



# Origami Diagramming

Evaluating and Improving the Origami  
Diagramming Tool Origrammer

BACHELOR THESIS

by

Julian Hardtung  
Matr.-Nr.: 11104591

at

TH KÖLN UNIVERSITY OF APPLIED SCIENCES  
CAMPUS GUMMERSBACH  
INSTITUTE OF INFORMATICS AND ENGINEERING

Course of Studies  
MEDIA INFORMATICS

First supervisor: Prof. Dr. Martin Eisemann  
TH Köln University of Applied Sciences

Second supervisor: Prof. Dr. Christian Kohls  
TH Köln University of Applied Sciences

Gummersbach, September 22, 2020

**Adresses:** Julian Hardtung  
Lachtstraße 12  
51645 Gummersbach  
+49 1578 7080524  
[ju.hardtung@gmx.de](mailto:ju.hardtung@gmx.de)

Prof. Dr. Martin Eisemann  
TH Köln University of Applied Sciences  
Institute of Informatics and Engineering  
Steinmüllerallee 1  
51643 Gummersbach  
[martin.eisemann@th-koeln.de](mailto:martin.eisemann@th-koeln.de)

Prof. Dr. Christian Kohls  
TH Köln University of Applied Sciences  
Institute of Informatics and Engineering  
Steinmüllerallee 1  
51643 Gummersbach  
[christian.kohls@th-koeln.de](mailto:christian.kohls@th-koeln.de)

## Contents

<b>1 Abstract</b>	<b>4</b>
<b>2 Introduction</b>	<b>5</b>
<b>3 Origrammer</b>	<b>7</b>
3.1 Menu Bar . . . . .	8
3.2 Editing Panel . . . . .	9
3.3 Side Panel . . . . .	9
3.4 Top Panel . . . . .	10
3.5 Navigation Panel . . . . .	10
<b>4 Evaluating Origrammer</b>	<b>11</b>
4.1 Origrammer Feature List . . . . .	12
4.2 10 Usability Heuristics . . . . .	13
<b>5 Changes and new Features</b>	<b>18</b>
5.1 Graphical User Interface related Changes . . . . .	18
5.1.1 Overall GUI Changes . . . . .	18
5.1.2 Side Panel . . . . .	19
5.2 User Input related Changes . . . . .	19
5.2.1 Arrows and Symbols . . . . .	19
5.2.2 Dynamic Input Previews . . . . .	20
5.2.3 Filling Tool . . . . .	21
5.3 New Origrammer Features . . . . .	22
5.3.1 Step Navigation and Step Editing Options . . . . .	22
5.3.2 Folding Presets . . . . .	23
5.3.3 Additional Line Input Types . . . . .	25
<b>6 Virtual Folding of the Paper</b>	<b>26</b>
6.1 Related Works . . . . .	27
6.1.1 Origami Simulation . . . . .	27
6.1.2 Foldinator . . . . .	28
6.1.3 eGami . . . . .	29
6.1.4 Conclusion . . . . .	30
6.2 Requirements . . . . .	31
6.3 Current Status . . . . .	31

6.4 Reworked System . . . . .	32
6.4.1 Current Limitations . . . . .	36
<b>7 Problems and other Findings</b>	<b>39</b>
7.1 Virtual Folding . . . . .	39
7.1.1 Folding over and over . . . . .	39
7.2 Folding multiple layers . . . . .	41
<b>8 Outlook</b>	<b>42</b>
<b>9 Declaration of Authorship</b>	<b>43</b>
<b>Glossary</b>	<b>44</b>

## 1 Abstract

The [Origrammer](#) is a desktop application for creating origami instructions, also called origami [Diagrams](#). The goal of this program is to maximize the efficiency of creating these diagrams, as the process is slow and tedious when creating them by hand. In this paper the current state of this program is evaluated using the 10 usability heuristics by Jakob Nielsen [1] with particular focus on flaws that reduce efficiency and speed of the diagram creation. With the identified flaws as a basis the Origrammer was further developed and improved.

A more flexible navigation through the folding steps with editing functions and step preview pictures was added. To further maximize efficiency, the whole system of rendering the paper was reworked as well. The Origrammer can now virtually fold paper, without the need to place every single edge or crease line by hand. Simple [Mountain Folds](#) and [Valley Folds](#) can be carried out and the paper folds automatically.

Various additional improvements were introduced, like folding presets for similar beginnings of an origami model, a reworked arrow system that is more flexible and improves on several issues, more line input variants, and a dynamic preview when inputting lines, arrows, or symbols.

## 2 Introduction

**Origami** is the Japanese art of folding paper into models of animals, people, or other objects. Folding such models can be a complex task, so in order to make origami more accessible, a system of folding instructions was developed. The diagramming system developed by Akira Yoshizawa was first published in the book *Atarashi origami geijutsu* in 1954 [2]. Yoshizawa's origami notation introduced a set of symbols and arrows to explain the different folding actions. After further development by Sam Randlett and Robert Harbin this set of origami symbols became the international standard [3] [4]. By creating these drawings for each folding step, even origami beginners could fold models. These origami diagrams (see Figure 2.1) have to be accurate representations of the paper for every folding step, in order to unambiguously explain how to fold the model. Every flap, crease and edge has to be drawn, which makes the process slow and especially error-prone for complex models. While for example the crane model in Figure 2.1 can be folded in 17 steps, more complex models can take hundreds of steps to complete.

