can be drawn as a symmetric graph. A goal of symmetric graph drawing is to determine the automorphisms of a graph which can be drawn symmetricaly.


Picture 11: Symmetric Graph

Drawing a graph „very very symmetric" is one of the open problems. If there are two drawings of a graph with the same symmetry that one is preferred which has the more elaborated symmetry but the scientists are far away from designing an algorithm to draw a graph „very very symmetric".

Proximity drawing

Proximity graph drawing is another question in graph visualisation. A geometric graph which is drawn straight-line and build by a group of nodes where pairs of nodes are connected is called a proximity graph when they have a defined proximity. Depending on the definition of closeness, the same set of nodes could lead to a variety of proximity graphs. Motivated by numerous scientific applications [GO04] [Tou05] [CPZ04] scientists try to efficiently compute different types of proximity graphs of a given set of nodes. The goal is to design an efficient algorithm for computing a proximity graph drawing which is called the proximity drawability problem.

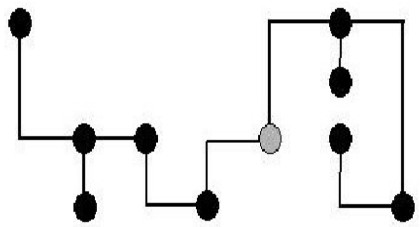


Picture 12: Proximity Drawing

Research area of proximity graph drawing is far from solving the problems. There are many questions which are unanswered. For example: minimum weight drawings [MS92], Delaunay and Voronoi drawings [SIII00], ß-drawings [Rad88], sphere of influence drawings [HJLM93], rectangle of influence drawings [LLMW98] and other proximity rules. Today this section of proximity graph drawing follows two research directions: One is the question of proximity drawings and ad-hoc networks, and the other question is that of proximity drawings and geometric checkers.

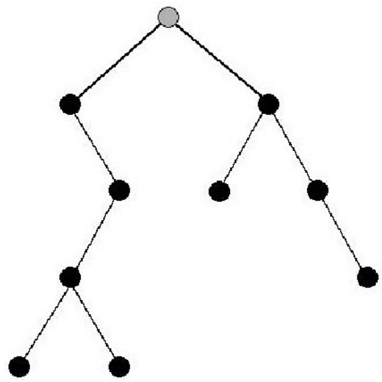
Tree drawing algorithms

For the purposes of representation of relational informations a tree drawing of a graph is suited. In such a tree a node represents an entity and an edge that of the association between the entities. These hierarchical informations could be a program nesting tree, an organisation chart, an activity tree, a knowledge-representation isa hierarchy, a structure of a website, an evolutionary tree, a molecular drawing, or indexes of databases. Typically an algorithm for tree



Picture 13: Graph drawn as Tree

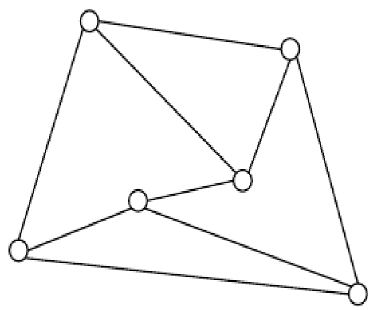
drawing is based on the understanding of the structure of the tree. Also aes-tethic aspects are important for tree drawing because the readability and under-standebility depends on an aes-thetical drawing. Several types of approaches are used to descibe the different hierarchical informa-tions. They are the level-based approach [BJL02], the path-based approach [GR03a], the ring circular layout approach [GADM04] and the separation-based approach [RS07]. There are special algorithms to draw binary trees too. A binary tree is a tree where every node has one or two children only [Mac03].



Picture 14: The same graph drawn in an other manner

Planar straight-line drawing algorithms

Already in the 1930s investigations had been under-taken to do planar straight-line drawings [SR34]. The results had shown, that every planar graph allows a planar straight-line drawing. Today there are two major algorithms to draw graphs straight-line: The shift method [dFPP90] and the realizer method [Sch90], an improved method of realizer was intro-duced later [BFM07].



Picture 15: Straight-line graph drawing

Planar orthogonal and polyline drawing algorithms

Drawing a planar orthogonal graph has the advantage that the graph has no crossings and the angles between the edges are 90° or 180°. But on the other side it also has the disadvantage that the graph can have a degree of a maximum of 4 only. Drawing a graph planar orthogonal is of industrial interest because VLSI design uses planar orthogonal graph drawing for the routing of the wires on a chip. More general are the polyline drawings of a graph. In this kind of drawing the edges can have angles of 45° and the multiple. These graphs than can have degrees of more than 4. There are different algorithms to draw an planar orthogonal graph: The network flow technique [GT02] and the mixed-model algorithm [GM98].

Spine and radial drawing

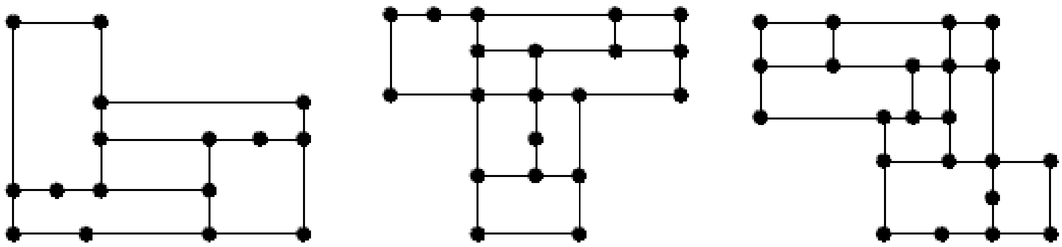
Directed graphs which edges are parallel straight lines are called spine drawings of graphs. Drawings of directed graphs where the edges are concentric circles are called radial drawings. These two kinds of graph drawings belong to the family of layered graph drawing. One of the problems is the point-set embeddability which is investigated in computational geometry [Bos02]. Also not solved are some theoretical connetions between spine and radial drawings. Problems in graph theory and compu-tational geometry are unanswered too [DDLW05] [Sug02].

Circular drawing algorithms

If the following three conditions are fulfilled, a graph drawing is called a circular graph drawing: a. the graph is partitioned into clusters, b. the nodes of each cluster are placed onto the circumference of an embedding circle and c. each edge is drawn as a straight line. For the calculation of circular graph drawings there are two efficient algorithms [Bra97] [DMM97]. The scientists could show that these algorithms work very well in applied practices and produce drawings with a low number of edge crossings [ST06].

Rectangular drawing algorithms

A graph where the vertices are drawn as points and the edges are drawn as horizontal or vertical lines only and the graph is a planar graph, than this graph drawing is called a rectangular graph drawing. This drawing has practical applications in VLSI floorplannings like chip design or architecture [NR04].



Picture 16: L-, T- and Z-shape rectagular graph drawings

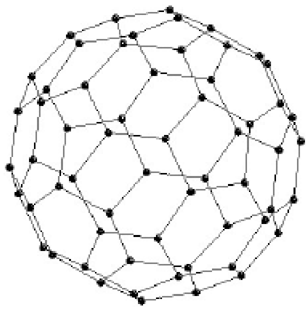
On a chip with sections which produces heat these sections should not be adjacent so that the heat dissipation can work sufficently [She95]. Another practical application is the floorplanning of buildings. For example should the cafeteria not be adjacent to laboratories where it is deald with poisonous chemicals [FW74]. There are several drawing algorithms in practice which deliver good results [KH97] [LL90].

Simultaneous embedding of planar graphs

In cases where not only one graph could explain the relations between the vertices there are two or more graphs which share some or even all vertices needed to describe the situation. This is called a simultaneous embedding of planar graphs. The question than is how could these graphs be best displayed. Because there are various scenarios which need different layouts not only one algorithm could satisfy all particular layouts. For a proper layout the scientists have to take into account the readability and the mental map preservation. These two criteria are often contradictory. The readability is an aspect of aestethic, e. g. the minimal number of crossings. The mental map preservation could be achieved by keeping the vertices of consecutive graphs at the same position. Optimising the one is to downgrade the other. It is essential to keep a balance. Simultaneous embeddings of graphs have much applications in sciences: software engineering, databases, or social networks. Also in industry the simultaneous graph embeddings are of interest. VLSI design uses such algorithms to solve the optimisation problem of chip placement [MOS98]. In this section there are many open problems.

Force-directed drawing algorithms

If there are forces between the vertices of a graph it is to bespoken of force-directed graphs. These forces could be repulsive or attractive between vertices which are adjacent.



Picture 17: Force-directed graph drawing

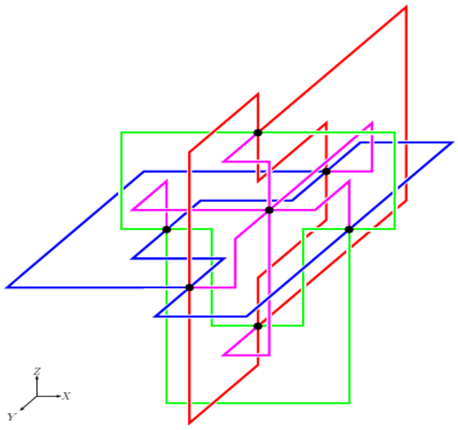
Force-directed algorithms have a long history in this research field. Already in the 1960s algorithms to calculate force-directed graph drawings were introduced [Tut63]. In the last years there had been good progress and many improved algorithms had been implemented [Ead84] [FR91]. But these algorithms work good for graphs with only a few vertices but not if the graphs have hundreds or even thousands of vertices. One of the obstacle is that the physical models have more than one minimum and even with sophisticated mechanisms it is not possible to get good layouts for bigger graphs. Since the late 1990s there had been introduced better algorithms which now can handle graphs with thousands of vertices. These layouts consist of a series of simple graphs so that the readability of the whole composition is acceptable [Wal03] [GGK04]. Also some algorithms do not use Euclidean geometry, but the surface of a sphere or a torus [KW05]. The newer scalable algorithms which can handle large dynamic graphs with thousands of vertices are used in many applications [Mun97].

Hierarchical drawing algorithms

A special case of a directed graph is a hierarchical graph drawing. Examples for hierarchical graphs are the organisation of an enterprise, function calls of a software, object-oriented class diagrams or a PERT chart of a project management. The major algorithm for drawing hierarchical graphs is the Sugiyama method [STT81]. In the last years there had been many modifications and enhancements of this framework [dNE02] [SM95a] [UBSE98]. But all these alternatives have their special applications and limitations. There are attempts to draw hierarchical graphs in 3D [GT97] [HN05a]. One of several approaches to overcome these limitations is the UPL system [CGMW11]. So-called radial level drawings are another alternative method for visualisation of a social network [BKW03]. The cycle style of drawing was also introduced to avoid the top-to-bottom leveling [BBBL08].

Three-dimensional drawing algorithms

In most cases it is sufficient to draw a graph in two dimensions. But for special applications a drawing in three dimensions is better. These applications could be VLSI design [LR86], software engineering [WHF93] or information visualisation [WM08].



Picture 18: 3D graph drawing

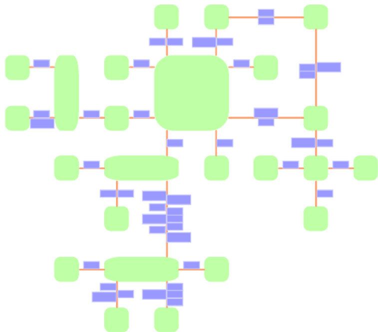
3D drawings have made great advantages in the last years by better hardware and increasing computer power. So-called grid-drawings, that are graph drawings where the vertices have integer co-ordinates only, ensure a minimum of grid spacing and the readability is better if the nodes would be too adjacent. Another aspect of 3D drawing is straight-line crossing-free graph drawing. Orthogonal drawings have edges which are parallel to one of the axis and they guarantee a good angular resolution. In VLSI design it is important to minimise the space for the chip to avoid dissipation of space. There are still many open problems in 3D graph drawing but with increasing computer power the scientists had made good progress in the last decades.

Labeling algorithms

Automated labeling edges and nodes is a major problem in graph drawing. Labeling has for example applications in cartography [RMM+95] and geographic information systems (GIS) [Fre91].

In cartography the labeling is elevated into an art over the decades and automatic labeling never will reach a sufficent placement.

But in some cases like real-time placement in on-line GIS, oil exploration [Zor90] or internet-based mapping it has acceptable qualitiy. At present time the semi-automated labeling systems are the best approach. They produce an initial labeling and adjacent it will be improved manually. By increasing computer capabilities the automated labeling will make further advances.



Picture 19: Labeled Graph

Conclusion

In graph drawing and visualisation the scientists have made good progress in the last decades. The theoretical basics in information technology has been augmented in the last sixty years. Some new languages in artificial intelligence have been developed to better support the implementation of algorithms for graph drawings. The efficiency of the algorithms has been improved over the time and the computational capabilities progressed with seven-league boots. But still there are several questions without an answer. The research field of graph drawing and visualisation is still an interesting field of research and will show some appealing and instructive changes over the next years.

Note: All pictures in this chapter are taken from [Tam13]

3 Methodology

In this Master Thesis we searched for documents about the libraries, evaluate them and decided which of them is the best for developing our tool. This first part is a document approach.

The second part of the research work is a design and creation approach. Here we develop a tool for the intended graph theory course of the UOC.

Stol and Babar [SB10] listed 20 Open Source Software evaluation methods. Some of them are industry driven and do not cover the intentions for the special purpose here. Others are very comprehensive and sophisticated; these are not suited to be taken into account in this research work because it would take too long to get familiar with these methods. Among the remaining there are the papers of Cruz, Wieland and Ziegler [CWZ06], the paper QSOS initiated by Atos Origin [Atos13] and the online paper of David Wheeler [Whe11] which describe methods for evaluations of free software / open source software. A first glance at these three papers showed that many points of the different methods are similar to the method described by D. Wheeler. All three methods look very much alike and only differ in some special points. The QSOS method (see www.qsos.org) is under an open source licence and provides a plug-in for Fire-fox. This tool could ease and facilitate the evaluation process.

[ADDENDUM: End of May the European Commission anounced the open source project OSSmeter (https://joinup.ec.europa.eu/elibrary/case/ossmeter-platform-automatically-assess-monitor-and-compare-oss-packages). From the description: “Evaluating whether an open source software package meets the requirements for a specific application, or determining the best match from a list

of packages, requires information on both the quality and the maturity of the software, as well as understanding whether the software is continuing to evolve and if there is a substantial and active community of users and developers......"
For this research work it is too late to take this project into account and include more information, but furhter research should not disregard it.]
Our decision therefore was to evaluate the software according D. Wheeler's IRCA method. For a detailed description see below. Regrettably we have not found any papers where researchers report that they had used Wheeler's method and we only found two papers where researchers describe their evaluation process of open source software. These are the papers of Graf and List: An Evaluation of Open Source E-Learning Platforms Stressing Adaptation Issues [GL05] and of Fleischfresser: Evaluation von Open Source Projekten: Ein GQM-basierter Ansatz [Flei07]. Due to this low number of references it is not possible to speak of a state-of-the-art in this research field.
David A. Wheeler elaboratedly described in his paper "How to Evaluate Open Source Software / Free Software Programs." a general 4-step process for evaluating programs. He calls this method "**IRCA**". IRCA is a short form of: **I**dentify the candidtes, **R**ead existing Reviews, **C**ompare the leading programs and **A**nalyse the top candidates in more depth. This method will be used to evaluate the open source candidates for graph visualisation. In this paper Wheeler described 14 criteria of evaluation. Not all of them may be relevant for UOC to use the open source software / libraries in a graph theory course. In this paper he gives specific informations on how to evaluate open source software / free software (OSS/FS). Wheeler developed this process so that anyone can compare OSS/FS side-by-side with proprietary software and determine which of the can-

didates best meets one's needs. After an introduction about open source software, other approaches for evaluation and a short overview over the IRCA process, he goes into depth in the following four chapters. In chapter two he declares the process step of identifying the candidates and gives some tips. In chapter three he recommends to read reviews about the candidates. Briefly compare the leading program's attributes to your needs is the title of chapter four. This chapter is very detailed. In 14 subchapters he describes important attributes to be considered including functionality, cost, market share, support, maintenance, reliability, performance, scaleability, useability, security, flexibility / customisability, interoperability, and legal / licence issues. The last step in his method is to perform an in-depth analysis of the top candidates which the author discusses in chapter 5. Chaptert 6 terminates Wheeler's paper. Therein he recommends some hints how to present the results of the evaluation.

3.1 In-detail description of the data analysis

David Wheeler called his method the IRCA method. IRCA is short for the four steps of which the method consists:

1. Identify the candidates
2. Read existing Reviews
3. Compare the programs' basic attributes and
4. Analyse the top candidates in depth.

But before the researcher begins with step one, it has to be clear what requirements the candidates must comply with. Wheeler strongly recommends that the researcher who evaluates free software / open source software according his method must have a basic idea of what his needs are, i.e. the researcher should have a list where all the requirements of the candidates are listed and according to which the search and evaluation will be geared to.

3.1.1 Identifying the candidates

In this first step the researcher has to do a comprehensive search, preferably in the internet, to find as many candidates as possible which roughly fulfill the requirements. Wheeler had listed some recommendations which can be an assistance for this search, e.g. a paper of the U.S. government and lists of so-called GRAM and GRAS (GRAM = generally recognised as mature) (GRAS = generally recognised as safe) software. At the end the researcher has to deselect the software which is not adequate, so that only these candidates remain, which absolutely accomplish with the researcher's needs.

3.1.2 Reading existing reviews

Wheeler's references are to visit the programs website, to read software comparisons to find the strengths and weaknesses of the candidates and he gives some examples. In our case where we search for software tools and libraries which can run in a browser and which have to be under an open source licence there are no such reviews to find. The only comments about the candidates which we had found are these in the forums. But these comments do not have any scientific explanatory power.

3.1.3 Comparing the leading programs

After the candidates that do not fulfill the requirements have been eliminated the next step is a comparison. For this comparison, step three of the IRCA method, Wheeler recommends to visit the projects web page. Most open source software projects have such a web page where most of the informations about the project can be found. There you will find the documentation of the software, FAQs, mailing lists on which the researcher could subscribe to receive an impression on “what is going on” in the software project and much more. Wheeler lists here 13 facts to take into account for this comparison: functionality, cost, market share, support, maintenance, reliability, performance, scalability, usability, security, exibility/customisability, interoperability and legal / licence issues. Not all of them are relevant for my research work. Costs, market share, support, maintenance and scalability are of less or even no importance for this special comparison. The attribute legal / licence issues were a prerequirement to fulfill, a candidate which does not fit this KO criterion does not remain on the list at this stadium. The other attributes: functionality, reliability, performance, usability, security, exibility / customisability, interoperability will now be examined in more depth. Therefore all necessary informations about the candidates will be accumulated wherever it can be found resulting in a matrix which shows the most important functionalities and features. The result of this matrix is that some of the candidates will be cancelled, e.g. because of less features, or other weaknesses.

3.1.4 In-depth analysis of the top candidates

This fourth and last step of Wheeler's IRCA method is the most important step because eventually a decision has to be made on which of the candidates fulfill the requirements best for the intended graph theory course. For this step Wheeler recommends to have a second but more in-depth comparison as in the previous step. He also recommends a try-out of the software. For our research work the functionality and the customisability are very important attributes. To get these informations is fundamental. In this step we had a look for special features which are important for the graph theory course: Labels, graph manipulation and graph drawing algorithms. At the end of this ultimate decision we chose the d3.js library, because this library is the most evolved one among the candidates, it has the greatest functionality and its customisability is best.

4 I R C A

In this research work we had to find and compare libraries which can run with a web-based framework in a browser.
These libraries have to be under an open source licence.
For this comparison we will operate like D. Wheeler suggested in his paper: „How to evaluate Open Source Software / Free Software (OSS/FS) Programs.“ [Whe11].

4.1 Identify

Our first search quarried numerous libraries from which we firstly sorted out those which are under proprietary licences.
Ten libraries which are under open source licences are left then.
The first six are written in Javascript and they can run in a browser.
Igraph is written in C, neo4j in Java and OGDF and PIGALE in C++, they can not run in a browser, so that we did not investigate them any further.
The remaining libraries we subjected to a comprehensive comparison.
In Table 1 there are listed the libraries and some informations about them.

Library	URL	Language	Licence	Latest version
D3.js	d3js.org	Javascript	BSD-3clauses	3.4.1
graphdracula	graphdracula.net	Javascript	MIT	0.0.3alpha5
jointjs	jointjs.com	Javascript	MPLv2	0.8.0
jsdot	code.google.com/p/jsdot	Javascript	MIT	0.9
jsplumb	jsplumbtoolkit.com	Javascript	MIT/GPLv2	1.5.5
Sigma.js	sigmajs.org	Javascript	MIT	1.0.0
igraph	igraph.sourceforge.net	C	GPLv2	0.6.5
neo4j	neo4j.org	Java	GPL, but community edition only	2.0.0
OGDF	ogdf.net	C++	GPL	2012.7
PIGALE	pigale.sourceforge.net	C++	GPL	1.3.15

Table 1: Identification of the Libraries

4.2 Read Reviews

D3.js

Searching the internet for „d3.js review“ lead to more than 4.400.000 hits. This first impression shows the high profile of d3. There are comments to find like: „D3.js is way more than just another visualization framework.“ Many other comments which praise d3 could be found.

Graphdracular

To find reviews about graphdracula is in contrast a difficult task because most hits concern „Graf Dracula“ and not the graph visualisation library. Beside the graphdracula webpage there are almost a dozen pages where the library is the theme.

Jointjs

For jointjs google delivers afterall 92.900 hits. It is a relatively unknown library.

Jsdot

Jsdot, which was a small project at google summer of code in 2009, has just 805 hits. It will not longer be developed further so that we leave it out of deeper investigations.

Jsplumb

Jsplumb has 163.000 hits. The project started in November 2011 and has less than 10 contributors. So we can say that it is a small project and the development seems to be slow.

Sigma.js

Sigma.js is much more known. Google delivers approximately 16.200.000 hits. Sigma.js is a new and lightweight library to draw graphs. The most reviews are warm to sigma.js which had just reached version 1.0.0.

The only comments about the candidates which we found are these in the forums. But these comments do not have any scientific explanatory power. The only conclusion which can be drawn is an qualitative one: Are the voices majoritarian pro, which means the commentator does praise the software due to its manifold features, or the commentator is contra, which means he/she criticises the program due to some bad bugs, lack of features etc. But on the basis of the number of comments it is possible to figure out the popularity of the software: The more positive comments, the greater the popularity. And from the popularity of a software it is legal to conclude to its market share.

4.3 Comparison

4.3.1 Functionality

On https://github.com/mbostock/d3/wiki/API-Reference you can find a very long list of API-References. This list with allmost 620 entries in 39 sections shows all the functions which are included in d3. We did not find an API-Reference for graphdracula. The API-Reference for jointjs is just 77 entries long. This is not that much. At the scantily project webpage of jsdot there is no information about an API-Reference to find. The API list of jsplumb counts almost 300 entries. At sigmas webpage there is no information about an API-Reference.
The functionality of the candidates is very diverse. While jsdot has only basic functions, jsplumb is more developed. Independant of the negative information about the APIs sigma is in the midfield, graphdracula has sophisticated graph drawing functions, but the best functionality by far has d3.js.

4.3.2 Market share

It is hard to evaluate the market share of open source software because there is no vendor which publishes sales volumes. But the numbers of releases and contributors could give a first impression about the popularity and the mightiness of a forum and its activity are indices for the market share of an open source program. In the following list we have summarised some information which we found to indicate the market share.

	d3.js	graph dracula	jointjs	jsdot	jsplumb	sigma.js
Source	github	github	github	project webpage	github	github
developers / contributors	67	6	10	5	7	2
releases						
first	2011 Feb17	n. a.	2011 Feb27	n.a.	2010 Mar15	n.a.
numbers	164	1	7	2	22	1
latest	2014 Jan14	2011 Jun30	2014 Jan20	2009 Dec18	2013 Dec06	2014 Jan30
github downloads	22065	332	641	392	1333	1793

Table 2: Data which indicate the market share of the libraries

D3.js is the project with the most developers / contributors, releases and downloads. This entitles to the assumption that d3.js is that library with the greatest market share.

4.3.3 Support / Maintenance / Longevity / Reliability

An index for the support, maintenance, longevity and reliability can be the ver-

sion number of the program and the activeness of its community. Also whether there are demos / examples and tutorials.
D3.js has reached version 3.4.1 which is an indication for an long development and the maintainer M. Bostock and the large community of contributors show that the development of d3 will go on. There is a great community where questions of users are answered and the further development is discussed. On the webpage there are numerous demos and examles and a lot of tutorials.
Graphdracula is still in the first steps of its development, it has just reached 0.0.3alpha5 and this release is two years old. There is no indication when it will come up to version 1.0 and it is unclear how many contributors carry the development on. There are two demos only and the documentation is in the source code only.
Jointjs also has a low version number: 0.8.0. But there is an active community so that there is a justifiable hope that its development will go on. On their webpage there are 10 demos but the documentation is less.
Jsdot was a litte code project in 2009 and has reached version 0.9 but it will not be developed any further. No demos are to be found on the homepage and the documentation is rather small.
Jsplumb has overcome version number 1 and is now at 1.5.5. The webpage gives the impression of professionality and there is an active community in its forum. Some demos and a substancial documentation is on the webpage.
Sigma.js has just reached version 1.0.0. It is a relatively new library but its popularity is increasing. Sigma is one of several projects of the medialab at Sciences Po in Paris. This leads the hope towards an ongoing development. On sigmas webpage there are only 2 demos and the documentation is not so

much.
In the table below these results are sumed up:

	d3.js	graph dracula	jointjs	jsdot	jsplumb	sigma.js
demos / examples	very much	2	10	no	some	2
tutorials / documen-tation	very much	in code only	yes	less	detailed	less

Table 3: Support information about the libraries

All attempts to get in contact with the project maintainers via eMail asking for commercial support failed. There had been no replies. We gather from that that the projects do not offer commercial support for the libraries. The only support available is that from the forums and from the community.
The quality of support can be infered from the activity of the community.
But to set up a rank order is rarely possible, because it depends on many unpredictable factors.

4.3.4 Flexibility / Customisability

D3.js
On https://github.com/mbostock/d3/wiki/Tutorials there are a lot of tutorials for d3.js on general and special themes, blogs, talks and videos, meetups and publications. If there is a function the user needs, which is not implemented, it is easy to write a plug-in for d3.js. The flexibility and customisability of d3.js is very

extensive.

Graphdracula

On http://www.graphdracula.net/documentation/ you may read about the documentation: „Don’t hesitate to dive into the source code! It is well described with comments and the library archive also includes some example files worth checking out.“ This means in other words: There is no documentation as that which is included in the source code. For graphdracula there are no plugins as it is to read here: „Dracula is a set of tools to display and layout interactive graphs, along with various related algorithms.
No Flash, no Java, no plug-ins. Just plain JavaScript and SVG.“ [see http://www.graphdracula.net] Flexibility and customisability of graphdracula is therefore non-existent.

Jointjs

The documentation and available tutorials keep within a low limit: http://jointjs.com/tutorial. There are just 8 plug-ins listed on the webpage. Flexibility and customisability of jointjs is small.

Jsdot

On the scantily project's webpage there are neither informations about an API-Reference, documentation nor plugins to find. Due to the fact that the project will not be developed further the flexibility and customisability of jsdot is non-existent.

Jsplumb

The documentation on http://jsplumbtoolkit.com/doc/home is very comprehensive. But there is no information about plug-ins. Flexibility and customisability of jsplumb is better than of jointjs but much less than of d3.js.

Sigma.js

On sigmas webpage there are neither informations about an API-Reference nor plugins to find. But github lists 6 plugins. The tutorial is lean. Documentation is one page only: https://github.com/jacomyal/sigma.js/wiki. The flexibility and customisability of sigma is at the time out of recognition.

4.3.5 Licence

All libraries are under an open source licence as was the strict requirement. The following table shows the licences.

	d3.js	graph dracula	jointjs	jsdot	jsplumb	sigma.js
licence	BSD 3 clauses	MIT	MPLv2	MIT	MIT / GPLv2	MIT

Table 4: Licences of the libraries

All licences are compatible with GNU GPLv2 and v3. See table below:

	BSD 3 clauses	MIT *	MPL v2	GPL v2	GPL v3
GPL v2	✔	✔	✔	✔	✔
GPL v3	✔	✔	✔	✔	✔

Table 5: Licence compatibility

The reader may read more information about all licences and the licence texts in Appendices A to F at the end of this document.

4.3.6 Other cirteria

D. Wheeler listed some more criteria in his paper. There are costs, performance, scaleability, useability, security and interoperability. The most of them are not relevant for this research. For instance costs: All libraries are under an open source licence and therefore no licence fee is to pay. Of course there are costs of installation and maintenance of the software but these costs are the same for other (proprietary) software. Other criteria are the performance, the useability and the security of the libraries which are dependent mainly on the program which includes the libraries. In the end there is the interoperability. The program which calls the library will run in a browser. Because it is an open source program only these browsers which are under an open source licence as well will be supported.

*The MIT licence is also known as X licence or as X11 licence.

4.4 Analysis in-depth

All the above listed information about the candidates is of general interest. We will now analyse the candidates for special features which are necessary for graph drawing and visualisation.
First of all there is the question of labels for nodes and edges: Which labeling does the library support? Eliptic, rectangle, polygon drawing of a node and weight for an edge, color and caption for both?
Graph manipulation functionality is the next question: Do the candidates support interactive zooming and interactive node moving? A very important question is the question for the integrated graph algorithms or whether it is possible to integrate graph drawing algorithms by plugins or any other mechanism.
The table below indicates these features.
Only graphdracula has implemented 4 graph drawing algorithms in its code. These algorithms are: Path finding (Dijkstra), shortest path (Bellman-Ford), max-flow-min-cut (Ford-Fulkerson) and string matching (Knuth-Morrison-Pratt).

This list shows the functionality of the candidates. It is clear to see that d3.js has the most functions integrated and graph drawing algorithms can easily be inserted by plug-ins. Here is the web-link for a first step to write plug-ins for d3.js: http://bost.ocks.org/mike/chart.
Graphdracula and jsdot have no flexibility and / or customisability. Jsplumb has a mean flexibility and customisability and that of jointjs and sigma is small.

	d3.js	graph dracula	jointjs	jsdot	jsplumb	sigma.js
Labels						
eliptic	yes	yes	yes	yes	yes	yes
rectangle	yes	yes	yes	yes	yes	no
polygon	yes	yes	yes	yes	no	no
weight	yes	yes	yes	no	no	yes
color	yes	yes	yes	yes	yes	yes
caption	yes	yes	yes	yes	yes	yes
Graph manipulation						
interactive zooming	yes	no	yes	no	yes	no
interactive node moving	yes	no	yes	yes	yes	no
Graph drawing						
number of supported algorithms	support by plug-ins	4 (see text)	n.s.	n.s.	n.s.	n.s.
force directed drawing	yes	yes	yes	no	no	by plug-in
hierarchical drawing	yes	no	yes	no	yes	no
tree drawing	yes	no	yes	no	yes	no
rectangle drawing	yes	n.s.	n.s.	no	no	no
Flexibility / customisa-bility	very great	non-existent	small	non-existent	mean	small

Table 6: Feature list of libraries

4.5 Conclusion

Jsdot's development is discontinued.
Graphdracula is still at the beginning and it is not clear whether it will develop. There is one developer only and it seems that he has discontinued the development.
Jointjs is a small project with just 10 developers and its flexibility is small.
Sigma.js is a new graph drawing project which may have a good future but its current flexibility is small. Probably it will improve.
Jsplumb has also got only 7 developers but it seems the development will improve its features.
Far away from the other candidates is d3.js. Its functionality and flexibility is much better than all the other libraries which we investigated. It is easy to write plug-ins for d3.js so any needed function can be integrated.
D3.js is our favorite to implement as library into the web-based framework.

5 Software Development

To develope software it is necessary to choose a methodical approach.
There are different requirements for this approach:

- structure the development of a model to partial tasks and steps
- use special means of representation
- give a guidance for the realisation of the modeling
- support the quality assurance of the models or includes critetia for „good“ models
- as a general rule no algorithm (in mathematical meaning)
- support the relevant partial models and their integration
- efficiency

In the development of a model the following numerous steps including repetitions and returns are generally used:

- definition of the object to model
- determination of the modeling purpose
- determination of the modeling environment / instruments of characterisation
- appointment of the rules for the abstraction
- determination of the reproduction of the reality to the modeling environment
- testing of the model

There are different views for modeling of information systems:

Function view (also known as process view): Describes the functions and partial functions prescinded from time and sequence.

Sequence view (also known as dynamic view): Describes what activities and processes coincidentaly or consecutively execute prescinded from the semantic of the data.

Data view: Describes the data which is treated or stored during a process or information system prescinded from time and independent of the sequence view.

Object view: Describes a process or an information system as an amount of interacting objects prescinded from sequence, integration of data and function view.

Organisation view: Describes the sequence of an organisational unit which is part of a process or information system. The communication and managerial authority correlations between the units are part of the organisational structure.

Performance view: Describes the results of the process execution by product models.

The purpose of such a model is:

1. To understand the reality (actual condition)
   - to describe the benefit of the assumption
   - to explore the forecast about the behaviour
   - to evaluate / analyse the assignment of faults
2. To construct the reality (target state)

- describing the requirements for the construction
- analysing / simulating the examination of the construction before realisation
- analysing potential / alternative representations of the reality
- co-ordinating the involved persons (customer, construtor, co-ordination between different employees / organisations, which execute the realisation co-operative)

One of these modeling languages for specification, visualisation, construction and documentation of (information) systems is the Unified Modeling Language (UML)

5.1 Unified Modeling Language (UML)

UML is a standard of the OMG (http://www.omg.org/uml) . It is not a method , but a notation and semantics for specification, visualisation, construction and documentation of models for business processes, object orientated software development and other general systems.

Like in all other languages there is a language range that kind what terms are part of the language and what denotation these terms have. This is the same what we know from other languages. But there is an important difference: While in other programing languages key words are real words, UML is a language that consists of simple geometric symbols. In UML there are defined numerous geometric forms like rectangle, arrows and ellipses and there are definitions of the denotation of these forms. UML knows rules how to combine these forms so that they will result in something meaningful. With these models the software developers can inspect a cut of an entire system, which important aspects will be highlighted and which unimportant aspects will be disregarded. This model is not a rebuild of the original, but a simplified description of the original with the goal to better understand the original.

13 types of diagrams are defined in UML. These diagram types are either structure diagrams or behaviour diagrams.
The following six diagram types are structure diagrams:

- classes diagram
- object diagram
- component diagram
- composition structure diagram
- distribution diagram
- packages diagram

Behaviour diagrams are the following seven diagrams:

- use case diagram
- activity diagram
- finite automation
- sequence diagram
- communication diagram
- timing diagram

- interaction diagram

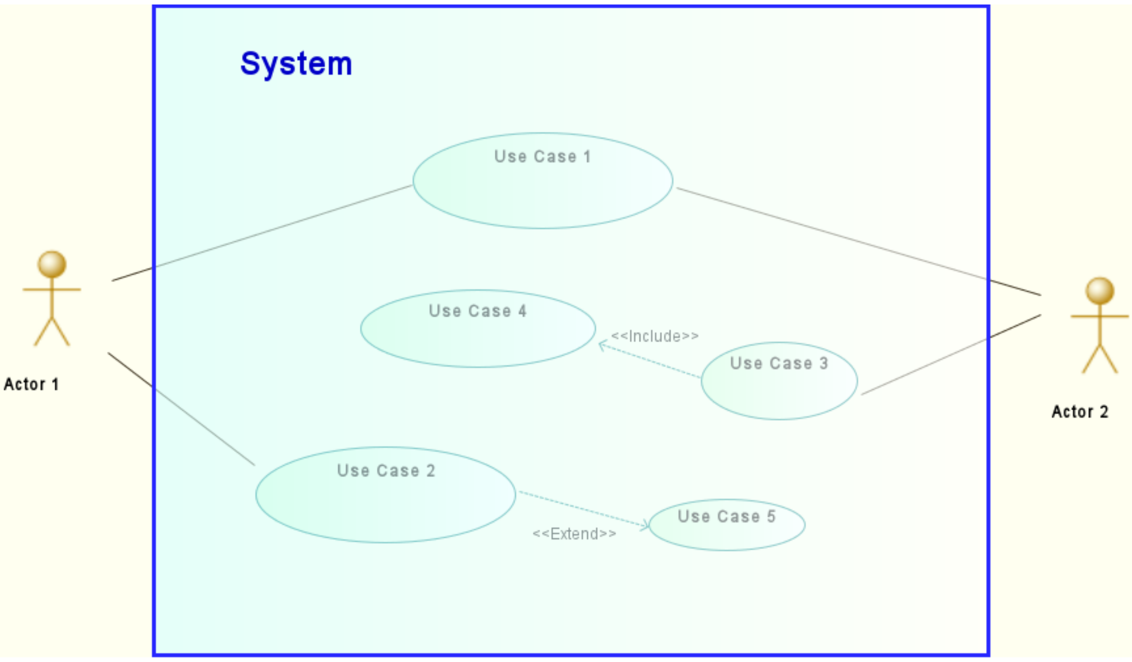
The modeling of software should aid to keep an overview. This diagram types should describe the software from different views. All together they show an overall picture of the software. Models are complied with requirements and are based of elements of the UML. The language range is independent of programing languages and platforms. The built models of software are hence independent of this technologies too.
UML is also suited as co-ordinating instrument. Every developer will understand the UML models. Since UML is a graphical modeling language it is suited to communicate to non-technical persons (management) too.

In the following we will restrict to show the use cases diagram and the sequence diagram of the software tool we are developing.
Because JavaScript does not know the classes concept a classes diagram is not possible.

5.1.1 Use Cases Diagram

A use cases diagram consists of a lot of use cases and describes the relations between actors and use cases. A use cases diagram shows the externally visible behaviour of the system from the view of the users.

Use cases will be represented by ellipses which hold the name of the use case and an amount of joint objects (actors). Any use case will be characterised in text form, more or less detailed. The correspondent use cases and actors will be connected by lines. The system limits are symbolised by a frame. Some use cases include further use cases, or will be extented by others. Example of an use cases diagram:



Picture 20: Example of an Use Cases Diagram

In the software tool which we developed the students are the actors (and the system administrator). The user and graph databases are used for user login and logout and for up- and downloading graphs. Additionally there may be a system administration tool via which the system administrator administrates the system, e.g. the user and the graph databases (this is not part of our software). The use cases of this software are:

- user login (includes the user database)
- editing a graph
- upload a graph (includes the graph database)

- download a graph (includes the graph database)
- system administration
- user logout (includes the user database)

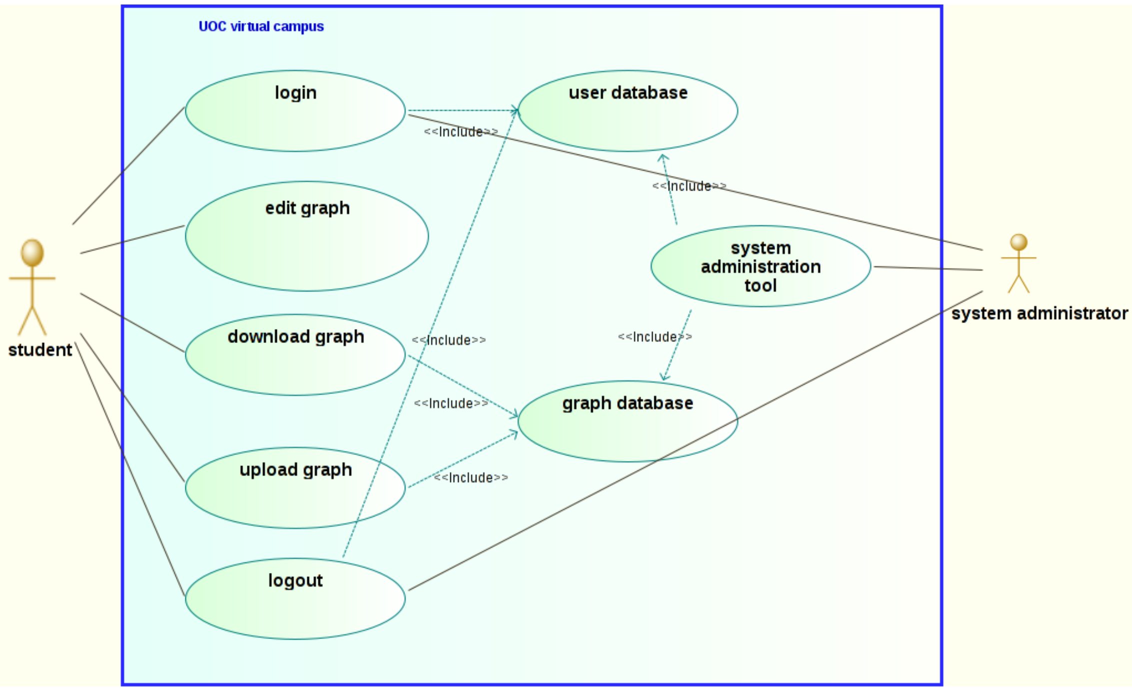
Editing a graph imbeds the following tasks:

Changing the features of nodes and edges like shape, line, text, style, colour and weight.