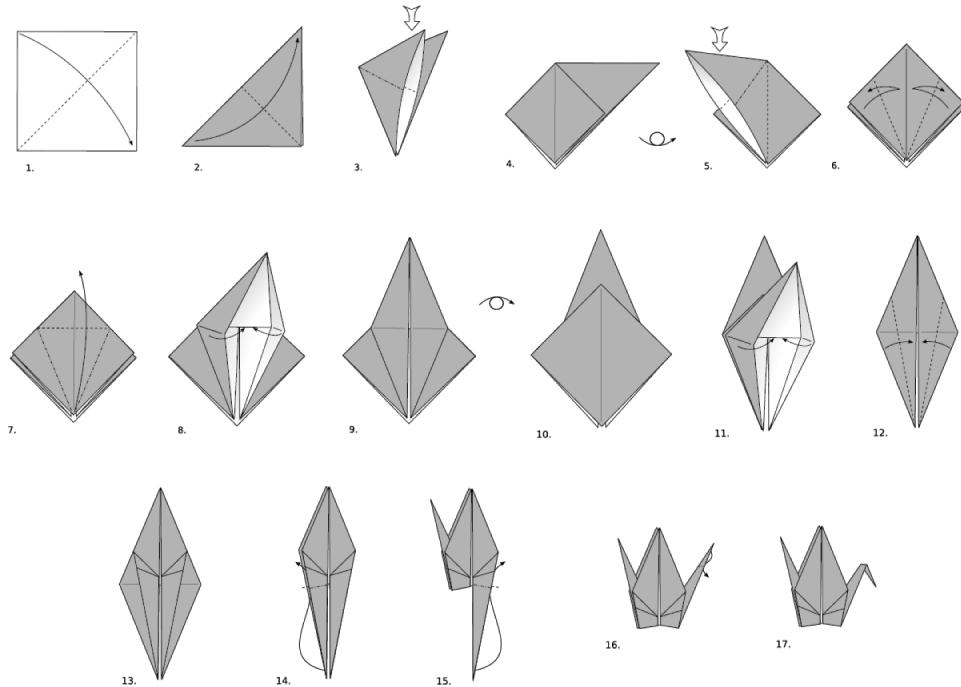


Figure 2.1: Crane Diagram - [Andrew Hudson, 2011 [5]]

In order to help artists during the creation process for origami diagrams, the [Origrammer](#) [6] was developed.

The Origrammer is a desktop application, developed by the author, which offers specific features for creating such diagrams. This program simulates the manual diagram creation process of an origami artist. Lines, arrows, and symbols can be placed and new diagram steps can be created. Additionally, helping features are included to speed up the diagramming process as a whole. A more detailed overview and explanation of the Origrammer is presented in Section 3.

This Bachelor Thesis now aims to evaluate the current state of this program and to develop it further to improve usability, efficiency, and effectiveness. This thesis starts by presenting a short summary of the Origrammer's functionality and operations, as well as displaying a typical workflow. Afterwards, an evaluation has to be carried out, to get a sense of the current state of this program. Flaws and missing features have to be explicitly defined, so that a plan can be created on how to further improve the Origrammer.

As the evaluation will potentially find a multitude of different types of flaws and problems, a specific focus has to be defined. In the case of this thesis, the main goal is to maximise the speed of creating origami diagrams, while also offering a usable user interface with which also novices can work with ease. This target was set with the original plan of the Origrammer in mind:

*“The goal for this project is to develop a desktop application that implements features specifically for the origami diagramming process. The standardized symbols and overall notations have to be included and the program has to offer functions that increase the efficiency of creating diagrams.”* (Hardtung, 2020, p.5 [6])

Then after evaluating and collecting potential improvements, a more detailed plan can be created on how to actually implement these changes. Although not all potential improvements can be implemented within the scope of this thesis, all identified flaws should be categorized and documented, which can help and facilitate further development in the future.

### 3 Origrammer

This section will provide a more detailed overview of the Origrammer. Its components will be explained and a typical workflow will be illustrated. The Origrammer is desktop application which is being developed in Java and uses the libraries Apache Commons IO<sup>1</sup> and javax.vecmath<sup>2</sup>. Figure 3.1 shows the user interface of the Origrammer, which is split up into five major parts.

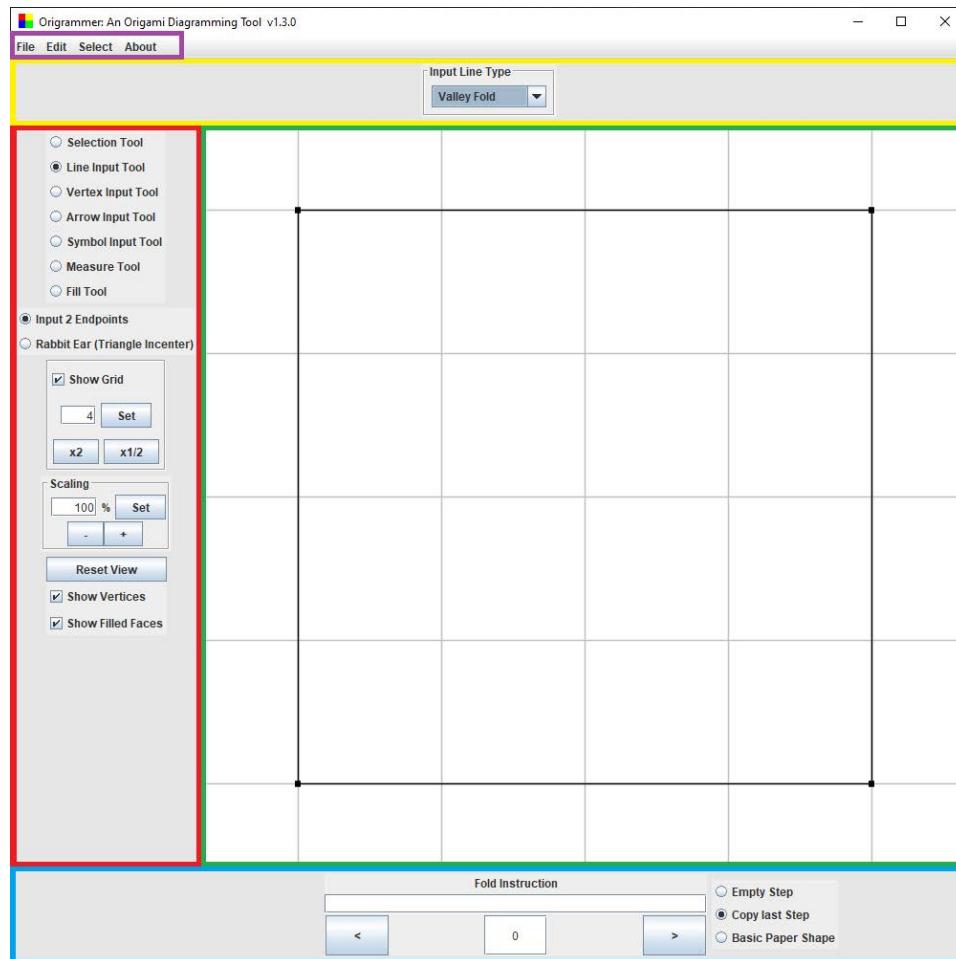


Figure 3.1: Menu Bar (Purple), Top Panel (Yellow), Side Panel (Red), Editing Panel (Green), Navigation Panel (Blue)

<sup>1</sup><https://commons.apache.org/proper/commons-io/>

<sup>2</sup><https://github.com/egonw/vecmath>

### 3.1 Menu Bar

The Menu Bar offers fast and easy access to diagram-wide functions. Under **File**, the user can create a new diagram, open or save an already existing Origrammer diagram, and *export a diagram to a different format (not yet fully implemented)*. When creating a new diagram, the following panel appears, in which several settings for the diagram can be adjusted.

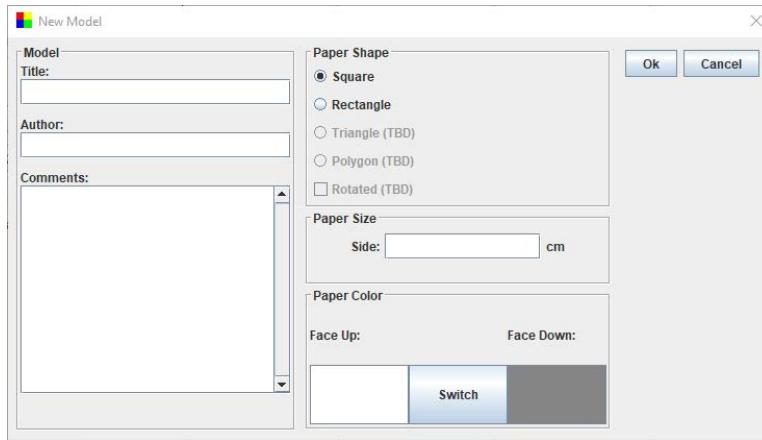


Figure 3.2: Options when creating a new diagram

The user is able to provide a **title**, an **author**, and **additional comments** (regarding copyright information, recommended paper material, or others). Additionally, paper-related settings can be changed. The required paper shape can be defined (square, rectangle, triangle, polygon, or pre-rotated), a recommended paper size can be suggested, and colours for both sides of the paper can be specified.

After confirming the settings for the new diagram, the paper outline of the first diagram step appears on the [3.2 Editing Panel](#).

The next button on the Menu Bar (**Edit**) includes the typical editing features **Undo** (Ctrl+Z), **Redo** (Ctrl+Y), **Cut** (Ctrl+X), **Copy** (Ctrl+C), **Paste** (Ctrl+V), and **Delete** selection (Delete). Additionally, the previously specified diagram information can be changed and edited at any point under **Model Preferences** and some Origrammer-wide options can be adjusted under **Origrammer Preferences** (see Fig. [3.3](#)).

Lastly, under the **Select** button of the Menu Bar, the user can **Select all** (Ctrl+A) or **Unselect all** (Ctrl+Shift+A) objects on a given diagram step.

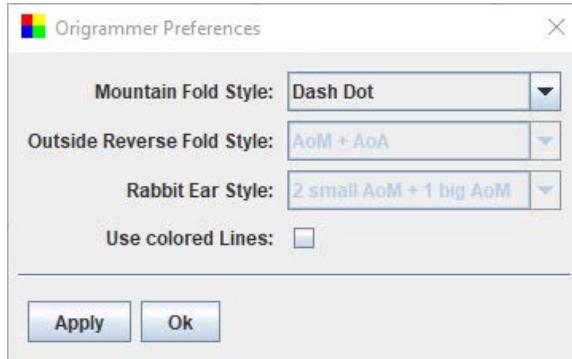


Figure 3.3: Origrammer Preferences

### 3.2 Editing Panel

The Editing Panel shows the diagram step currently being worked on. All lines, arrows, and other symbols are being placed and edited here and all tools and input functions can then be used on the Editing Panel. Furthermore, the working grid is being displayed here, which can be adjusted on the Side Panel and offers input points for all lines, arrows, and other symbols.

### 3.3 Side Panel

All major input tools and adjustments of the Editing Panel are displayed on the Side Panel. The input tools consist of the **Selection Tool**, the **Line Input**, **Vertex Input**, **Arrow Input**, **Symbol Input**, **Measure Tool**, and **Fill Tool**.

Through the **Selection Tool**, the user can select, move, and edit already placed objects on a diagram step. The **Line**, **Vertex**, **Arrow**, and **Symbol Input Tools** make it possible to place the respective objects on a given step. Moreover, the user can measure the angle and length of lines within the diagram, in order to use these values as input for additional creases. The last tool is the **Filling Tool**, which offers the function to color parts of the paper, according to the specified paper colour.

Furthermore, the options for the assisting grid of the Editing Panel can be changed, the scaling can be adjusted to see more detailed parts of a diagram step, and the rendering for vertices and the coloured faces of the diagram can be turned off.

### 3.4 Top Panel

After choosing a tool from the Side Panel, the respective input/editing options of the activated tool are being displayed on the Top Panel. These can be for example sliders for rotations, check boxes for mirroring a symbol, combo boxes for choosing different line/arrow types, as well as other features.

### 3.5 Navigation Panel

The Navigation Panel makes it possible to move through the individual diagram steps, as well as to create new ones. When creating a new step, the user has three different options. The new step can either be completely empty, can only consist of the original paper outlines, or can be a full copy of the previous step.

This is also the area, where the user can write down the textual folding instructions for a certain step.

## 4 Evaluating Origrammer

This evaluation of the Origrammer is supposed to give a basic feedback on which parts of the Origrammer are the most problematic and could benefit from automating parts of their processes. The main metrics of the evaluation will be efficiency or speed of use, which can be measured and systematically determined. Consequently, user involvement in the evaluation is at this stage of development not of significant impact and therefore not applied. Nonetheless, an evaluation with real users should be carried out at a later time before the final release of the program, as even though the duration of certain processes will be reduced during this work, some usability issues can still be overlooked.

As a result of the aforementioned reasons, the evaluation will be carried out using the *10 usability heuristics* by Jakob Nielsen [1]. These heuristics offer clear reference points to accurately evaluate the Origrammer, though this approach also brings limitations. This is mentioned by Nielsen and Molich in the proceedings of the SIGCHI Conference on Human Factors in Computing Systems from 1990.

*“A disadvantage of the method is that it sometimes identifies usability problems without providing direct suggestions for how to solve them.”*

(Nielsen and Molich, 1990, p. 255 [7])

This limitation of not providing direct solutions for the identified flaws is why this chapter will focus on finding problems and shortcomings of the Origrammer first. Afterwards, solutions can be developed that optimally address and fix most, or all, discovered issues without contradicting or counteracting other measures. These solutions should prioritize efficiency, giving a clear metric on which one of multiple changes should be implemented.

As I intentionally refrain from end user involvement for this evaluation process, other measures have to be applied to ensure maximum efficiency and thoroughness. This is why the following section shows an exhaustive list of all parts and features of the Origrammer. Figure 3.1 briefly displays the features contained in the Origrammer and their respective location.