Nodes can be arranged in force-directed or fixed graph layout.

Documenting a graph includes editing of text discribing a graph and embedding of images.

The use cases diagram is shown here.



Picture 21: Use Cases Diagram of the software tool

5.1.2 Sequence Diagram

A sequence diagram describes the chronology of interactions between a lot of objects within a temporal limited context.

The objects will be visualised by rectangles from which the vertical life lines originate, depicted by dashed lines. The messages will be described by horizontal arrows between the life lines of the objects. At this arrows the messages will be noted in the form: message(arguments). Messages which are answers have the form: answer:=message(). Messages can also be allocated conditions by the form: [condition] message(). Iterations of messages will be depicted by a „*“ before the name of the message. Objects which are just active in interactions will be marked by a bar in the life line.

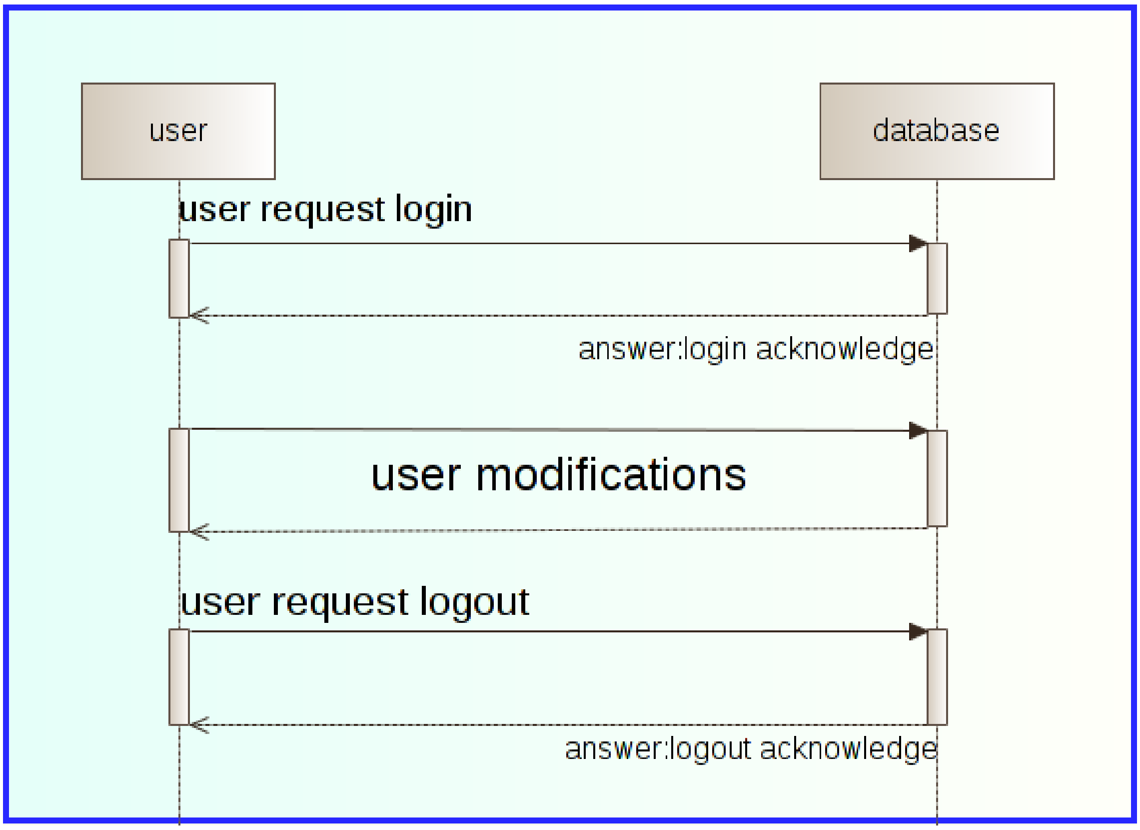
An example is shown on the next page.

In this software there are the students as actors, the user database which stores the allowed users and their passwords, the graph database where is stored an amount of example graphs and the students graphs.

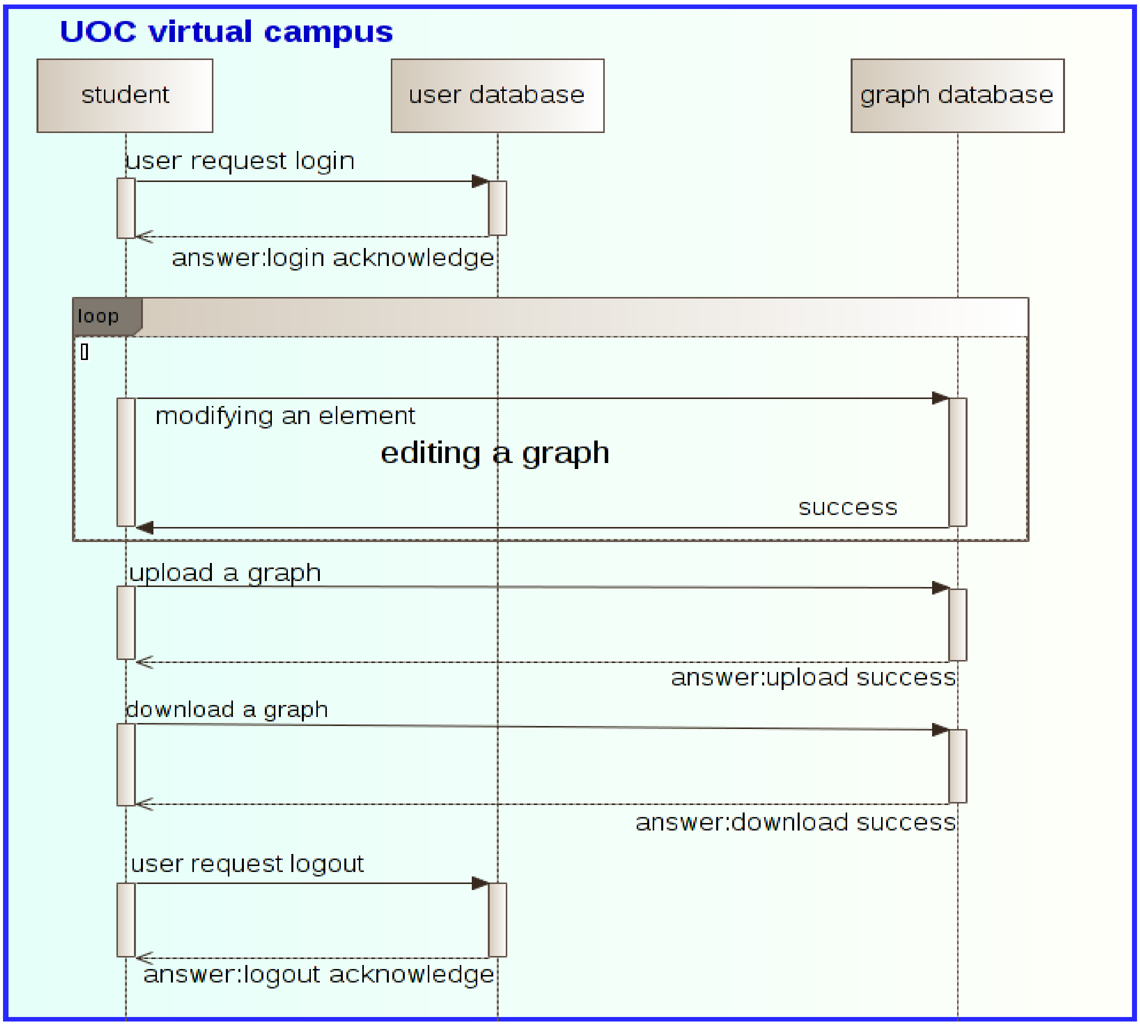
The intercations are:

- user login
- graph editing as loop of numerous minor tasks
- uploading a graph
- downloading a graph
- user logout

The sequence diagram is shown on the nextbut one page.



Picture 22: Example of a Sequence Diagram



Picture 23: Sequence Diagram of the software tool

6 Documention of the software tool

The purpose of the graph tool is:

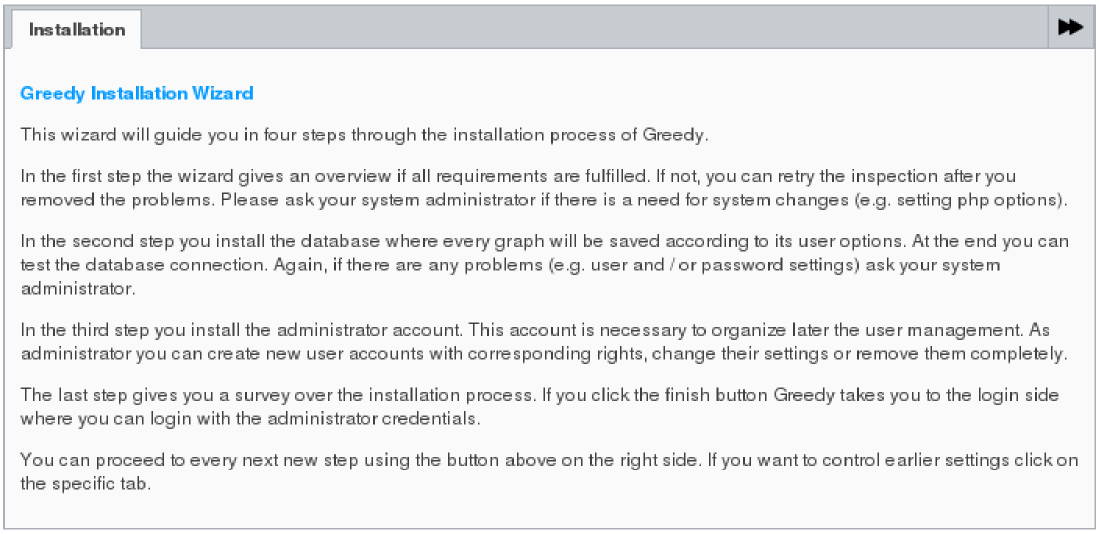
- Drawing and editing graphs
- Representation of graph theoretical structures
- Visual representation of graph algorithms

6.1 Installation in 4 steps

Prerequirements before installation:

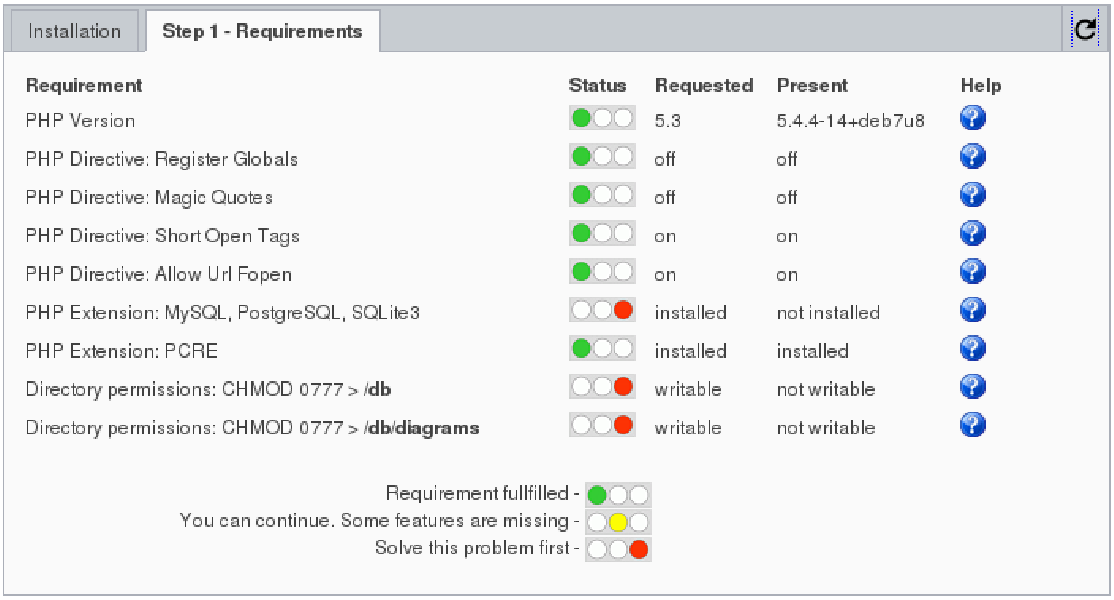
- LAMP (XAMP) has to be installed. Instead of MySQL the user can also install SQLite or PostgreSQL.
- Apache2 must be running.
- Extraction of greedy.tar.gz to /var/www/
- Directory /var/www/greedy, all subdirectories and files has to be owned by www-data or current apache user respectively.
  (command: chown -R www-data:www-data /var/www/greedy)
- The rights of /var/www/greedy/core/db have to be 777. If the users security policies do not allow 777, 755 would be good too.
  (command: chmod -R 777 /var/www/greedy/core/db)

After the user has extracted the files to /var/www/ he has to insert "localhost/greedy" in a browser. The installation process will then start with an initial picture.



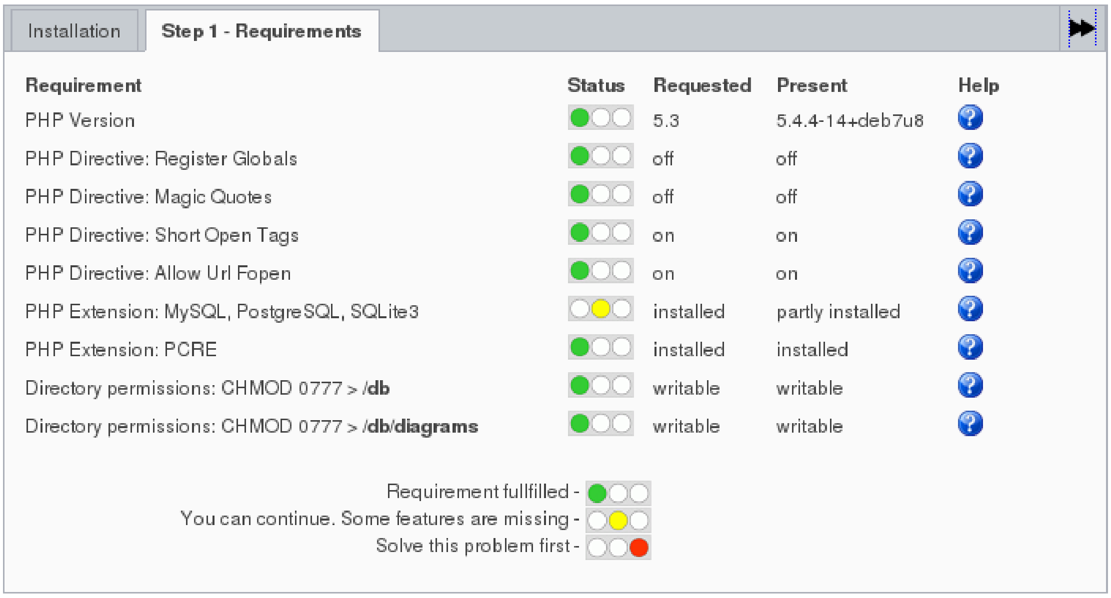
Picture 24: Greedy installation initial picture

He than just has to click on the arrow to proceed. In step 1 all requirements will be scanned. As long as not all requirements are fulfilled the user will see red lights as is presented in the next picture.



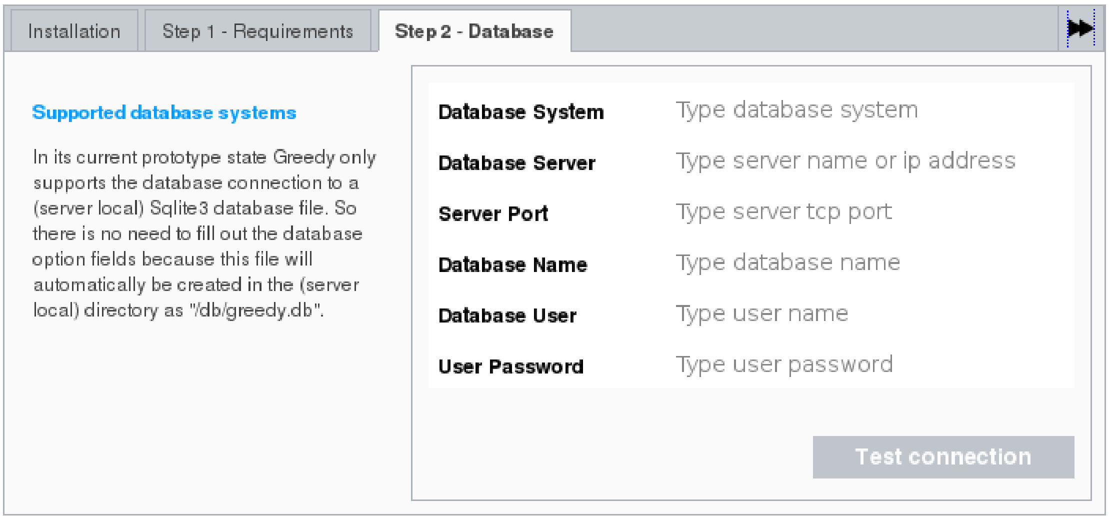
Picture 25: Not all requirements are fulfilled

When there are green lights only as seen in the following picture the user may click to continue. Otherwise the displayed problems nust be resolved. This steps have to be repeated as often as necessary by clicking on the arrow in top right corner.



Picture 26: All requirements fulfilled

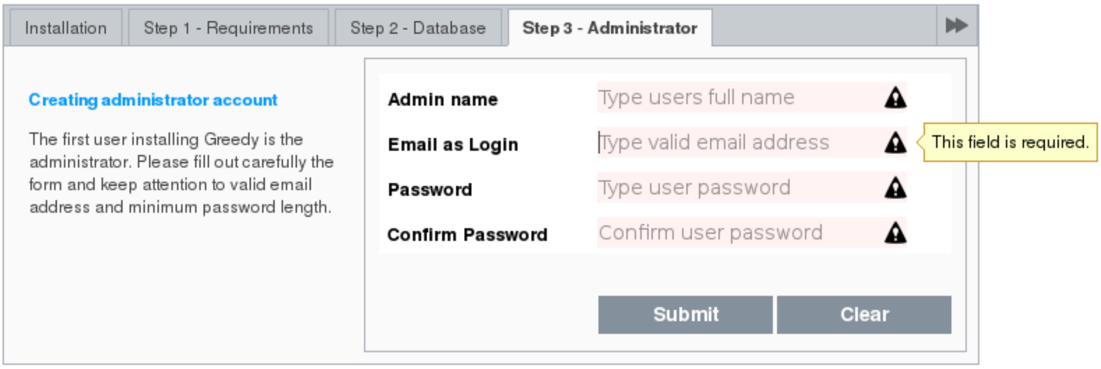
In step 2 in the current state, the user has nothing to do just click the arrow.