## 4.1 Origrammer Feature List<sup>3</sup>

### 1. Menu Bar (Purple)

- (a) New File      (b) Open File      (c) Save File
- (d) *Export File*      (e) Model Preferences      (f) Origrammer Preferences

### 2. Side Bar (Red)

- (a) Selection Tool
  - 1. Click to Select    2. Hover over Object    3. Rectangular Selection
- (b) Line Input
  - 1. Two Point Input    2. Triangle Incenter
- (c) Vertex Input
  - 1. Absolute Position    2. Splitting an existing Line at a specified fraction
- (d) Arrow Input
  - 1. Valley Fold    2. Mountain Fold    3. Turn over
  - 4. Push Here    5. Pull out    6. Inflate here
- (e) Symbol Input
  - 1. Leader    2. Repetition Box    3. Next View Here
  - 4. Rotations    5. Hold Here    6. Hold Here and Pull
  - 7. X-Ray Circle    8. *Fold Over & Over*    9. Equal Distances
  - 10. Equal Angles    11. Crimps    12. Pleats
  - 13. Closed Sinks
- (f) Measure Tool
  - 1. Measure Length    2. Measure Angle
- (g) Fill Tool
- (h) Grid Settings
- (i) Scaling Settings

### 3. Navigation Panel (Blue)

- 1. Fold Instructions    2. Step Navigation    3. New Step Options

### 4. Top Panel (Yellow)

See Side Panel for related features, as options for the selected Side Panel Tools appear on the Top Panel.

### 5. Editing Panel (Green)

See Side Panel for related features, as the selected Side Panel Tools are being used on the Editing Panel.

---

<sup>3</sup>Cursive entries are not yet fully implemented features in the Origrammer

## 4.2 10 Usability Heuristics

Based on the Origrammer Feature List, the evaluation using the 10 Usability Heuristics can be carried out. Every feature from the list will be checked against all 10 heuristics in order to try and maximise the completeness of the result. The identified usability issues will then lead to the planning of potential usability improvements. Additional focus should be set on the 7th heuristic *Flexibility and Efficiency of Use*, as it includes the main goal of improving the efficiency of creating diagrams with the Origrammer.

### 1. Visibility of System Status

Nr:	Affects	Impact	Description
1.01	2.b; 2.d-2.g	3	When an action requires multiple input points (e.g. placing a Fold Line by two endpoints), the user is not aware of his current state within the process.
1.02	2.d; 2.e	7	Always show a preview of the <code>OriArrow</code> or <code>OriSymbol</code> before final placement of these objects within the diagram
1.03	3.2; 3.3	6	The user doesn't know the overall amount of diagram steps and the current step within the process flow.

### 2. Aesthetic and Minimalist Design

Nr:	Affects	Impact	Description
2.01	2.a-2.i	4	The text in the Tool Selection on the Side Panel should be replaced by icons (with additional tooltips)
2.02	4	3	The text in the TopPanel for input and editing options should be replaced by icons (with additional tooltips)
2.03	2.f; 3	6	The User Interface should always fit properly (is currently not fitting in the SidePanel for MeasureTool and for some settings in the TopPanel)

### 3. User Control and Freedom

Nr:	Affects	Impact	Description
3.01	2.a; 2.b; 2.d; 2.e; 2.g	5	Users should always be able to cancel an action (e.g., when inputting an object with multiple input points)
3.02	3.2; 3.3	8	Navigating through the diagram steps is unflexible and slow
3.03	2; 3; 4; 5;	10	Creating the individual diagram steps is slow due to the need of placing lines one by one
3.04	2.b; 5	7	Some lines are impossible to put-in with only the existing vertices and the grid (give more line input variants)

### 4. Consistency and Standards

Nr:	Affects	Impact	Description
4.01	2.a; 2.b; 2.d; 2.e; 2.g	2	Editing options on the TopPanel should be labeled correctly (consistent naming scheme needed)
4.02	2.a; 4; 5	6	Selecting and editing too many Symbols/Arrows simultaneously will overflow the TopPanel (e.g., selecting different OriObject types like OriArrows and OriSymbols at the same time)
4.03	1; 2; 3; 4	6	UI elements should have consistent sizes (e.g., “Set” Buttons, JTextFields)
4.04	1; 2; 3; 4	4	The sequential placement of UI parts should be consistent (e.g., for Input and Editing options on the TopPanel, have JTextFields first and then JCheckboxes to the right)

## 5. Error Prevention

Nr:	Affects	Impact	Description
5.01	2.b.2	7	Selecting the same point multiple times breaks the RabbitEar lines
5.02	2.e.10	8	Selecting points in the wrong order breaks inputs for the EqualAngle symbol
5.03	2.g	7	Selecting the same point multiple times breaks the FillTool
5.04	1.a; 1.e;	6	Restrict the input of every JTextField (e.g. only allow numbers for number inputs)

## 6. Recognition Rather than Recall

Nr:	Affects	Impact	Description
6.01	2.f	5	The user should not need to remember the measured values to remember the measured values (currently the measured values are being hidden, once the user selects a different tool from the SideBar)
6.02	2.b; 2.d; 2.e; 5	6	The user should not need to remember the specific input sequences for all lines, arrows, symbols (give input previews)

## 7. Flexibility and Efficiency of Use

Nr:	Affects	Impact	Description
7.01	2.h	3	Specify shortcuts for the Grid halve/ double buttons
7.02	2.i	3	Specify Give shortcuts for the Re- setView button
7.03	2.i	3	Specify shortcuts for the step-by-step zoom-in/zoom-out
7.04	2.b	8	Specify more Line Input options to avoid tedious work with grid adjust- ments
7.05	2.d; 3; 4	8	Specify more flexibility when editing or placing arrows
7.06	2.e; 3; 4	8	Specify more flexibility when editing or placing symbols
7.07	3.2; 3.3	9	The user can not change the order of steps, remove unwanted steps, or enter new steps between existing ones
7.08	2.g	7	The user has to select the vertices by hand and is limited to triangles
7.09	5	6	The user has to repeat similar dia- gram beginning steps for every new diagram
7.10	2; 3; 4; 5	10	Creating the individual diagram steps is slow because lines have to be placed seperately
7.11	5	10	The diagramming rule of distorting the model in order to see hidden layers (Lang, 2011, para. <i>Distort the model for clarity</i> [8]) is unflexible when hav- ing to change every single line by hand

## 8. Recognition, Diagnosis and Recovery from Errors

Nr:	Affects	Impact	Description
8.01	2.b-2.	5	Error messages for wrong user inputs should be self-explanatory
8.02	2.b-2.g	4	Error messages for wrong user inputs should explain how to fix the error

## 9. Help and Documentation

Nr:	Affects	Impact	Description
9.01	2.a-2.i; 3; 4	6	Show tooltips for all icons or non self-explanatory parts
9.02	2.a-2.g	7	Show brief explanation on how inputs work for every possible input feature (could be combined with Nr.01 → <a href="#">4.2 Visibility</a> )

## 10. Match between System and Real World

Nr:	Affects	Impact	Description
10.01	2.b-2.e	5	Use “Origami terminology” everywhere (e.g. Rabbit Ear instead of Triangle Incenter; Next View Here Symbol instead of Eye Symbol)

## 5 Changes and new Features

This section builds on the revealed usability issues identified during the initial evaluation process (see Section 4). On this basis, solutions can be developed that fix the most impactful usability issues. Furthermore, new features can be created that increase productivity (by automating processes and part of the workflow), while also fixing the current problems.