Picture 27: Nothing to do

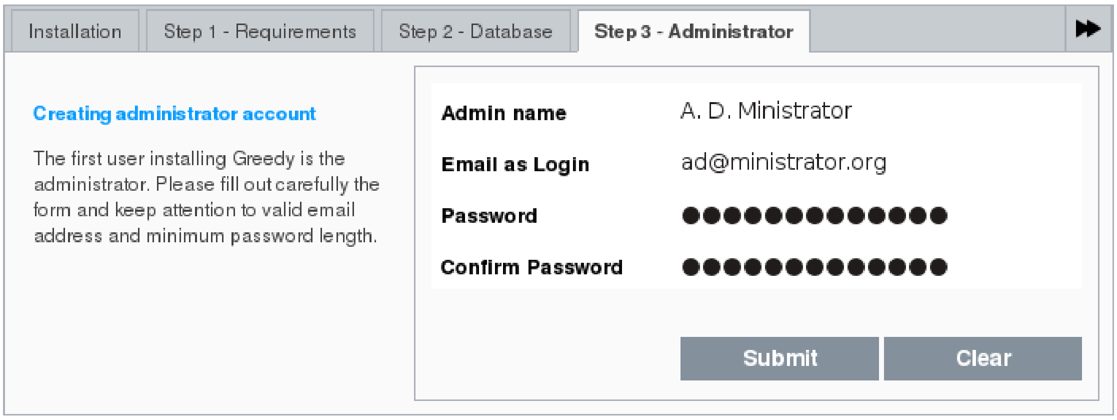
In further releases, the installation can support MySQL and/or PostgreSQL as database systems too.

The data for administration has to be inserted in step 3.



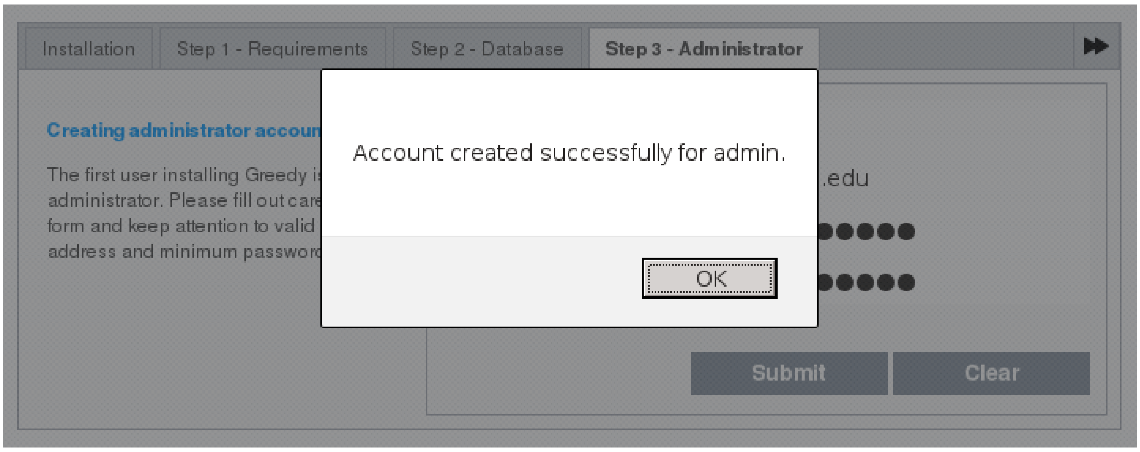
Picture 28: Data for administration not completed

As long as not all required data is inserted the user will get an error message and cannot proceed to the next step.



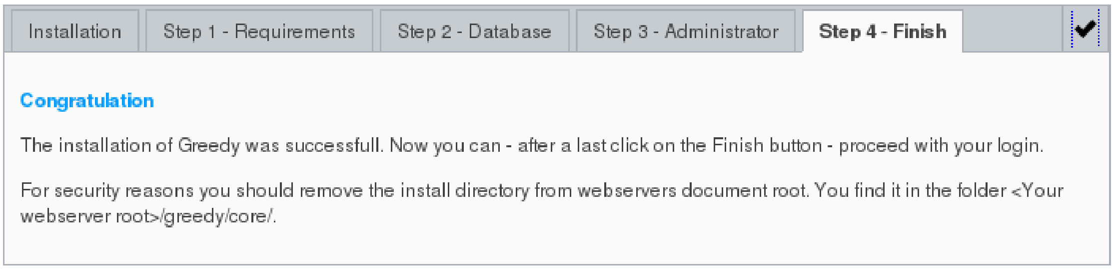
Picture 29: Data input completed

After a click on “Submit” the user will see the next picture.



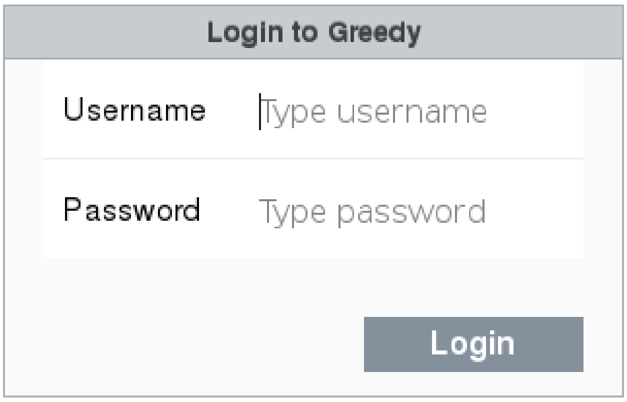
Picture 30: Admin account created successfully

A click on OK will terminate the installation and the user can proceed after confirmation to the last step with the login to greedy.

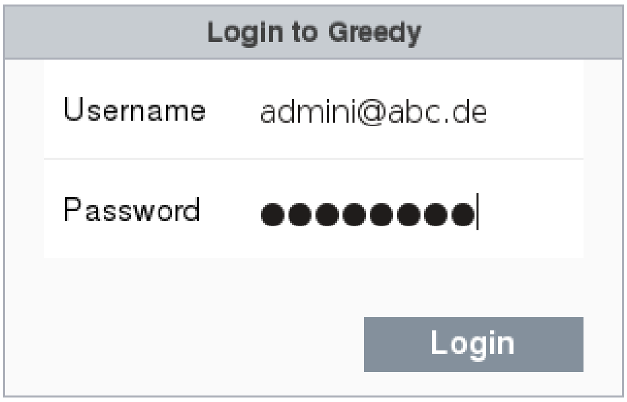


Picture 31: Installation is finished

Here the user has to insert his username and password.

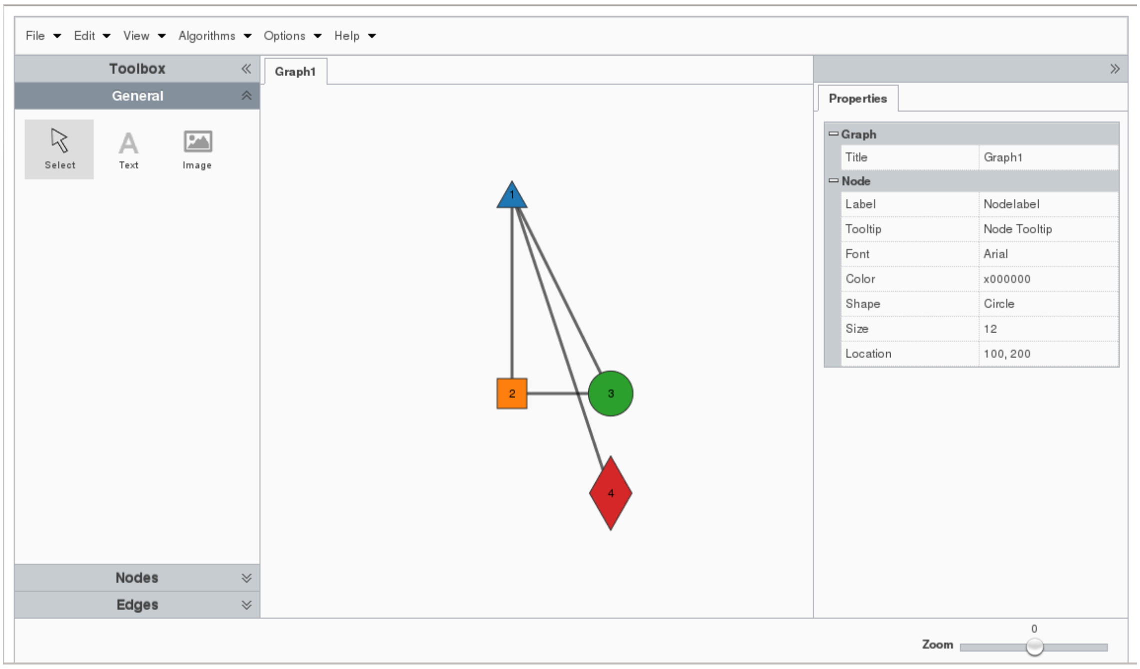


Picture 32: Greedy login window



Picture 33: User data inserted

With the last click on “Login” the user will see the initial screen of greedy.



Picture 34: The initial screen of greedy

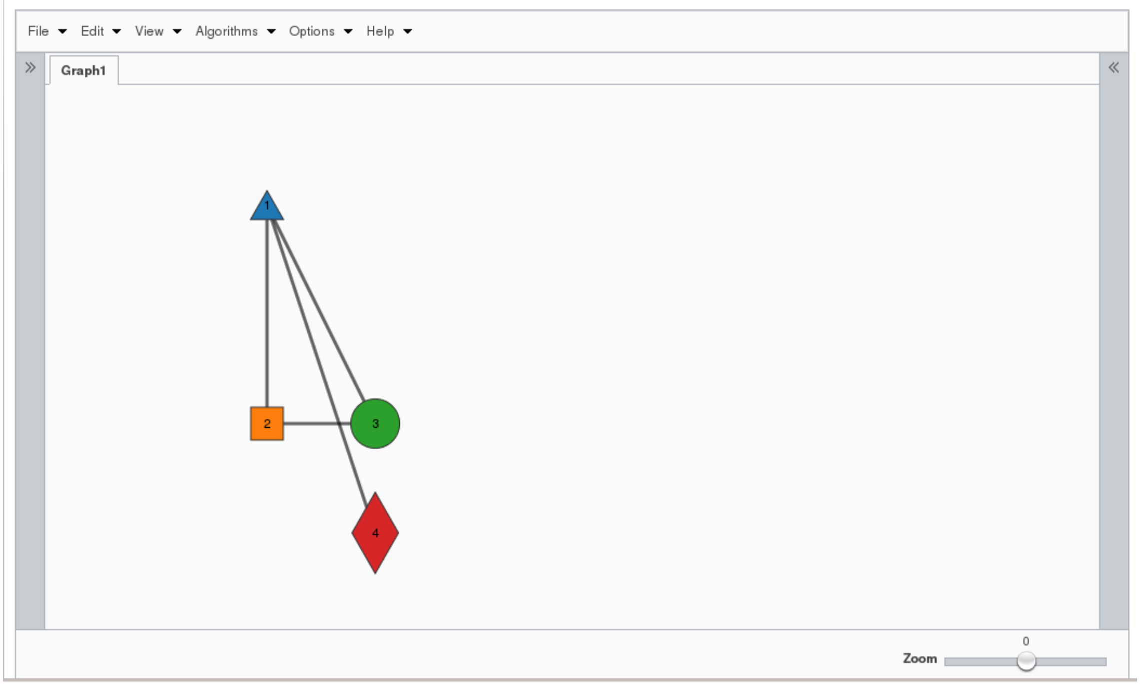
(The displayed graph is for testing purposes only, in the final version the work-space will be blank.)

Now the user may begin to work with greedy.

6.2 Workspace, Toolbar and Properties

In the centre there is the workspace. Right and left of it you find the toolbar and the area where the properties can be displayed and edited for the selected node. Both can be turned off and on.See picture 35. (The property “Location” cannot be edited, it is for information only.)

In the toolbar, which is an accordion mechanism, there are the items “General” for selection of a graph element and to insert text or an image. Under the item



Picture 35: Toolbar and Properties turned off

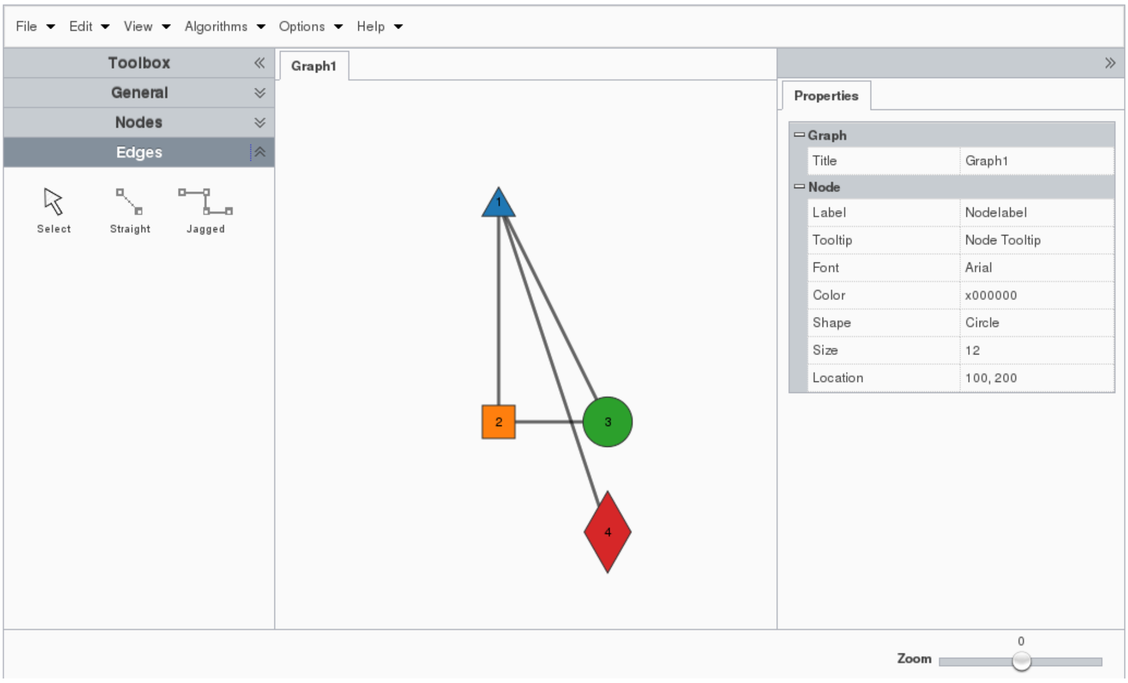
“Nodes” the user can select a node, or choose one of the four node shapes: circle, square, diamond or triangle, to create an new node. Under the item “Edges” it is possible to select an edge, or to create a new edge as straight line or jagged line. See picture 36.

On the right side of the workspace there is the properties area, where the name of the graph can be altered and the node properties lable, tooltip, font, colour, shape and size can be changed. Below the properties area there is a slider with which the graph can be zoomed in and out (currently this feature is not enabled, it is on the To-do-list, but instead the user can zoom in and out with the mouse wheel).

6.3 The Menu Bar

In the menu bar above the workspace there are the items: File, Edit, View, Algorithms, Options and Help.

The item “File” contains: “New” to create a new graph, “Open” and “Save” to



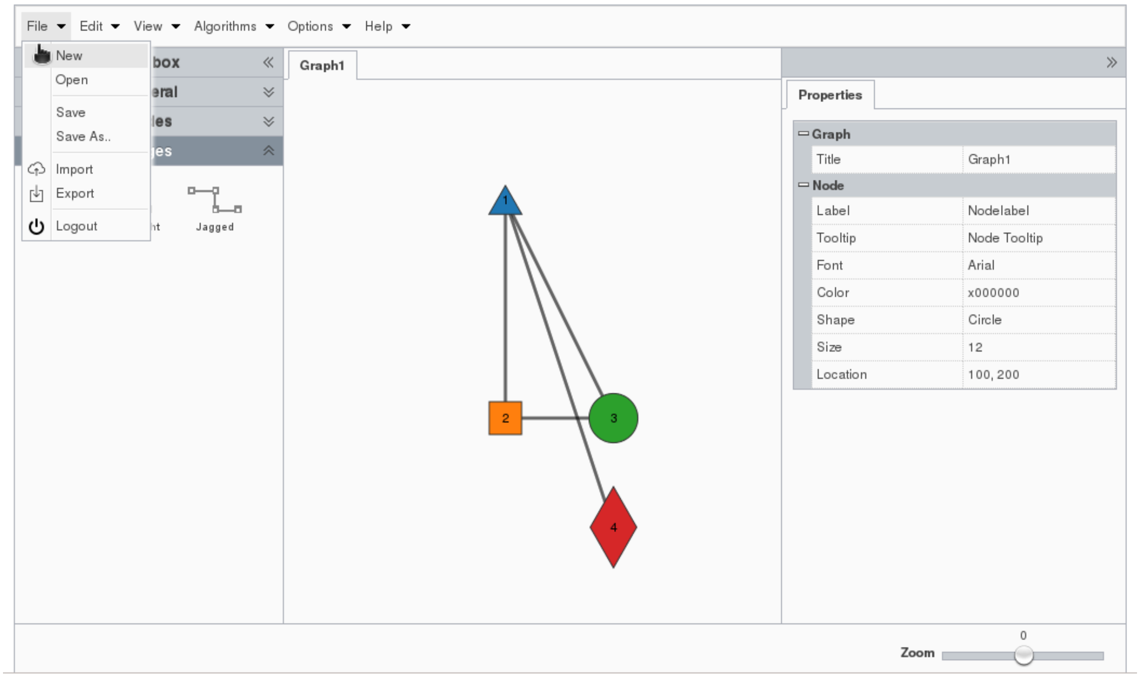
Picture 36: Toolbar with item "Edges" choosen

down- and up-load a graph from the graph database which can be located at UOC, “Save as...” to save a graph with a different name, “Import” and “Export” to load and save a graph on the local computer and “Logout” to quit greedy. See picture 37.

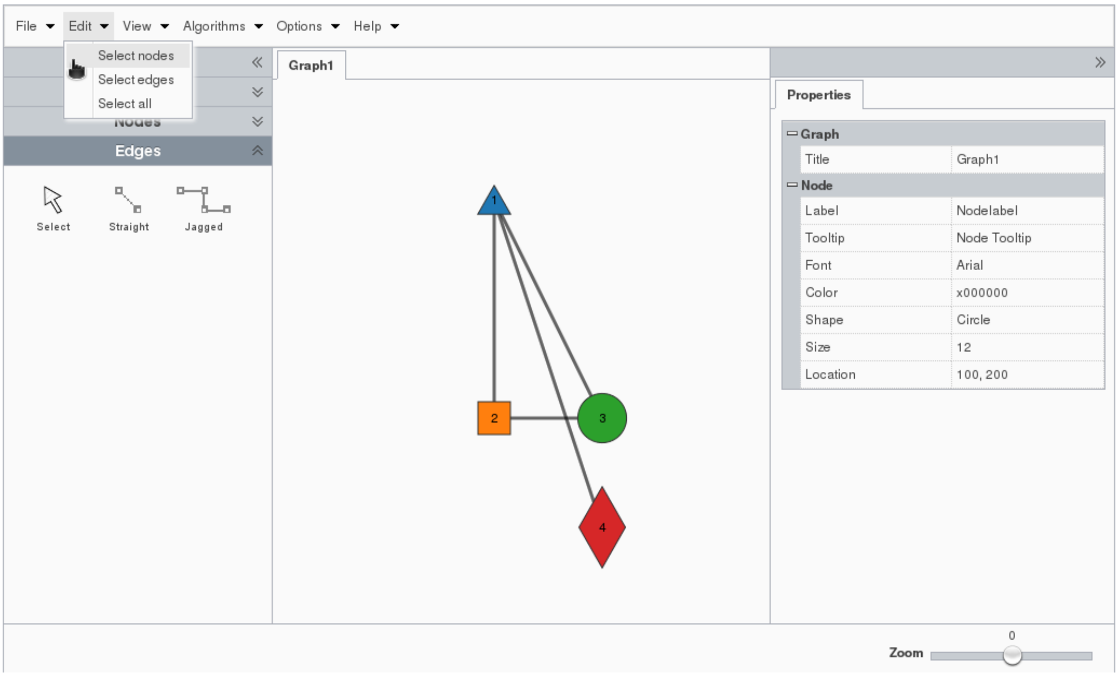
Next is the item “Edit” where nodes, edges or “all” can be selected for demonstration purposes. See picture 38.