### 5.1 Graphical User Interface related Changes

These changes include everything that is related to the graphical user interface (GUI) with which the user interacts with.

#### 5.1.1 Overall GUI Changes

Every feature area (e.g. grid options, scaling options, arrow/symbol inputs etc.) should be separated through an `EtchedBorder` with its name as the title. Within these borders, elements should follow a consistent order. This is why the following hierarchy is being set for the placement within these areas: `JRadioButton` → `JTextField` → `JButton` → `JCheckBox`.

Though additional `JLabels` can be used to explain or to indicate the elements' specific functionalities. Whenever possible, icons should replace text and tooltips should be used to provide further explanations. Explanatory `JLabels` should only be used when icons and tooltips are not enough.

Additionally, to the more consistent placement of elements within the GUI, their sizes should be standardized as well. `JTextFields` for example should only be sufficiently large to encapsulate the largest possible entry. Consequently, the `height` should always be set to 25 pixels and the `width` depending on what is being entered (e.g. entering an angle between 0-360° will be shorter than entering the textual explanation of a folding step).

**Fixes:** #2.01; #2.02; #2.03; #4.01; #4.03

### **5.1.2 Side Panel**

The text of the `JRadioButtons` at the tool bar should be replaced by self-explanatory icons in order to improve overall clarity. But the textual explanation of all tools should still be present to provide help - especially for novice users. This can be done through tooltips that appear when hovering over the buttons.

**Fixes:** #2.01; #2.03

At a later point, a complete redesign and rework of the GUI can be carried out, to facilitate the goal of a unified and obstructionless design. At the current stage of development though the focus is being set to increase the efficiency and speed of creating origami diagrams.

## **5.2 User Input related Changes**

### **5.2.1 Arrows and Symbols**

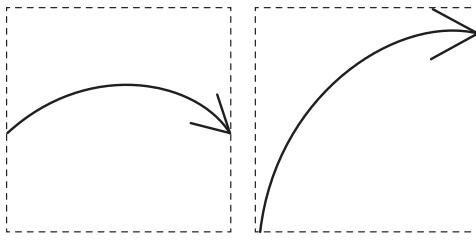
A large part of the Origrammer is build on the different arrows and symbols that can be used in diagrams. So far, these objects were simple vector graphics that were being placed on the diagram as `ImageIcons` of `JLabels`. This approach worked-out well initially, but inherently contained restrictions and problems.

As vector graphics do not have explicit width or height values, the library Batik<sup>4</sup>, which was being used to load the .svg files, presented wrong values to the `JLabel.setBounds(width, height)`-method. As a result, the arrows and symbols got partially cut off at the original bounds of the `JLabel` when rotating them. To fix this issue, a new, pre-rotated vector graphic was being loaded whenever an arrow or symbol got rotated. Though this in turn facilitated itself in a wrong, always square border around the arrows and symbols (see Fig. 5.1, where the border doesn't adjust to the arrow rotations). Additionally, this made interactions with arrows and symbols far slower and unflexible.

Another sideeffect was a change in scale when rotating a non-square object. A `JLabel` tries to display the biggest possible object that can fit within the border bounds. As displayed on Figure 5.1 the size of the arrow changes after rotating it by 45°.

---

<sup>4</sup><https://xmlgraphics.apache.org/batik/>



(a) Straight Arrow    (b) Angled Arrow

Figure 5.1: Unwanted scaling when rotating the wrong, always square hitbox, regardless of arrow shape

To encounter and solve these problems and limitations, the decision was made to completely rework how arrows and symbols function in the Origrammer. The new approach was to rebuild all objects with simple `java.awt.Shape`-objects. This gave total control over rotations, scaling, and on-the-fly-editing of all arrows/symbols. Another advantage was the removed reliance on `JLabels` and associated with that, there were no longer issues with wrong hitboxes or different scaling while rotating. The only disadvantage is the work-intensive nature of remodeling all arrows and symbols with `Shape`-objects by hand.

**Fixes:** #1.02; #7.05; #7.06

### 5.2.2 Dynamic Input Previews

The Origrammer currently has 24 different lines, arrows, and symbols that can be placed, some of which require varying input sequences. This can be confusing for novices that are not familiar with the specific inputs. Due to this potential source of frustration, a dynamic preview was implemented for all lines, arrows, and symbols. This preview shows the object to be placed, before the final placement is made.

For example, lines only require two mouse clicks to be placed, such that the preview can be shown after the first point of the line has been defined. Afterwards the line preview can show a line between the first point and the mouse cursor.

Objects like the rotation symbol that only require one mouse click for their placement, immediately show the symbol preview at the current mouse position.

**Fixes:** #1.01; #1.02; #3.01; #6.02

### 5.2.3 Filling Tool

The current limitations of the Filling Tool make it slow and cumbersome to use. The user has to manually put-in three vertices to create a triangle that shows the specified filling color. The limit of only three vertices per `OriFace`-object was originally introduced to give finer control for the user, as well as to simplify the development process for this feature.

But this decision contradicts the original goal of minimizing the time needed to create diagrams. As a result, the Filling Tool has to be reworked, so that it can be used faster and with fewer user inputs.