The third item in the menu bar is “View”. Here the toolbar and properties can be

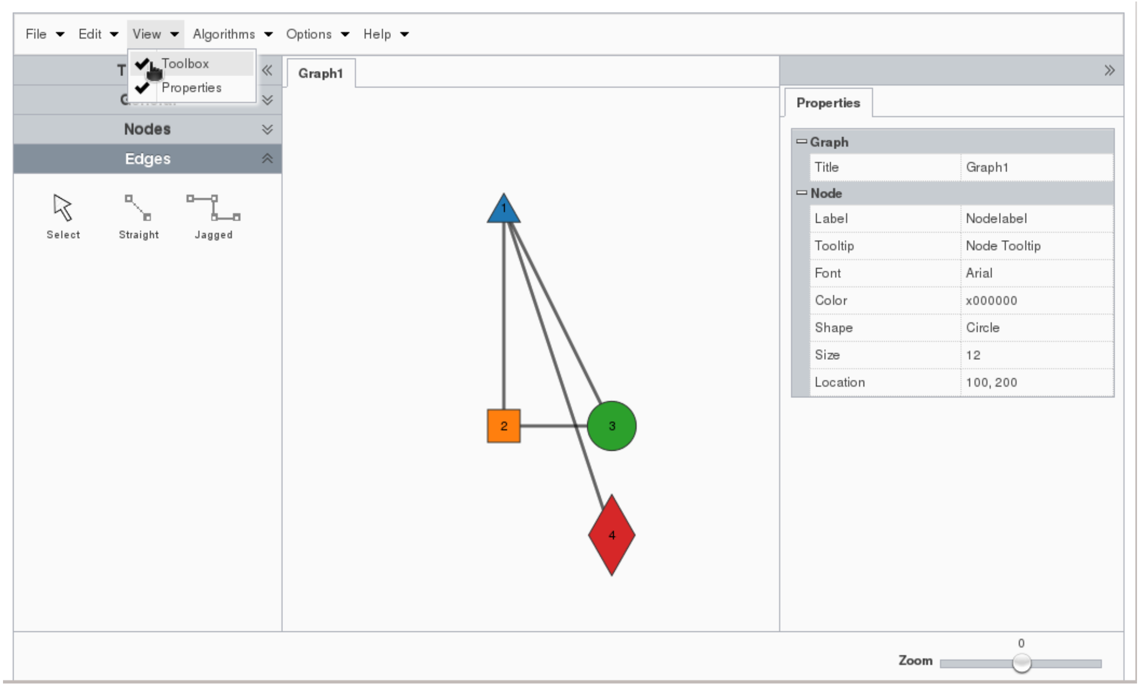
selected to be displayed or not. See picture 39.
“Algorithms” is the next item in the menu bar. Here the user can choose an algorithm to apply to a graph. Currently only the Dijksta algorithm (shortest path) is implemented (description see below). More algorithms can be integrated easily. See picture 40.
The last but one item in the menu bar is the “Options” item. Here it is possible to change a graph's drawing from fixed to force-directed. If the hook at “Fix selected node” is set and a force-directed graph is drawn and the user had moved a node to another position than this node will be fixed at this position.
The third item in Options is “Generate graphs”. Here the user can choose template graphs from a list. See pictures 41 and 42. As an example the graph K5 is selected in picture 43.
The most right item in the menu bar is “Help”. Here the user will find the help sytem (not yet fully integrated) and the information about the licence and a link to it. See picture 44.



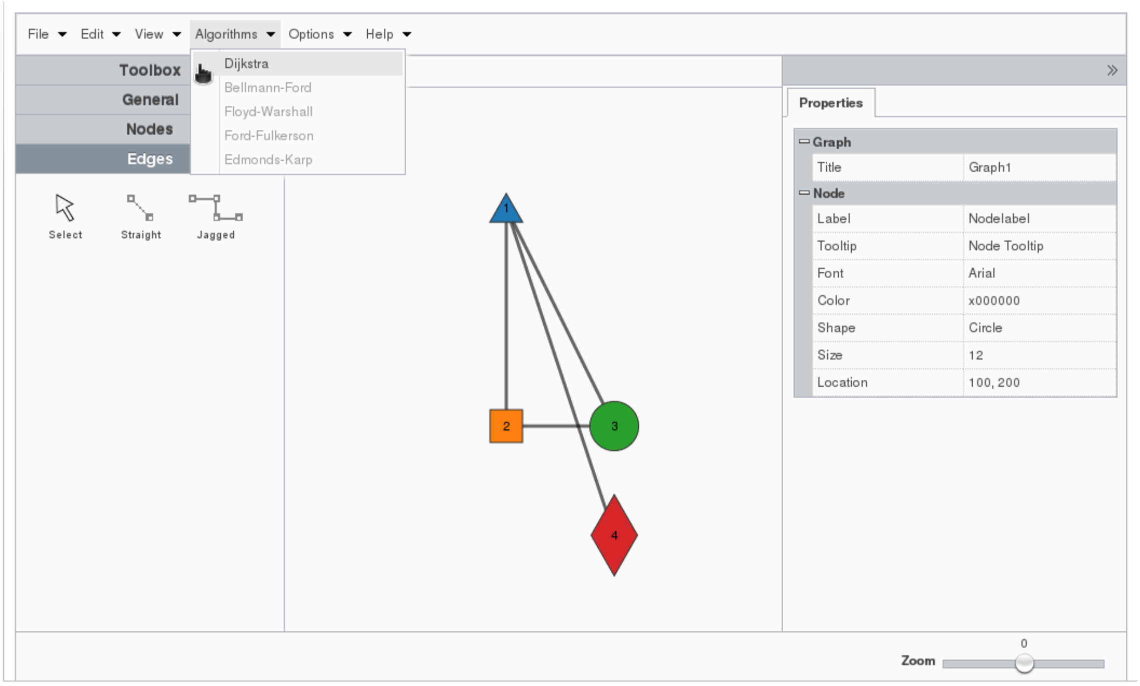
Picture 37: Menu bar with item Files selected



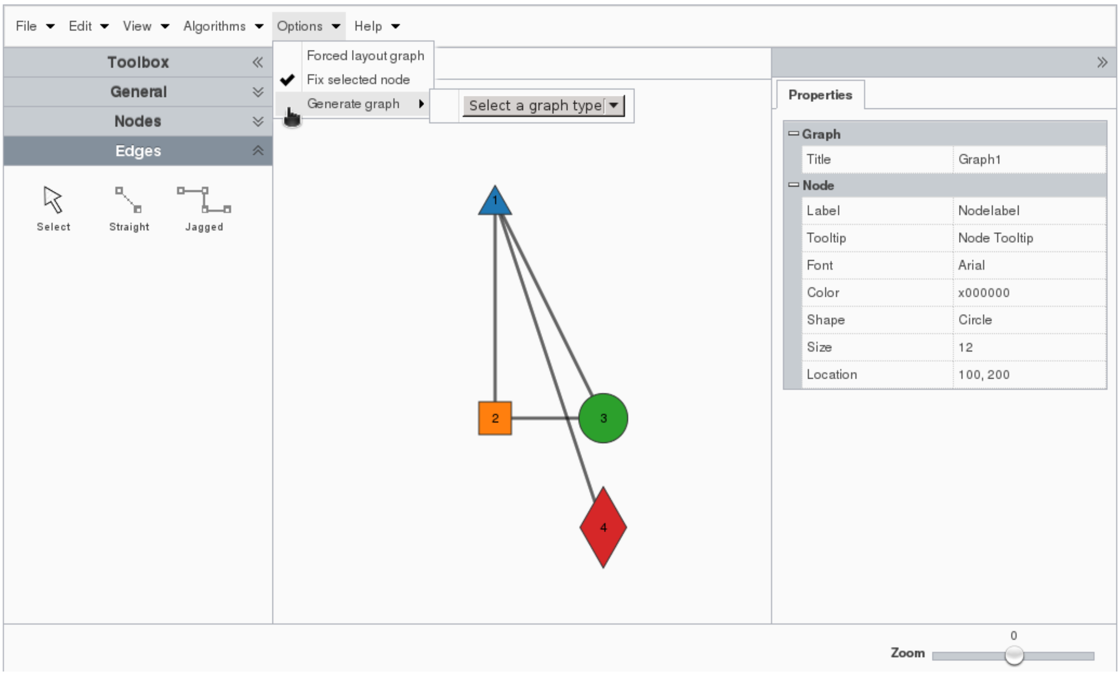
Picture 38: Menu bar with item Edit choosen



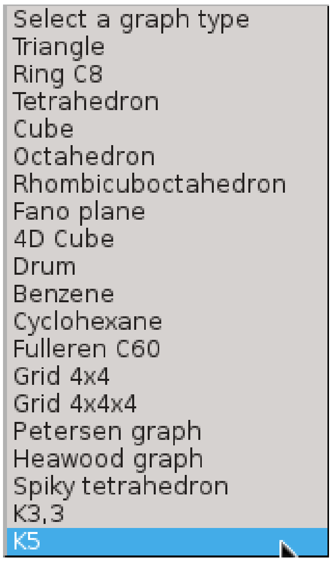
Picture 39: View item of the menu bar



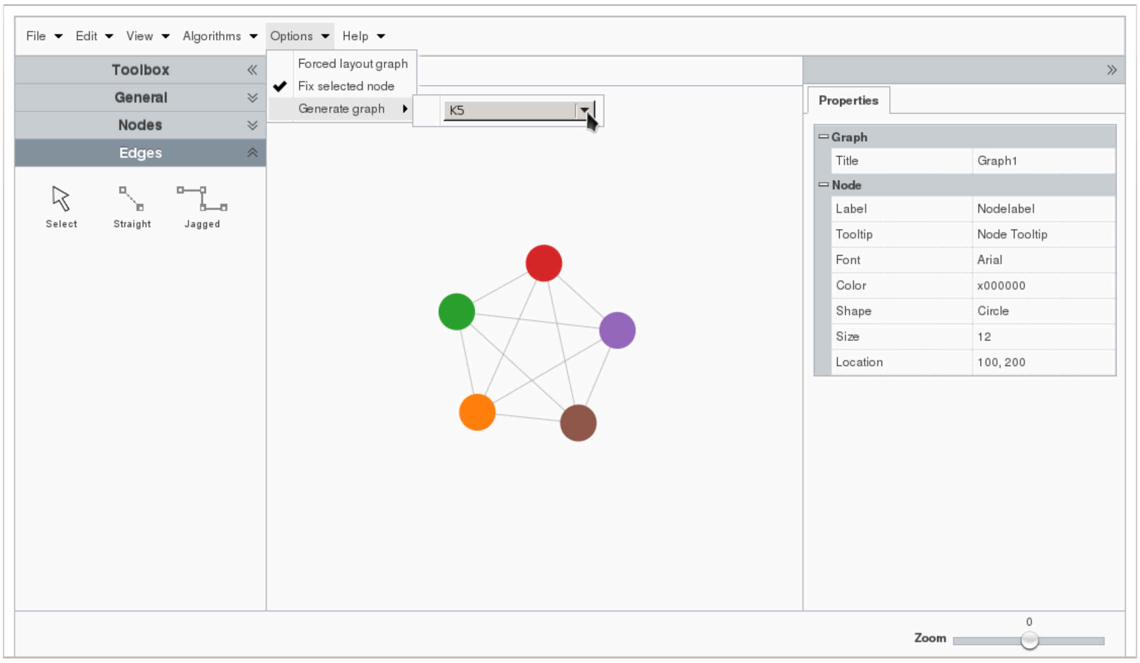
Picture 40: Menu item Algorithms selected



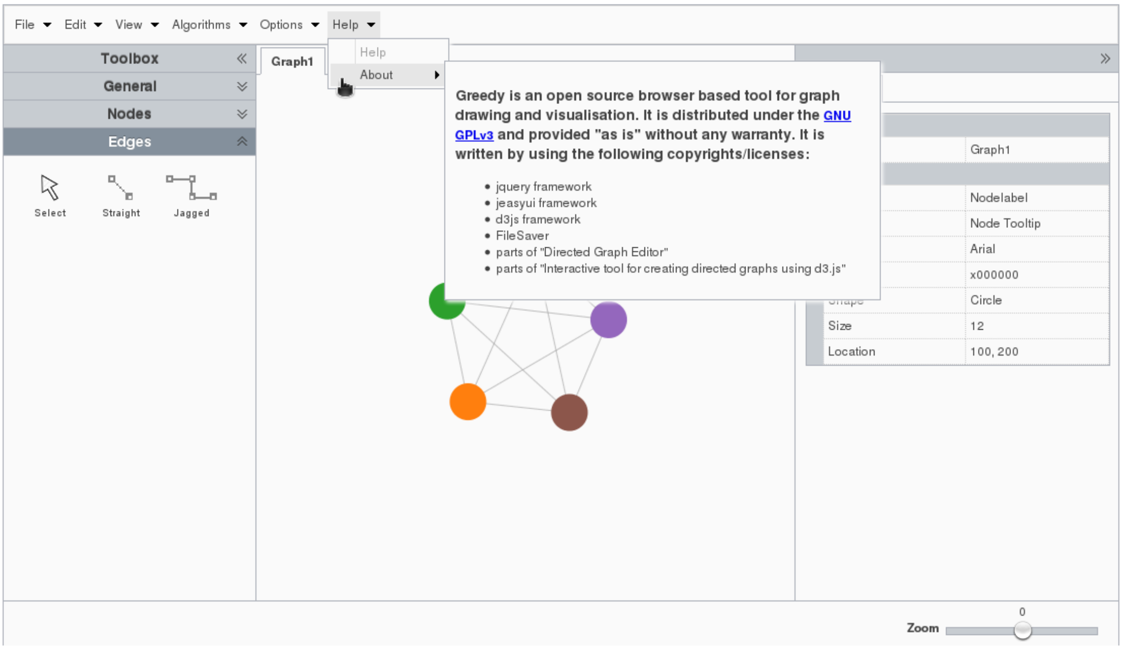
Picture 41: Item Options in the menu bar selected



Picture 42: List of template graphs



Picture 43: Graph K5 selected from list



Picture 44: Item Help in menu bar selected with information about the licence

6.4 Graph edit options

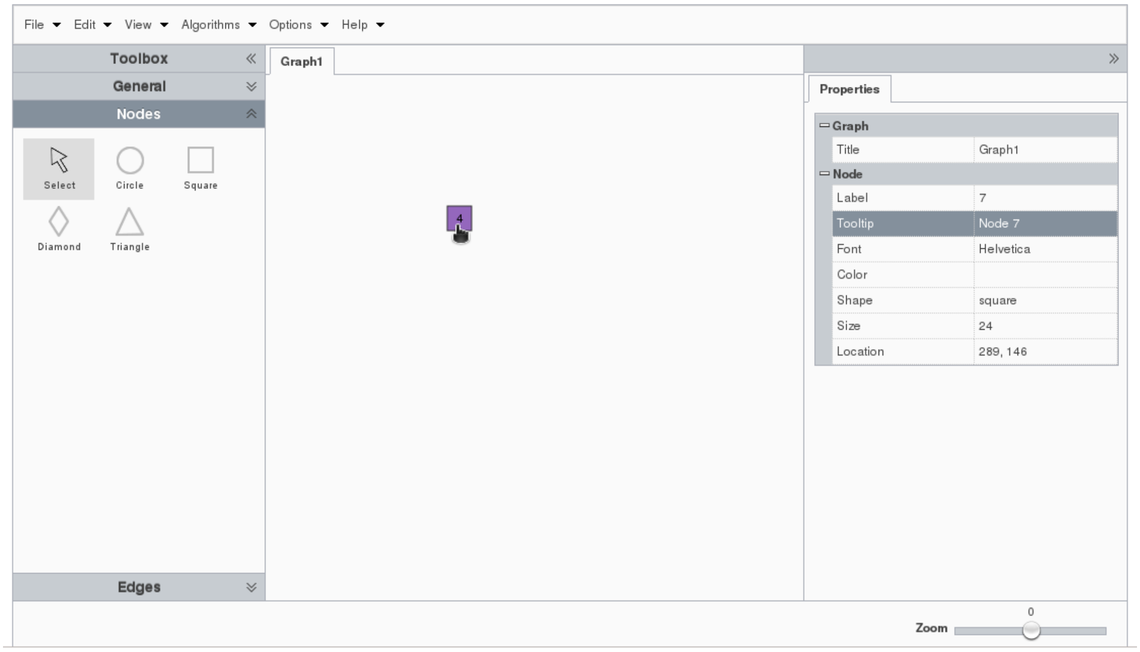
The following features are implemented in the graph editor:

The user can draw graphs interactively and

load graphs a. from server site database (central)

b. import / export (JSON format, jpeg format)

To create a new graph the user has to choose “Nodes” from the toolbar, select one of the shapes and subsequently to click anywhere on the workspace. To create more nodes these steps have to be repeated until all nodes are created. See picture 45.

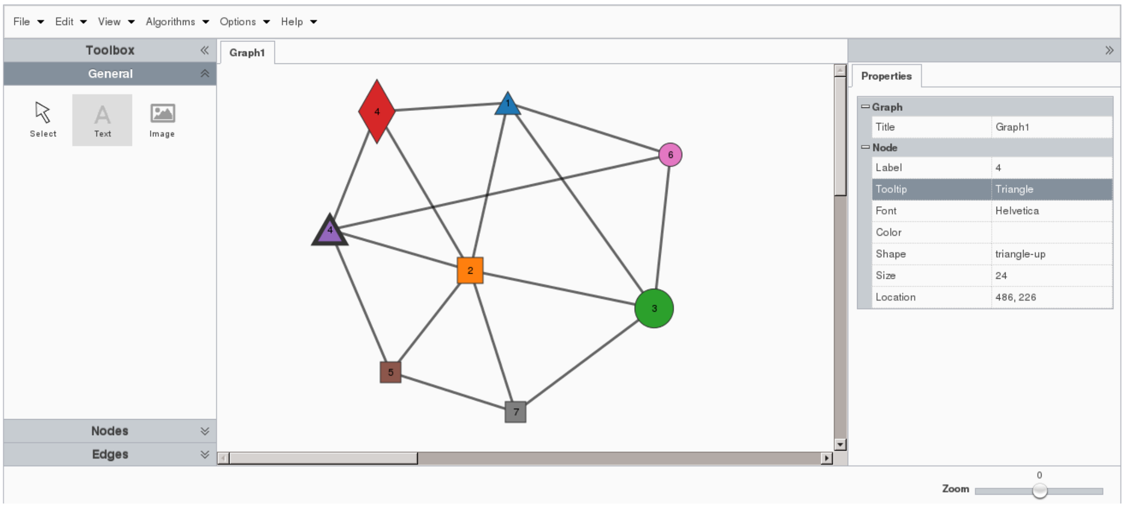


Picture 45: New node created

Now the edges have to be created. The user has to choose “Edges” from the toolbar, select “straight” and subsequently to click on the first node and by holding down the mouse button move to the second node to which the connection shall go. Alternatively the user can select the straight icon to draw edges between nodes. This steps have to be repeated until all edges are created. See picture 46. By clicking anywhere in the workspace, holding down the mouse button and moving, the graph can be moved interactively on the workspace.

The Dijkstra algorithm

This is a simple implementation of the Dijkstra algorithm in JavaScript by Neal Bohlings. For further information see: http://www.nealbohling.com/2014/05/dijkstra-javascript-d3-js, http://en.wikipedia.org/wiki/Edsger_W._Dijkstra and

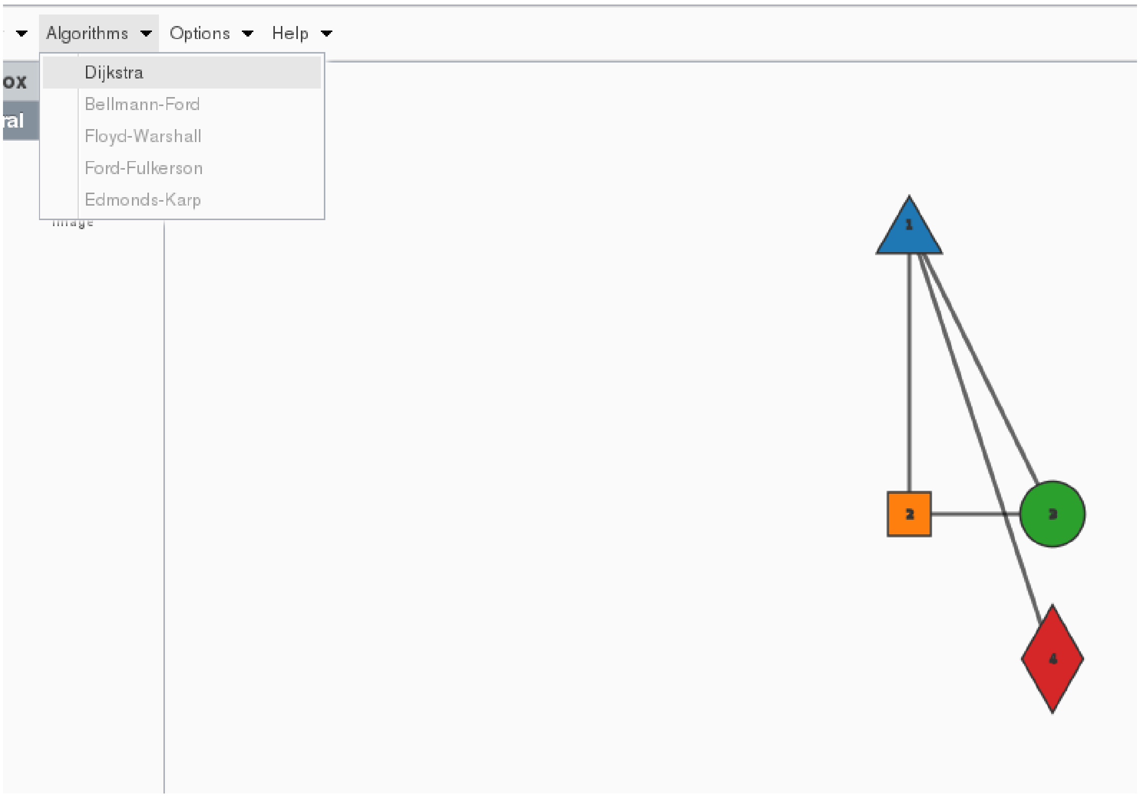


Picture 46: A new graph was created

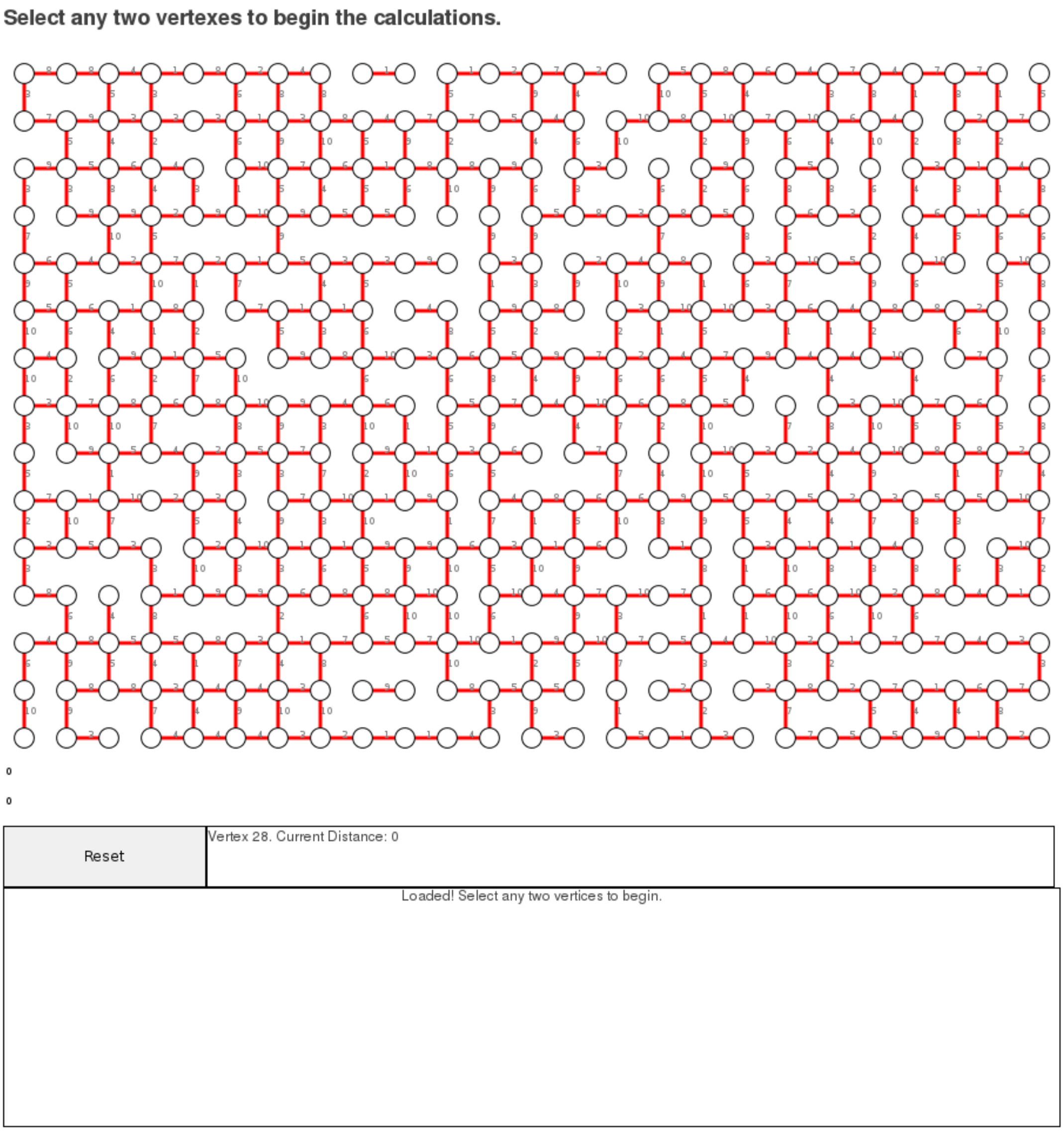
http://en.wikipedia.org/wiki/Dijkstra's_algorithm

This algorithm finds the minimum distance between two nodes.

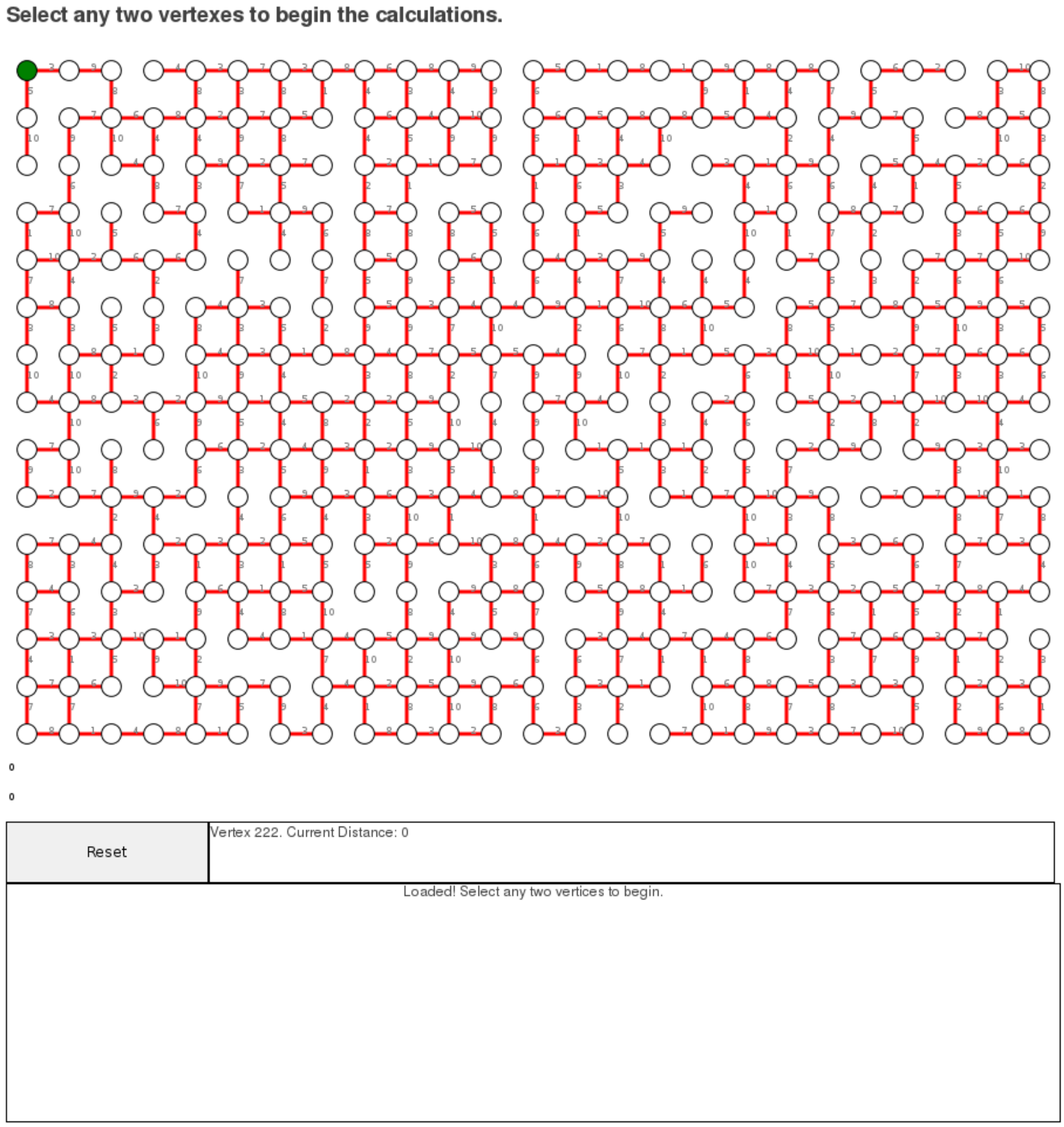
Initialy it generates a grid of nodes where the edges are weighted randomly. Not all possible edges are created but only 75%, so that some connections do not exist. After the user has selected the start and the end nodes, the algorithm begins, "visited" nodes will turn gray, the node under current consideration yellow and when the calculation is completed, the path will be coloured red. A status messages will be shown below, as well as information for any individual node. See pictures 47 to 51.



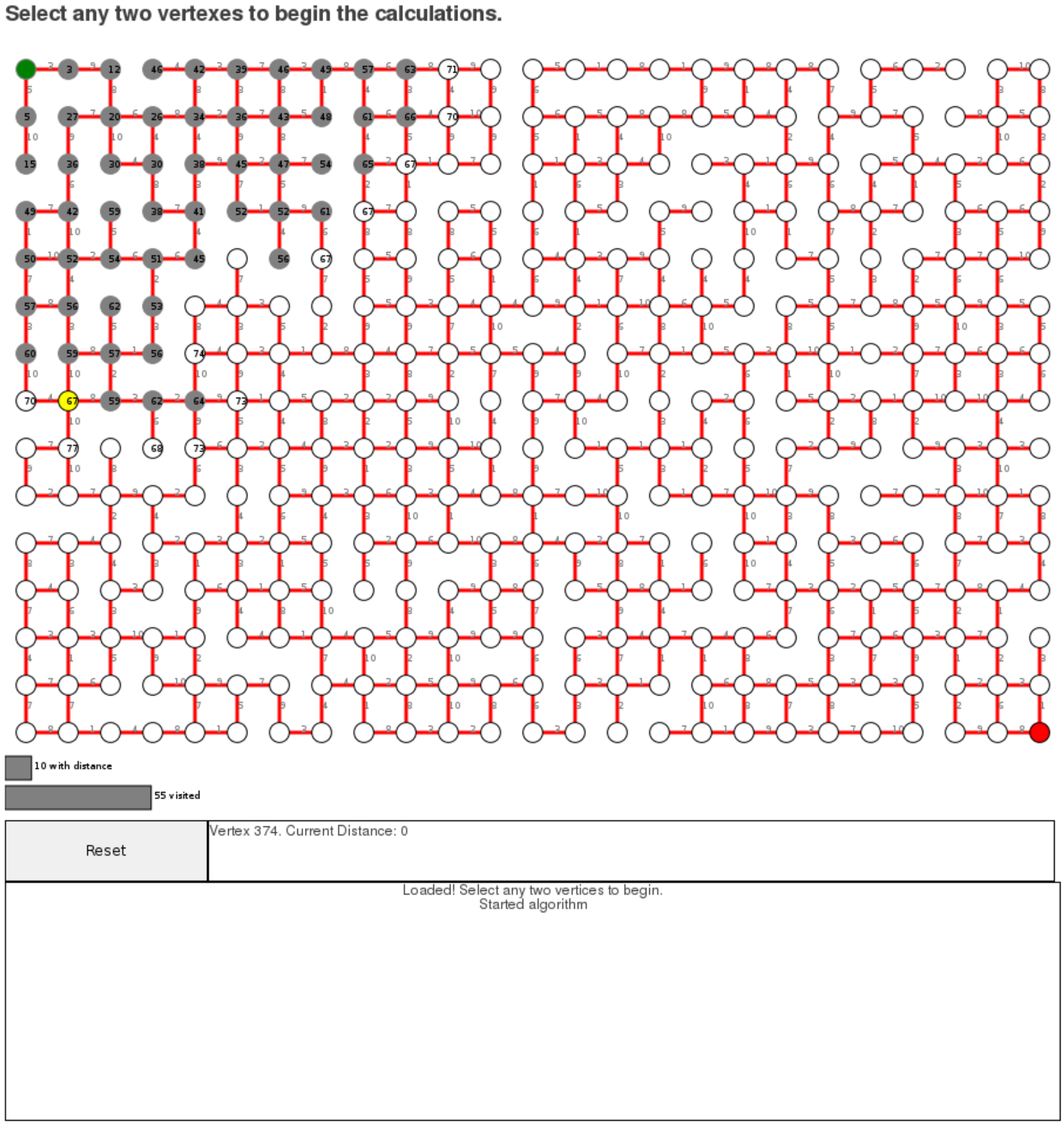
Picture 47: Dijkstra algorithm selected in the Algorithms item



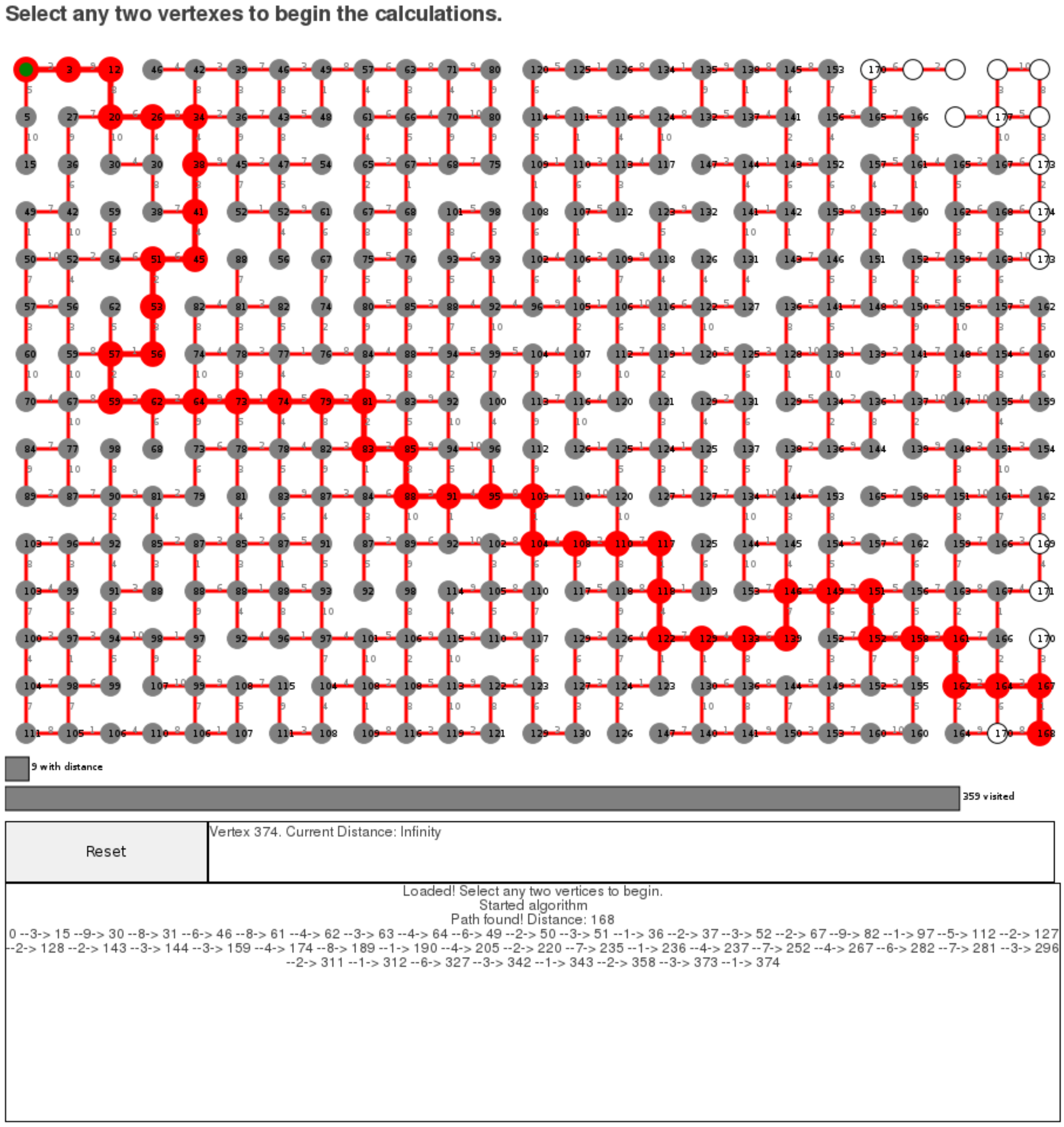
Picture 48: A new Dijkstra grid was generated



Picture 49: The start node is marked (green node in upper left corner)



Picture 50: After the end node (red node in lower right corner) was marked the calculation starts (gray and yellow nodes)



Picture 51: Calculation of the shortest path is finished (red nodes)

6.5 To Do List

Software one time will reach a stable version, which will be called as a general rule version 1, but this does not mean that the development is finished. Software development will never terminate, there are new features to implement, bugs have to be fixed and much more is to do.
This software is not an exception.
Things to do:

- integrate more graph drawing algorithms
- improve labeling
- implement interactive zooming
- integrate gravity slider for force-directed graphs
- make template graphs editable
- make edges directed (right-, left- and bi-directed)
- predefine custom created graphs
- establish multiple graph tabs simultanously
- add user administration
- add custom settings
- complete the help support

6.6 General Features

An user authentication with login / password / course number and access rights is integrated in the software tool.
Used libraries and their respective licences:

Library	Licence
D3.js	BSD 3-clauses
jQuery.js	MIT
jQuery.easyui.js	GPLv3
easyloader.js	GPLv3
filesaver.js	MIT / X11
jason2.js	Public Domain *

Table 7: Used libraries and their licences

All licences are compatible with the GNU GPL.