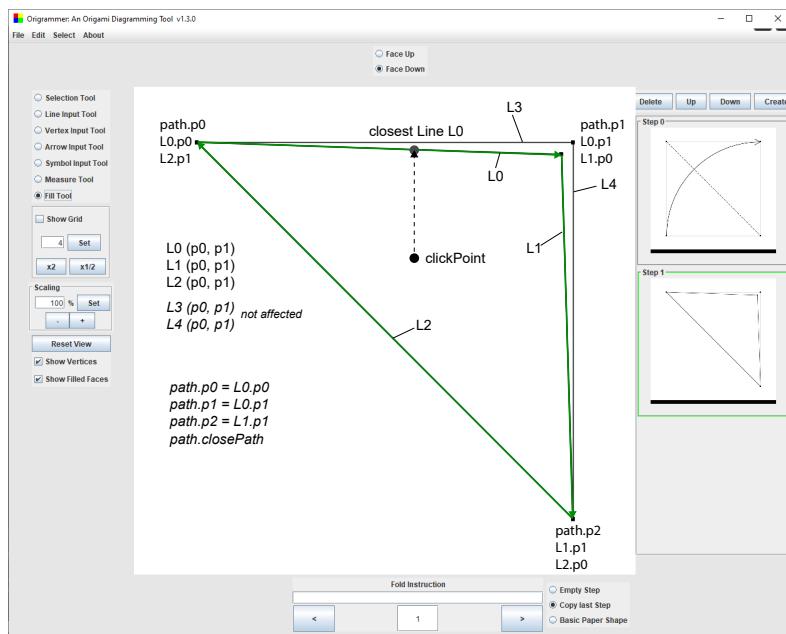


Figure 5.2: Example on how the Filling Tool works after being reworked

The new Filling Tool only requires one mouse click to fill an area with color. When clicking in an area, the Origrammer first looks for the closest line to the `clickPoint`. Afterwards, both endpoints of the line are checked for other touching or intersecting lines. The additional lines that were found have to be checked if they are visible from the original `clickPoint`, which means, that they are part of the enclosing lines. The endpoints of every new discovered line will be checked in the same manner, until a full enclosure of lines is established. Once this has happened, a `GeneralPath` is created that goes through all points of the enclosing area. Lastly, the area of the `GeneralPath` can be filled with the specified color.

## 5.3 New Origrammer Features

These new features should aim at increasing the productivity and efficiency of the Origrammer. The focus should be set on maximising the work that can be done in the shortest amount of time. This will be achieved by implementing features that either automate parts of the workflow, or give new, faster possibilities of achieving the goal.

### 5.3.1 Step Navigation and Step Editing Options

Currently, the Origrammer can create new diagram steps with three different options (*Empty Step*, *Copy last Step*, and *Basic Paper Shape*). Furthermore, the user can navigate through them with a *Next-Step-*/ and a *Previous-Step-*-button, though this approach brings limitation with it. Moving through the steps one by one hinders the overall usefulness and especially the speed of working with the Origrammer.

Deleting redundant steps can be quite important once the folding sequence goes through changes or simply when the user has made a mistake. In the same manner of offering solutions to user-induced mistakes, the Origrammer should make it possible to move steps to a different position within the diagram. Being able to create new steps in between existing ones can also be quite useful in preventing user error.

An additional problem arises, once a diagram with a lot of steps (100+) is created. As the user currently only knows where he is in the folding sequence by the step number, this can lead to confusion. For example, when folding an animal, there can be a different folding sequence, depending on what the artist decides to fold first (e.g. head first, the front or the hind legs, the tail, the abdomen, or the back). In order to continuously see where the user is within the diagram, there should be a small preview picture for all the steps within the step navigation. This gives the user an idea on what area a range of steps is working on. When clicking on one of the previews, the clicked step should made active and should be displayed on the Editing Panel.

As a result of the mentioned issues, the step navigation will be reworked to offer more functionality, visibility, and ease of use. This will be accomplished by offering the following features in the rework:

- Small preview picture of individual steps
- Click on preview picture to show the step in the Editing Panel
- Show step number, the preview picture, as well as the folding description of the step in the navigation
- Delete a selected step
- Move a selected step to the front or the back
- Create a new step between existing steps

**Fixes:** #1.03; #3.02; #7.07

### 5.3.2 Folding Presets

In origami diagramming, there are eight widely established bases (Lang, 2012, p.53-64 [9]). When starting with these bases, one can fold a variety of different models. The bases in question are:

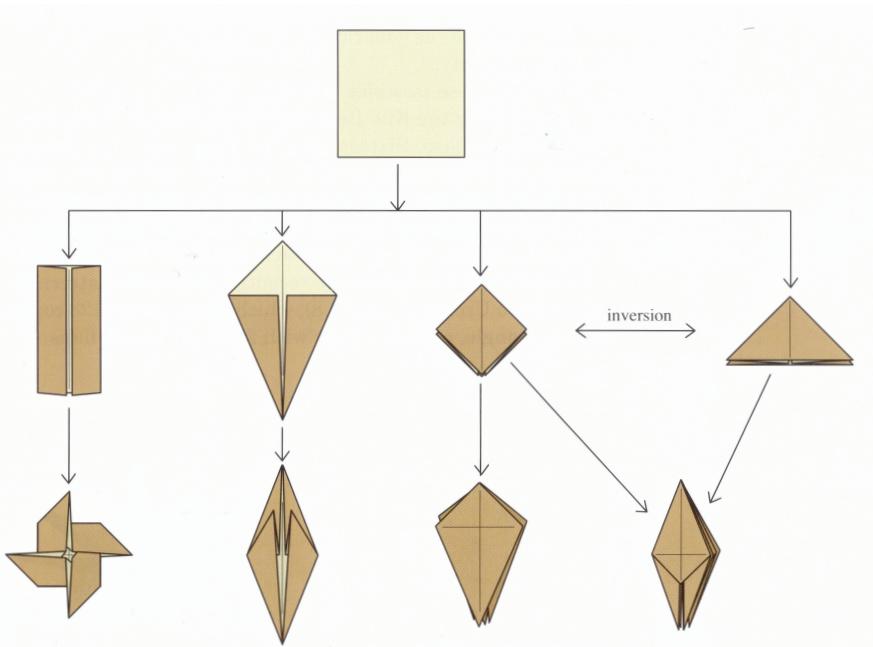


Figure 5.3: Family Tree of the standard bases.

Top to bottom and left to right: Cupboard Base, Windmill Base, Kite Base, Fish Base, Preliminary Fold, Bird Base, Waterbomb Base, Frog Base - (Robert J. Lang, 2011, p.61 [10])

As these bases serve as the starting point for a wide array of models, it would be helpful to include them in the Origrammer. By giving the user the option to start a diagram with one of these bases, the overall diagramming process will be shortened quite a bit. Not having to keep repeating the same beginning steps for different models will save time and can also show novices of the program, how parts of the Origrammer look like or work.

Additionally to the origami bases, instructions on how to fold different grid sizes, should be included in the folding presets as well. Folding a 3x3, 5x5, or 7x7 grid from a square paper is not as straight forward and requires multiple folding steps to achieve.