* Being in the public domain is not a license; rather, it means the material is not copyrighted and no license is needed. Practically speaking, though, if a work is in the public domain, it might as well have an all-permissive non-copyleft free software license. Public domain material is compatible with the GNU GPL. See http://www.gnu.org/licenses/license-list.en.html#GPLCompatibleLicenses

Short descriptions of the libraries (taken from en.wikipedia.org or the respective web pages)

D3.js

D3.js (or just D3 for Data-Driven Documents) is a JavaScript library that uses digital data to drive the creation and control of dynamic and interactive graphical forms which run in web browsers. It is a tool for data visualization in W3C-compliant computing, making use of the widely implemented Scalable Vector Graphics (SVG), JavaScript, HTML5, and Cascading Style Sheets (CSS3) standards. In contrast to many other libraries, D3 allows great control over the final visual result. Its development was noted in 2011,[3] as version 2.0.0 was released in August 2011. As of February 2014, the library is at version 3.4.3. Developers are Michael Bostock (maintainer), Jeffrey Heer, Vadim Ogievetsky, and community.

jQuery.js

jQuery is a cross-platform JavaScript library designed to simplify the client-side scripting of HTML. It was released in January 2006 at BarCamp NYC by John Resig. It is currently developed by a team of developers led by Dave Methvin. Used by over 80% of the 10,000 most visited websites, jQuery is the most popular JavaScript library in use today. jQuery is free, open source software, licensed under the MIT License. jQuery's syntax is designed to make it easier to navigate a document, select DOM elements, create animations, handle events, and develop Ajax applications. jQuery also provides capabilities for developers to create plug-ins on top of the JavaScript library. This enables developers to create abstractions for low-level interaction and animation, advanced effects and high-level, theme-able widgets. The modular approach to the jQuery library allows the creation of powerful dynamic web pages and web applications. The set of jQuery core features — DOM element selections, traversal and manipulation — enabled by its *selector engine* (named "Sizzle" from v1.3), created a new "programming style", fusing algorithms and DOM-data-structures; and influenced the architecture of other JavaScript frameworks like YUI v3 and Dojo.

jQuery.easyui.js

jQuery EasyUI framework helps you build your web pages easily.

- easyui is a collection of user-interface plugin based on jQuery.
- easyui provides essential functionality for building modern, interactive, javascript applications.
- using easyui you don't need to write many javascript code, you usually

defines user-interface by writing some HTML markup.

- complete framework for HTML5 web page.
- easyui save your time and scales while developing your products.
- easyui is very easy but powerful.

(taken from http://www.jeasyui.com)

easyloader.js

easyloader.js is a part of jQuery.js

filesaver.js

filesaver.js implements the HTML5 W3C `saveAs()` FileSaver interface in browsers that do not natively support it. FileSaver.js is the solution to saving files on the client-side, and is perfect for webapps that need to generate files, or for saving sensitive information that shouldn't be sent to an external server. taken from https://github.com/eligrey/FileSaver.js

jason2.js

JSON is a light-weight, language independent, data interchange format. See http://www.JSON.org. The files in this collection implement JSON encoders / decoders in JavaScript. JSON became a built-in feature of JavaScript when the ECMAScript Programming Language Standard - Fifth Edition was adopted by the ECMA General Assembly in December 2009. Most of the files in this collection are for applications that are expected to run in obsolete web browsers. For most purposes, json2.js is the best choice. json2.js: This file creates a JSON property in the global object, if there isn't already one, setting its value to an object containing a stringify method and a parse method. The parse method uses the eval method to do the parsing, guarding it with several regular expressions to defend against accidental code execution hazards. On current browsers, this file does nothing, prefering the built-in JSON object. Taken from https://github.com/douglascrockford/JSON-js

Structure of the file system:
Document root consists of these folders:

- core
  - diagrams additional info about server saved diagrams
  - install quick installation routine for php
  - js Javascript files
  - lib external libraries
  - plug-ins js script for installable algorithms
  - sys php files
  - test internal test routines

- res
  - css cascading style sheets
  - icons open source icon libs
  - themes files for layout of the UI

7 References

[AH77] K. Appel and W. Haken. Every planar map is four colorable, part I: discharging. Illinois J. Math., 21:429–490, 1977.

[Atos13] www.qsos.org/2013/06/02/qsos-20-is-out, Version 2.0, 2013.

[BBBL08] Ch. Bachmauer, F. J. Brandenburg, W. Brunner and R. Fülöp. Drawing recurrent hierarchies. Journal of Graph Algorithms and Applications, 16(2):151–198, 2012.

[BFM07] N. Bochinon, S. Felsner and M. Mosbah. Convex drawings of 3-connected plane graphs. Algorithmica, 47:399–420, 2007.

[BJL02] V. Buchheim, M. Jünger and S. Leipert. Improving Walker's algorithm to run in linear time. In Michael T. Goodrich and Stephan G. Kobourov, editors, Graph Drawing, (Proceedings of 10th International Symposium on Graph Drawing, 2002), volume 2528 of Lecture Notes in Computer Science, pages 344–353. Springer, 2002.

[BKW03] U. Brandes, P. Kenis and D. Wagner. Communicating centrality in policy network drawings. IEEE Transact. Vis. Comput. Graph., 9(2):241–253, 2003.

[BM04] J. Boyer and W. Myrvold. On the cutting edge: Simplidies O(n) planarity by edge addition. Journal of Graph Algorithms and Applications, 8(3):241–273, 2004.

[Bos02] P. Bose. On embedding an outer-planar graph in a oiunt-set. Computational Geometry, 23(3):303–312, 2002.

[Bra97] F. J. Brandenburg. Graph clustering I: Cycles of cliques. In Proc. GD ’97, LNCS 1353, pages 158–168, 1997.

[BTT84] C. Batini, M. Talamo and R. Tamassia. Computer aided layout of entity-relationship diagrams. Journal of Systems and Software, 4:163–173, 1984.

[CGMW11] M. Chimani, Ph. Hungerländer, M. Jünger and P. Mutzel. An SDP approch to multi-level crossing minimization. In M. Müller-Hannemann and R. F. Werneck, editors, ALENEX, pages 116–

126, SIAM, 2011.

[CLY01] H.-L. Chen, H.-I. Lu and H.-Ch. Yen. On maximum symmetric subgraphs. In Graph Drawing, 8th International Symposium, GD00, Colonial Wiliamsburg, VA, USA, Sept. 2000, Proceedings, vol. 1984 of Lecture Notes in Computer Science, pages 372–383. Springer, 2001.

[CMS08] M. Chimani, P. Mutzel and J. M. Schmidt. Efficient extraction of multiple Kuratowski subdivisions. In Seok-Hee Hong, Takao Nishizeki, and Wu Quan, editors, Graph Drawing (GD 2007), volume 4875 of LNCS, pages 159–170. Springer, 2008.

[CPZ04] M. A. Carreira-Perpinan and R. S. Zemel. Proximity graphs for clustering and manifold learning. In Neural Information Processing Systems, NIPS 2004, 2004.

[CWZ06] D. Cruz, T. Wieland and A. Ziegler: Evaluation Criteria for Free / Open Source Software Products Based on Project Analysis. Software Process Improvement and Practice, pages: 107-122, 2006.

[DBTV01] G. Di Battista, R. Tamassia and L. Vismara. Incremental convex planarity testing. Information Computation, 169:94–126, August 2001.

[DDLW05] E. di Giacomo, W. Didimo, G. Liotta and S. K. Wismath. Curve-constrained drawings of planar graphs. Computational Geometry: Theory and Applications, 30:1–23, 2005.

[dFPP90] H. de Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. Combinatorica, 10(1):41–51, 1990.

[DMM97] U. Dogrusoz, B. Madden and P. Madden. Circular layout in the graph layout toolkit. In Proc. GD ’96, LNCS 1190, pages 92–100, 1997.

[dNE02] H. A. D. do Nascimento and P. Eades. A focus and constraint-based genetic algorithm for interactive directed drawing. HIS, pages 634–643, 2002.

[Ead84] P. Eades. A heuristic for graph drawing. Congressus Numerantium, 42:149–160, 1984.

[Flei07] T. Fleischfresser: Evaluation von Open Source Projekten: Ein GQM-basierter Ansatz. Diplomarbeit, Freie Universität Berlin, 2007.

[FR91] T. Fruchterman and E. Reingold. Graph drawing by force-directed placement. Softw. – Pract. Exp., 21(11):1129–1164, 1991.

[Fre91] H. Freeman. Computer name placement. In D. J. Maguire, M. F. Goodchild and D. W. Rhind, editors, Geographical Information Systems: Principles and Applications, pages 445–456. Longman, London, 1991.

[FW74] R. L. Francis and J. A. White. Facility Layout and Location, Prentice-Hall, New Jersey, 1974.

[GADM04] S. Grivet, D. Auber, J.-P. Domenger and G. Melancon. Bubble tree drawing algorithm. In International Conference on Computer Vision and Graphics, pages 633–641. Springer Verlag, 2004.

[GGK04] P. Gajer, M. T. Goodrich, and S. G. Kobourov. A fast multi-dimensional algorithm for drawing large graphs. Computational Geometry: Theory and Applications, 29(1):3–18, 2004.

[GL05] S. Graf and B. List: An Evaluation of Open Source E-Learning Platforms Stressing Adaptation Issues. In ICALT, pp. 163-165. 2005.

[GM98] C. Gutwenger and P. Mutzel. Planar polyline drawings with good angular resolution. In S. Whitesides, editor, Graph Drawing (Proc. GD 98), volume 1547 of Lectures Notes Comput. Sci., 167–182. Springer Verlag, 1998.

[GO04] J. E. Goodman and J. O'Rourke, editors. Handbook of Discrete and Computational Geometry, 2nd Edition. CRC Press, 2004.

[GR03a] A. Garg and A. Rusu. Area-efficient order-preserving planar straight-line drawings of ordered trees. International Journal of Computational Geometry and Applications, 13(6):487–505, 2003.

[GT02] M. T. Goodrich and R. Tamassia. Algorithm design: foundations, analysis and Internet examples. John Wiley and Sons, Inc., New York, NY, 2002.

[GT97] A. Garg and R. Tamassia. GIOTTO: A System for visualizing hierarchical structures in 3D. In S. North, editor, Graph Drawing: Symposium on Graph Drawing, GD '96, volume 1190 of Lecture Notes in Computer Science, pages 193–200, Springer Verlag, 1997.

[HJLM93] F. Harary, M. S. Jacobson, M. J. Lipman and F. R. Morris. On abstract sphere of influence graphs. Mathematical and Computater Modelling, 17(11):77–83, 1993.

[HN05a] S.-H. Hong and N. S. Nikolov. Hierarchical layouts of directed graphs in three dimensions. In P. Healy and N. S. Nikolov, editors, Graph Drawing: Proceedings of 13th International Symposium, GD 2005, volume 3843 of LNCS. Springer Verlag, 2005.

[HT08] B. Haeupler and R. E. Tarjan. Planarity algorithms via PQ-trees (extended abstract). Electronic Notes in Discrete Mathematics, 31:143–149, 2008.

[HT74] J. Hopcroft and R. E. Tarjan. Efficinet planarity testing. J. ACM, 21(4):549-568, 1974.

[KH97] G. Kant and X. He. Regular edge labeling of 4-connected plane graohs and its applications in graph drawing problems, Theoretical Computer Science, 172, pp. 175–193, 1997.

[KK89] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. Inform. Process. Lett., 31:7–15, 1989.

[Kur30] K. Kuratowski. Sur le probleme des courbes en topologie. Fund. Math., 15:271–283, 1930.

[KW05] S. G. Kobourov and K. Wampler. Non-Euclidean spring embedders. IEEE Transactions on Visualisation and Computer Graphics, 11(6):757–767, 2005.

[LEC67] A. Lempel, S. Even and I. Cederbaum. An algorithm for planarity testing of graphs. In Theory of Graphs: Internat. Symposium (Rome 1966), pages 215-232, New York, Gordon and Breach, 1967.

[LL90] Y.-T. Lai and S. M. Leinwand. A theory of rectangular dual graphs, Algorithmica, 5, pp. 467–483, 1990.

[LLMW98] G. Liotta, A. Lubiw, H. Meijer and S. H. Whitesides. The rectangle of influence drawability problem. Comput. Geom. Theory and Applications, 10(1):1–22, 1998.

[LR86] F. T. Leighton and A. L. Rosenberg. Three-dimensional circuit layouts. SIAM J. Comput., 15(3):793–813, 1986.

[Mac03] B. MacLennan. Molecular combinatory computing for nanostructure synthesis and control. In Proceedings 3rd IEEE Conference on Nanotechnology, volume 2 of IEEE Press, pages 179–182, 2003.

[MOS98] P. Mutzel, T. Odenthal and M. Scharbrodt. The thickness of graphs: A survey. Graphs and Combinatorics, 14:59–73, 1998.

[MS92] C. Monma and S. Suri. Transitionsin geometric minimum spanning trees. Discrete Comput. Geom., 8:265–293, 1992.

[Mun97] T. Munzner. H3: Laying out large directed graphs in 3D hyperbolic space. In L. Lavagno and W. Reisig, editors, Proceedings of IEEE Symposium on Information Visualization, pages 2–10, 1997.

[NR04] T. Nishizeki and M. S. Rahman. Planar Graph Drawing, World Scientific, Singapore, 2004.

[PT00] J. Pach and G. Toth. Which crossing number is it, anyway? J. Comb. Theory Ser. B, 80(2):225–246, 2000.

[Rad88] J. D. Radke. On the shape of a set of points. In G. T. Toussaint, editor, Computational Morphology, pages 105–136. North-Holland, Amsterdam, The Netherlands, 1988.

[RMM+95] A. H. Robinson, J. L. Morrison, P. C. Muehrcke, A. J. Kimerling and S. C. Guptill. Elements of Cartography. John Wiley & Sons, Inc., 6th edition, 1995.

[RS07] A. Rusu and C. Santiago. A practical algorithm for planar straight-line grid drawings of general trees with linear area and arbitrary aspect ratio. In Proceedings 11th International Conference on Information Visualisation, pages 743–750. IEEE Computers Society, 2007.

[SB10] K.-J. Stol and M. A. Babar: A comparison framework for open source software evaluation methods. In: Open Source Software: New Horizons. Springer Berlin Heidelberg, pages: 389-394, 2010.

[Sch90] W. Schnyder. Embedding planar graphs on the grid. In Proc. 1st ACM-SIAM Sympos. Discrete Algorithms, pages 138–148, 1990.

[SH99] W. K. Shih and W. L. Hsu. A new palanrity test. Theor. Comp. Sci., 223, 1999.

[She95] N. Sherwani. Algorithms for VLSI Physical Design Automation, 2nd edition, Kluwer Academic Publishers, Boston, 1995.

[SIII00] K. Sugihara, M. Iri, H. Inagaki and T. Imai. Topology-oriented implementation – an approach to robust geometric algorithms. Algorithmica, 27(1):5–20, 2000.

[SM95a] K. Sugiyama and K. Misue. Graph drawing by the magneting spring model. Journal of Visual Languages and Computing, 6(3):217–231, 1995.

[SR34] E. Steinitz and H. Radermacher. Vorlesung über die Theorie der Polyeder. Julius Springer, Berlin, Germany, 1934.

[ST06] J. M. Six and I. G. Tollis. A framework and algorithms for circular drawings of graphs. Jrnl. Of Discrete Algorithms, 4(1), pages 25–50, 2006.

[STT81] K. Sugiyama, S. Tagawa and M. Toda. Methods for visual understanding of hierarchical system structures. IEEE Transactions on Systems, Man, and Cybernetics, 11(2):109–125, 1981.

[Sug02] K. Sugiyama. Graph Drawing and Applications. World Scientific, Singapore, 2002.

[Tam13] R. Tamassia (editor). Handbook of Graph Drawing and Visualization (Discrete Mathematics and Its Applications). CRC Press, Boca Raton, Florida, USA. 2013.

[Tou05] G. Toussaint. Geometric proximity graphs for improving nearest neighbor methods in instance-based learning and data mining. International Journal of Computational Geometry and Applications, 15(2):101–150, 2005.

[Tut63] W. T. Tutte. How to draw a graph. Proc. London Math. Society, 13(52):743–768, 1963.

[UBSE98] J. Utech, J. Branke, H. Schmeck and P. Eades. An evolutionary algorithm for drawing directed graphs. In Proceedings of the 1998 International Conference on Imging Science, Systems, and Technology (CISST'98), pages 154–160, 1998.

[Wal03] C. Walshaw. A multilevel algorithm for force-directed graph drawing. Journal of Graph Algorithms and Applications, 7(3):253–285, 2003.

[Whe11] D. A. Wheeler. How to Evaluate Open Source Software / Free Software (OSS/FS) Programs, published on D. Wheelers homepage: http://www.dwheeler.com/oss_fs_eval.html, revised as of August 5, 2011

[WHF93] C. Ware, D. Hui and G. Franck. Visualizing object oriented software in three dimensions. In Proc. IBM Centre for Advanced Studies Conf. (CASCON'93), pages 1–11, 1993.

[WM08] C. Ware and P. Mitchell. Visualizing graphs in three dimensions. ACM Trans. Appl. Percept., 5(1), 2008.

[Zor90] S. Zoraster. The solution of large 0-1 integer programming problems encountered in automated cartography. Operation Research, 38(5):752–759, September-October 1990.

Appendix A

The BSD 3-clauses licence

BSD licences are a family of permissive free software licenses, imposing minimal restrictions on the redistribution of covered software. This is in contrast to copyleft licenses, which have reciprocity share-alike requirements. The original BSD license was used for its namesake, the Berkeley Software Distribution (BSD), a Unix-like operating system. The original version has since been revised and its descendants are more properly termed modified BSD licenses. Two variants of the license, the New BSD License/Modified BSD License (3-clause) and the Simplified BSD License/FreeBSD License (2-clause) have been verified as GPL-compatible free software licenses by the Free Software Foundation, and have been vetted as open source licenses by the Open Source Initiative, while the original, 4-clause license has not been accepted as an open source license and, although the original is considered to be a free software license by the FSF, the FSF does not consider it to be compatible with the GPL due to the advertising clause.

Besides the original license used in BSD, there are several derivative licenses that are commonly referred to as a "BSD license". Today, the typical BSD license is the 3-clause version, which is revised from the original 4-clause version.

Note that: In all BSD licences as following, **<organization>** is the organization of the **<copyright holder>** or just the <copyright holder>, and **<year>** is the year of the copyright. As published in BSD, <copyright holder> is "Regents of the University of California", and <organization> is "University of California, Berkeley".

Previous BSD licence

Some releases of BSD prior to the adoption of the 4-clause BSD license used a license that is clearly ancestral to the 4-clause BSD license. These releases include 4.3BSD-Tahoe (1988) and Net/1 (1989). Though largely replaced by the 4-clause license, this license can be found in 4.3BSD-Reno, Net/2, and 4.4BSD-Alpha.

The original BSD Licence (4-clause license)

The original BSD license contained a clause not found in later licenses, known as the "advertising clause". This clause eventually became controversial, as it required authors of all works deriving from a BSD-licensed work to include an acknowledgment of the original source in all advertising material.

This clause was objected to on the grounds that as people changed the license to reflect their name or organization it led to escalating advertising requirements when programs were combined together in a software distribution—every occurrence of the license with a different name required a separate acknowledgment. In arguing against it, Richard Stallman has stated that he counted 75 such acknowledgments in a 1997 version of NetBSD. In addition, the clause presented a legal problem for those wishing to publish BSD-licensed software which relies upon separate programs using the more-restrictive GNU GPL: the advertising clause is incompatible with the GPL, which does not allow the addition of restrictions beyond those it already imposes; because of this, the GPL's publisher, the Free Software Foundation, recommends developers not use the license, though it states there is no reason not to use software already using it. Today, this original license is now sometimes called "**BSD-old**" or "4-clause BSD".

The BSD 3-clause license

("Revised BSD License", "New BSD License", or "Modified BSD License")
The advertising clause was removed from the license text in the official BSD on 22 July 1999 by William Hoskins, Director of the Office of Technology Licensing for UC Berkeley. Other BSD distributions removed the clause, but many similar clauses remain in BSD-derived code from other sources, and unrelated code using a derived license.

While the original license is sometimes referred to as "**BSD-old**", the resulting 3-clause version is sometimes referred to by "**BSD-new**." Other names include "New BSD", "revised BSD", "BSD-3", or "3-clause BSD". This version has been vetted as an Open source license by the OSI as "The BSD License". The Free Software Foundation, which refers to the license as the "Modified BSD License", states that it is compatible with the GNU GPL. The FSF encourages users to be specific when referring to the license by name (i.e. not simply referring to it as "a BSD license" or "BSD-style") to avoid confusion with the original BSD license.

This version allows unlimited redistribution for any purpose as long as its copyright notices and the license's disclaimers of warranty are maintained. The license also contains a clause restricting use of the names of contributors for endorsement of a derived work without specific permission.

License terms

Copyright (c) <year>, <copyright holder>

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
* Neither the name of the <organization> nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO

Taken from: http://en.wikipedia.org/wiki/BSD_licenses

Appendix B

The MIT licence

The MIT License is a free software license originating at the Massachusetts Institute of Technology (MIT). It is a permissive free software license, meaning that it permits reuse within proprietary software provided all copies of the licensed software include a copy of the MIT License terms. Such proprietary software retains its proprietary nature even though it incorporates software under the MIT License. The license is also GPL-compatible, meaning that the GPL permits combination and redistribution with software that uses the MIT License.

License terms

A common form of the MIT License (from OSI's official site, which is the same version as the "Expat License", and which is not identical to the X source code) is defined as follows:

> Copyright (C) <year> <copyright holders>
>
> Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to

Taken from: http://en.wikipedia.org/wiki/MIT_License

Appendix C

The Mozilla Public Licence Version 2

The Mozilla Public License (MPL) is a free, open source, and detailed software license developed and maintained by the Mozilla Foundation. It is characterized as a hybridization of the modified BSD license and GNU General Public License (GPL) that seeks to balance the concerns of proprietary and open source developers.

It has undergone two revisions, most recently to version 2.0 with the goals of greater simplicity and better compatibility with other licenses.

The MPL is the license for the Mozilla Firefox, Mozilla Thunderbird, and most other Mozilla software, but it has been used by others, such as Adobe to license their Flex product line, and LibreOffice 4.0 (also on LGPL 3+). Version 1.1 was also notably adapted by companies to form derivative licenses like Sun Microsystems' own Common Development and Distribution License.

Unlike strong copyleft licenses, code under the MPL may be combined with files under any license in a "larger work", so long as conditions for the MPL are still met for "covered" components (Section 3.3 of the license). The MPL treats the source code file as the boundary between MPL-licensed and proprietary parts, meaning that all or none of the code in a given source file falls under the MPL.

MPL version 2.0 is compatible with both the Apache License and by default "the GNU GPL version 2.0, the GNU LGPL version 2.1, the GNU AGPL version 3.0, and all later versions of those licenses". Version 1.1 had "some complex restrictions" that made it incompatible with the GPL by default (and thus preventing updating to the MPL 2.0). Although the MPL 1.1 did include a provision (Section 13) for providing a work under a secondary license (including the GPL or GPL-compatible ones), MPL 1.1 and GPL code could not "legally be linked," leading the Free Software Foundation to discourage using the MPL 1.1. For these reasons, earlier versions of Firefox were released under multiple licenses: the MPL 1.1, GPL 2.0, and LGPL 2.1.

License terms

Mozilla Public License

Version 2.0

1. Definitions

1.1. “Contributor”

means each individual or legal entity that creates, contributes to the creation of, or owns Covered Software.

1.2. “Contributor Version”

means the combination of the Contributions of others (if any) used by a Contributor and that particular Contributor’s Contribution.

1.3. “Contribution”

means Covered Software of a particular Contributor.

1.4. “Covered Software”

means Source Code Form to which the initial Contributor has attached the notice in Exhibit A, the Executable Form of such Source Code Form, and Modifications of such Source Code Form, in each case including portions thereof.

1.5. “Incompatible With Secondary Licenses”

means

a. that the initial Contributor has attached the notice described in Exhibit B to the Covered Software; or

b. that the Covered Software was made available under the terms of version 1.1 or earlier of the License, but not also under the terms of a Secondary License.

1.6. “Executable Form”

means any form of the work other than Source Code Form.

1.7. “Larger Work”

means a work that combines Covered Software with other material, in a separate file or files, that is not Covered Software.

1.8. “License”

means this document.

1.9. “Licensable”

means having the right to grant, to the maximum extent possible, whether at the time of the initial grant or subsequently, any and all of the rights conveyed by this License.

1.10. “Modifications”

means any of the following:

a. any file in Source Code Form that results from an addition to, deletion from, or modification of the contents of Covered Software; or

b. any new file in Source Code Form that contains any Covered Software.

1.11. “Patent Claims” of a Contributor

means any patent claim(s), including without limitation, method, process, and apparatus claims, in any patent Licensable by such Contributor that would be infringed, but for the grant of the License, by the making, using, selling, offering for sale, having made, import, or transfer of either its Contributions or its Contributor Version.

1.12. “Secondary License”

means either the GNU General Public License, Version 2.0, the GNU Lesser General Public License, Version 2.1, the GNU Affero General Public License, Version 3.0, or any later versions of those licenses.

1.13. “Source Code Form”

means the form of the work preferred for making modifications.

1.14. “You” (or “Your”)

means an individual or a legal entity exercising rights under this License. For legal entities, “You” includes any entity that controls, is controlled by, or is under common control with You. For purposes of this definition, “control” means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

2. License Grants and Conditions

2.1. Grants

Each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license:

a. under intellectual property rights (other than patent or trademark) Licensable by such Contributor to use, reproduce, make available, modify, display, perform, distribute, and otherwise exploit its Contributions, either on an unmodified basis, with Modifications, or as part of a Larger Work; and

b. under Patent Claims of such Contributor to make, use, sell, offer for sale, have made, import, and otherwise transfer either its Contributions or its Contributor Version.

2.2. Effective Date

The licenses granted in Section 2.1 with respect to any Contribution become effective for each Contribution on the date the Contributor first distributes such Contribution.

2.3. Limitations on Grant Scope

The licenses granted in this Section 2 are the only rights granted under this License. No additional rights or licenses will be implied from the distribution or licensing of Covered Software under this License. Notwithstanding Section 2.1(b) above, no patent license is granted by a Contributor:

a. for any code that a Contributor has removed from Covered Software; or

b. for infringements caused by: (i) Your and any other third party’s modifications of Covered Software, or (ii) the combination of its Contributions with other software (except as part of its Contributor Version); or

c. under Patent Claims infringed by Covered Software in the absence of its Contributions.

This License does not grant any rights in the trademarks, service marks, or logos of any Contributor (except as may be necessary to comply with the notice requirements in Section 3.4).

2.4. Subsequent Licenses

No Contributor makes additional grants as a result of Your choice to distribute the Covered Software under a subsequent version of this License (see Section 10.2) or under the terms of a Secondary License (if permitted under the terms of Section 3.3).

2.5. Representation

Each Contributor represents that the Contributor believes its Contributions are its original creation(s) or it has sufficient rights to grant the rights to its Contributions conveyed by this License.

2.6. Fair Use

This License is not intended to limit any rights You have under applicable copyright doctrines of fair use, fair dealing, or other equivalents.

2.7. Conditions

Sections 3.1, 3.2, 3.3, and 3.4 are conditions of the licenses granted in Section 2.1.

3. Responsibilities

3.1. Distribution of Source Form

All distribution of Covered Software in Source Code Form, including any Modifications that You create or to which You contribute, must be under the terms of this License. You must inform recipients that the Source Code Form of the Covered Software is governed by the terms of this License, and how they can obtain a copy of this License. You may not attempt to alter or restrict the recipients' rights in the Source Code Form.

3.2. Distribution of Executable Form

If You distribute Covered Software in Executable Form then:

a. such Covered Software must also be made available in Source Code Form, as described in Section 3.1, and You must inform recipients of the Executable Form how they can obtain a copy of such Source Code Form by reasonable means in a timely manner, at a charge no more than the cost of distribution to the recipient; and

b. You may distribute such Executable Form under the terms of this License, or sublicense it under different terms, provided that the license for the Executable Form does not attempt to limit or alter the recipients' rights in the Source Code Form under this License.

3.3. Distribution of a Larger Work

You may create and distribute a Larger Work under terms of Your choice, provided that You also comply with the requirements of this License for the Covered Software. If the Larger Work is a combination of Covered Software with a work governed by one or more Secondary Licenses, and the Covered Software is not Incompatible With Secondary Licenses, this License permits You to additionally distribute such Covered Software under the terms of such Secondary License(s), so that the recipient of the Larger Work may, at their option, further distribute the Covered Software under the terms of either this License or such Secondary License(s).

3.4. Notices

You may not remove or alter the substance of any license notices (including copyright notices, patent notices, disclaimers of warranty, or limitations of liability) contained within the Source Code Form of the Covered Software, except that You may alter any license notices to the extent required to remedy known factual inaccuracies.

3.5. Application of Additional Terms

You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Software. However, You may do so only on Your own behalf, and not on behalf of any Contributor. You must make it absolutely clear that any such warranty, support, indemnity, or liability obligation is offered by You alone, and You hereby agree to indemnify every Contributor for any liability incurred by such Contributor as a result of warranty, support, indemnity or liability terms You offer. You may include additional disclaimers of warranty and limitations of liability specific to any jurisdiction.

4. Inability to Comply Due to Statute or Regulation

If it is impossible for You to comply with any of the terms of this License with respect to some or all of the Covered Software due to statute, judicial order, or regulation then You must: (a) comply with the terms of this License to the maximum extent possible; and (b) describe the limitations and the code they affect. Such description must be placed in a text file included with all distributions of the Covered Software under this License. Except to the extent prohibited by statute or regulation, such description must be sufficiently detailed for a recipient of ordinary skill to be able to understand it.

5. Termination

5.1. The rights granted under this License will terminate automatically if You fail to comply with any of its terms. However, if You become compliant, then the rights granted under this License from a particular Contributor are reinstated (a) provisionally, unless and until such Contributor explicitly and finally terminates Your grants, and (b) on an ongoing basis, if such Contributor fails to notify You of the non-compliance by some reasonable means prior to 60 days after You have come back into compliance. Moreover, Your grants from a particular Contributor are reinstated on an ongoing basis if such

Contributor notifies You of the non-compliance by some reasonable means, this is the first time You have received notice of non-compliance with this License from such Contributor, and You become compliant prior to 30 days after Your receipt of the notice.

5.2. If You initiate litigation against any entity by asserting a patent infringement claim (excluding declaratory judgment actions, counter-claims, and cross-claims) alleging that a Contributor Version directly or indirectly infringes any patent, then the rights granted to You by any and all Contributors for the Covered Software under Section 2.1 of this License shall terminate.

5.3. In the event of termination under Sections 5.1 or 5.2 above, all end user license agreements (excluding distributors and resellers) which have been validly granted by You or Your distributors under this License prior to termination shall survive termination.

6. Disclaimer of Warranty

Covered Software is provided under this License on an “as is” basis, without warranty of any kind, either expressed, implied, or statutory, including, without limitation, warranties that the Covered Software is free of defects, merchantable, fit for a particular purpose or non-infringing. The entire risk as to the quality and performance of the Covered Software is with You. Should any Covered Software prove defective in any respect, You (not any Contributor) assume the cost of any necessary servicing, repair, or correction. This disclaimer of warranty constitutes an essential part of this License. No use of any Covered Software is authorized under this License except under this disclaimer.

7. Limitation of Liability

Under no circumstances and under no legal theory, whether tort (including negligence), contract, or otherwise, shall any Contributor, or anyone who distributes Covered Software as permitted above, be liable to You for any direct, indirect, special, incidental, or consequential damages of any character including, without limitation, damages for lost profits, loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses, even if such party shall have been informed of the possibility of such damages. This limitation of liability shall not apply to liability for death or personal injury resulting from such party's negligence to the extent applicable law prohibits such limitation. Some jurisdictions do not allow the exclusion or limitation of incidental or consequential damages, so this exclusion and limitation may not apply to You.

8. Litigation

Any litigation relating to this License may be brought only in the courts of a jurisdiction where the defendant maintains its principal place of business and such litigation shall be governed by laws of that jurisdiction, without reference to its conflict-of-law provisions. Nothing in this Section shall prevent a party's ability to bring cross-claims or counter-claims.

9. Miscellaneous

This License represents the complete agreement concerning the subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not be used to construe this License against a Contributor.

10. Versions of the License

10.1. New Versions

Mozilla Foundation is the license steward. Except as provided in Section 10.3, no one other than the license steward has the right to modify or publish new versions of this License. Each version will be given a distinguishing version number.

10.2. Effect of New Versions

You may distribute the Covered Software under the terms of the version of the License under which You originally received the Covered Software, or under the terms of any subsequent version published by the license steward.

10.3. Modified Versions

If you create software not governed by this License, and you want to create a new license for such software, you may create and use a modified version of this License if you rename the license and remove any references to the name of the license steward (except to note that such modified license differs from this License).

10.4. Distributing Source Code Form that is Incompatible With Secondary Licenses

If You choose to distribute Source Code Form that is Incompatible With Secondary Licenses under the terms of this version of the License, the notice described in Exhibit B of this License must be attached.

Exhibit A - Source Code Form License Notice

```
This Source Code Form is subject to the
terms of the Mozilla Public License, v.
2.0. If a copy of the MPL was not
distributed with this file, You can obtain
one at http://mozilla.org/MPL/2.0/.
```

If it is not possible or desirable to put the notice in a particular file, then You may include the notice in a location (such as a LICENSE file in a relevant directory) where a recipient would be likely to look for such a notice.

You may add additional accurate notices of copyright ownership.

Exhibit B - “Incompatible With Secondary Licenses” Notice

```
This Source Code Form is “Incompatible With
Secondary Licenses”, as defined by the
Mozilla Public License, v. 2.0.
```

Taken from: http://en.wikipedia.org/wiki/Mozilla_Public_License
and: http://www.mozilla.org/MPL/2.0/

Appendix D

The GNU General Public Licence Version 2

The GNU General Public License (GNU GPL or GPL) is the most widely used free software license, which guarantees end users (individuals, organizations, companies) the freedoms to use, study, share (copy), and modify the software. Software that allows these rights is called free software and if the software is copyleft ensures those are retained. The GPL demands both. The license was originally written by Richard Stallman of the Free Software Foundation (FSF) for the GNU project.

In other words, the GPL grants the recipients of a computer program the rights of the Free Software Definition and uses copyleft to ensure the freedoms are preserved whenever the work is distributed, even when the work is changed or added to. The GPL is a copyleft license, which means that derived works can only be distributed under the same license terms. This is in distinction to permissive free software licenses, of which the BSD licenses and the MIT License are the standard examples. GPL was the first copyleft license for general use. Prominent free software programs licensed under the GPL include the Linux kernel and the GNU Compiler Collection (GCC). Some other free software programs (MySQL is a prominent example) are dual-licensed under multiple licenses, often with one of the licenses being the GPL.

It is believed that the copyleft provided by the GPL was crucial to the success of Linux-based systems, giving the programmers who contributed to the kernel the assurance that their work would benefit the whole world and remain free, rather than being exploited by software companies that would not have to give anything back to the community.

Licence Terms

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

> **a)** Accompany it with the complete corresponding machine-readable source code, which must be distributed under the

terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN

OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year> <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by t the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of

MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program `Gnomovision' (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989
Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License.

Taken from: http://en.wikipedia.org/wiki/GNU_General_Public_License
and: http://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html

Appendix E

The GNU General Public Licence Version 3

On 29 June 2007, the third version of the license (GNU GPLv3) was released to address some perceived problems with the second version (GNU GPLv2) that were discovered during its long-time usage. To keep the license up to date the GPL license includes an optional "any later version" clause, allowing users to choose between the original terms or the terms in new versions as updated by the FSF. Developers can omit it when licensing their software; for instance the Linux kernel is licensed under GPLv2 without the "any later version" clause.

Licence Terms

GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the

Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS"

WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.>

Copyright (C) <year> <name of author>

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

<program> Copyright (C) <year> <name of author>
This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'. This is free software, and you are welcome to redistribute it  under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <http://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <http://www.gnu.org/philosophy/why-not-lgpl.html>.

Taken from: http://en.wikipedia.org/wiki/GNU_General_Public_License
and: http://www.gnu.org/licenses/gpl.html

Appendix F

Public Domain

Works in the public domain are those whose intellectual property rights have expired, have been forfeited, or are inapplicable. The term is not normally applied to situations where the creator of a work retains residual rights, in which case use of the work is referred to as "under license" or "with permission".

In informal usage, the public domain consists of works that are publicly available; while according to the formal definition, it consists of works that are unavailable for private ownership or are available for public use. As rights are country-based and vary, a work may be subject to rights in one country and not in another. Some rights depend on registrations with a country-by-country basis, and the absence of registration in a particular country, if required, implies public domain status in that country.

Public Domain is one of the traditional safety valves in copyright law.

The term public domain did not come into use until the mid-17th century, although as a concept "it can be traced back to the ancient Roman Law, as a preset system included in the property right system." The Romans had a large proprietary rights system where they defined "many things that cannot be privately owned" as res communes, res publicae and res universitatis. The term res commune was defined as "things that could be commonly enjoyed by mankind, such as air, sunlight and ocean." The term res publicae referred to things that were shared by all citizens, and the term res universitatis meant things that were owned by the municipalities of Rome. When looking at the public domain from a historical perspective, one could say the construction of the idea of "public domain" sprouted from the concepts of res commune, res publicae, and res universitatis in early Roman Law.

When the first early copyright law was first established in Britain with the Statute of Anne in 1710, public domain did not appear. However, similar concepts were developed by British and French jurists in the eighteenth century. Instead of "public domain" they used terms such as publici juris or propriété publique to describe works that were not covered by copyright law. The phrase "fall in the public domain" can be traced to mid-nineteenth century France to describe the end of copyright term. The French poet Alfred de Vigny equated the expiration of copyright with a work falling "into the sink hole of the public domain" and if the public domain receives any attention from intellectual property lawyers it is still treated as little more than that which is left when intellectual property rights, such as copyright, patents, and trademarks, expire or are abandoned. In this historical context Paul Torremans describes copyright as a "little coral reef of private right jutting up from the ocean of the public domain." Because copyright law is different from country to country, Pamela Samuelson has described the public domain as being "different sizes at different times in different countries".

Definition

Definitions of the boundaries of the public domain in relation to copyright, or intellectual property more generally, regard the public domain as a negative space, that is, it consists of works that are no longer in copyright term or were never protected by copyright law. According to James Boyle this definition underlines common usage of the term public domain and equates the public domain to public property and works in copyright to private property. However, the usage of the term public domain can be more granular, including for example uses of works in copyright permitted by copyright exceptions. Such a definition regards work in copyright as private property subject to fair use rights and limitation on ownership. A conceptual definition comes from Lange, who focused on what the public domain should be: "it should be a place of sanctuary for individual creative expression, a sanctuary conferring affirmative protection against the forces of private appropriation that threatened such expression". Patterson and Lindberg described the public domain not as a "territory", but rather as a concept: "[T]here are certain materials – the air we breathe, sunlight, rain, space, life, creations, thoughts, feelings, ideas, words, numbers – `not subject to private ownership. The materials that compose our cultural heritage must be free for all living to use no less than matter necessary for biological survival." The term public domain may also be interchangeably used with other imprecise and/or undefined terms such as the "public sphere" or "commons", including concepts such as "commons of the mind", the "intellectual commons", and the "information commons".

Value

Pamela Samuelson has identified eight "values" that can arise from information and works in the public domain.

Possible values include:

1. Building blocks for the creation of new knowledge, examples include data, facts, ideas, theories, and scientific principle.
2. Access to cultural heritage through information resources such as ancient Greek texts and Mozart’s symphonies.
3. Promoting education, through the spread of information, ideas, and sci-

entific principles.

4. Enabling follow-on innovation, through for example expired patents and copyright.
5. Enabling low cost access to information without the need to locate the owner or negotiate rights clearance and pay royalties, through for example expired copyrighted works or patents, and non-original data compilation.
6. Promoting public health and safety, through information and scientific principles.
7. Promoting the democratic process and values, through news, laws, regulation, and judicial opinion.
8. Enabling competitive imitation, through for example expired patents and copyright, or publicly disclosed technologies that do not qualify for patent protection.

Dedicating works to the public domain

Few if any legal systems have a process for reliably donating works to the public domain. They may even prohibit any attempt by copyright owners to surrender rights automatically conferred by law, particularly moral rights. An alternative is for copyright holders to issue a licence which irrevocably grants as many rights as possible to the general public, e.g., the CC0 licence from Creative Commons.

Continantal-european law

Public Domain is a legal term of the anglo-american law. It is similar, but not identical to the continental-european “Gemeinfreiheit”

For further informations about “Gemeinfreiheit” the reader may refer to:
http://de.wikipedia.org/wiki/Gemeinfreiheit

Taken from: http://en.wikipedia.org/wiki/Public_domain