#### Fixes: #7.09

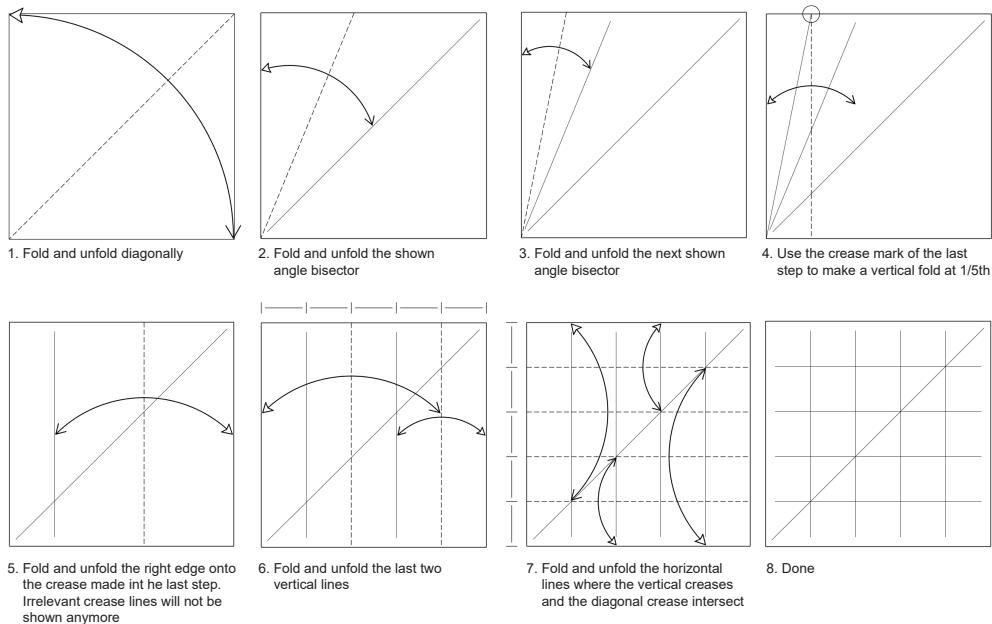


Figure 5.4: Example diagram of a 5x5 grid created with preset feature of the Origrammer

### 5.3.3 Additional Line Input Types

So far, lines in the Origrammer can only be inputted at existing vertices or through the helping grid. In order to give the user more flexibility and control more line input variants have been implemented.

#### Angle Bisector

In case an angle has to be folded in half, the user can now use the *Angle Bisector* line input. The two lines that span the angle that needs to be folded in half have to be selected and the Origrammer automatically places this new crease line.

#### Perpendicular

Oftentimes during diagram creation, crease lines will be placed at irregular angles. Enabling the creation of perpendicular ( $90^\circ$ ) lines at these existing, irregular creases is a significant improvement. This perpendicular line is extended until it touches an existing line. Its direction can be reversed when holding the *CTRL*-key during line input.

#### Extend Line

Occasionally, important reference points are missing and make some lines impossible to input. The *Extend Line* feature works by inserting a normal line through the existing reference points and the grid but then the line gets extended to both sides until it intersects other existing lines.

#### Mirror Lines

Origami models frequently have mirrored sides and thus a *Mirror Lines* feature was introduced. First, the user selects the lines that should be mirrored. Afterwards a mirror line needs to be specified, constituting the line from which other selected lines will be mirrored.

#### By Length and Angle

Placing lines with a specific angle (towards the x-axis) and length gives the user even more fine control over the line placement. This input method can be combined with the *Measure Angle/Length*-tool to copy existing lines.

**Fixes: #3.04**

## 6 Virtual Folding of the Paper

When creating diagrams with the Origrammer, the most time-consuming activity is the placement of all required lines. All edges, current creases, and existing creases have to be placed precisely to allow the user to closely follow the instructions simply by the visual cues. This large bottleneck in the diagramming process can only be eliminated by simulating the actual folding process virtually.

Being able to drag and drop a corner or edge of the paper to the desired position would significantly reduce the work required.

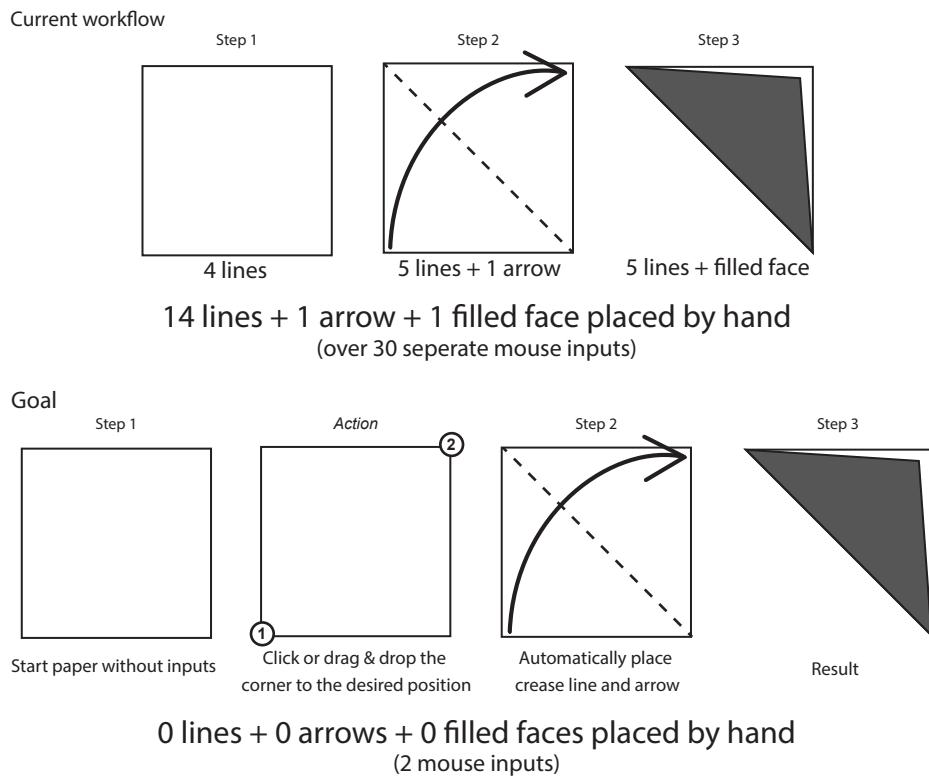


Figure 6.1: Current workflow (top) & Desired workflow (bottom)

To implement this automated folding, several parts of the Origrammer will have to be reworked. As there have already been a few attempts to simulate paper folding, the following sections will explore these projects to gain additional knowledge relevant for the Origrammer rework.

## 6.1 Related Works

Folding virtually to simulate folding origami models constitutes a goal pursued by several projects. This section will list and explore several of those attempts to gain information on how the folding had been implemented in these projects. Building on this extant knowledge, the implementation of the Origrammer can be developed further while additionally improving on previous attempts at the same time.

### 6.1.1 Origami Simulation

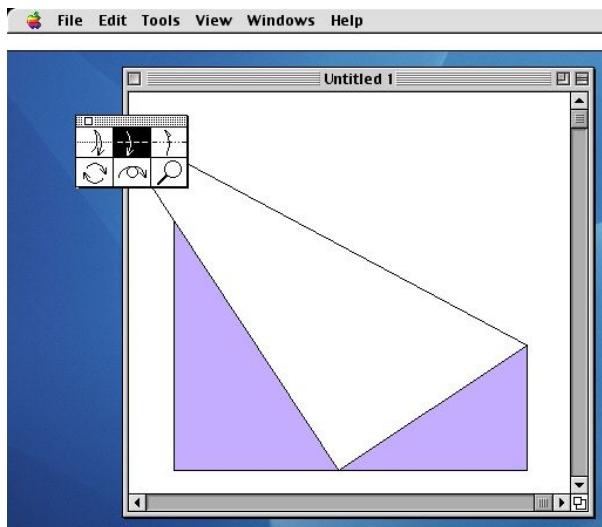


Figure 6.2: Origami Simulation by Robert J. Lang

One of the first documented approaches to simulate origami folding is *Origami Simulation* [11] by Robert J. Lang. This desktop application was “written in Object Pascal for the Mac OS, using Lightspeed’s THINK Pascal and the THINK Class Library” (Lang, 2015, para. 2 [11]). It is able to carry out simple Valley and Mountain Folds, as well as to rotate and turn the paper around. Though, as this application was originally developed in the 1980s, it is now only runnable on Mac Classic mode, such that it does no longer run on Mac OS X 10.5 and later or on Intel-based Macs. However, the source code is still publicly available<sup>5</sup> and could prove useful for conceptional purposes when implementing the virtual folding of the Origrammer.

<sup>5</sup><http://www.langorigami.com/wp-content/uploads/2015/09/origamisim.src.zip>

### 6.1.2 Foldinator

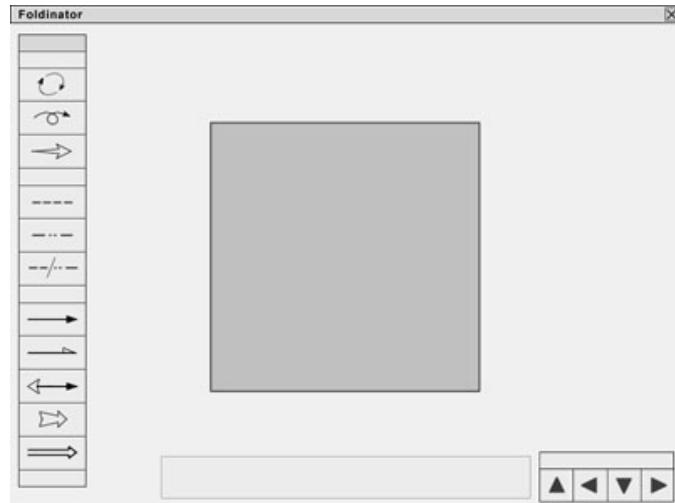


Figure 6.3: Foldinator by John Szinger

The Foldinator [12] is a software application developed by John Szinger aiming at virtualizing the folding process and subsequently creating origami diagrams. The program was originally developed in Flash using ActionScript1 but was later ported to the Apache Flex SDK<sup>6</sup>. The paper in Szingers approach was represented by polygons that are split up and moved when folding. Further, there is no publicly available explanation on what constraints and processes are used for the folding itself.

Featurewise, the Foldinator is able to carry out Valley and Mountain folds, rotate, and turn the paper around.

---

<sup>6</sup><http://flex.apache.org/index.html>

### 6.1.3 eGami

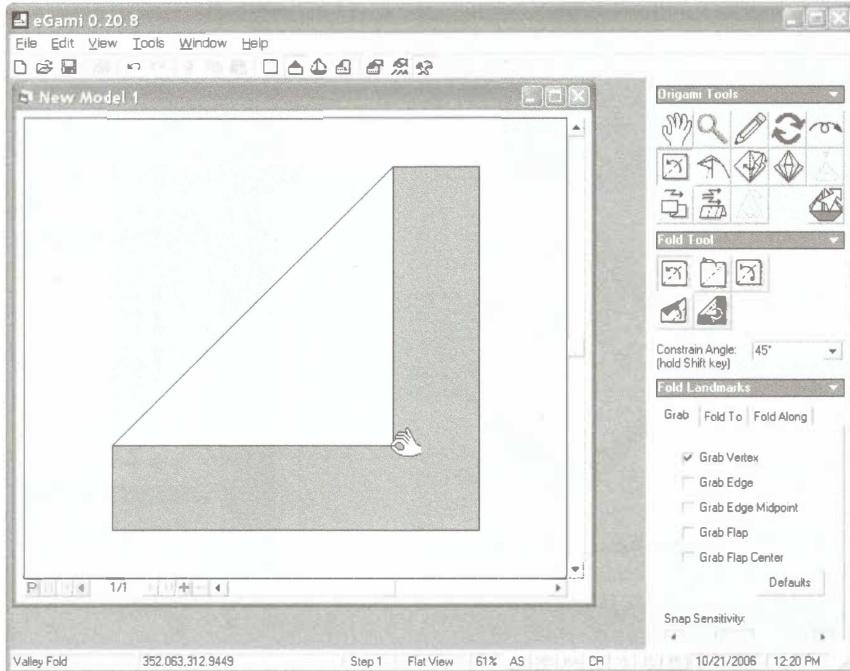


Figure 6.4: eGami by Jack Fastag

The desktop application eGami [13] by Jack Fastag can simulate the folding of flat origami models and create automatically generated diagrams. While publicly available information on which programming language the application was developed on could not be identified, the paper published in *Origami<sup>4</sup>: Fourth International Meeting of Origami Science, Mathematics, and Education* contains more detailed explanations of the inner workings.

eGami is using a winged-edge boundary representation (Foley, 1996, p.545-546 [14]) to store and work with a polygon mesh, which is representing the paper. Through this representation, a series of constraints ensures correct properties of the paper. Additional constraints also guarantee that only valid folds can be carried out:

- (a) *All polygons are closed*
- (b) *All edges are used by not more than two polygons*
- (c) *Each vertex is referenced by at least two edges and at least one polygon*
- (d) *The mesh is completely connected, topologically planar and has no holes*
- (e) *Vertices on the raw edge of the paper must be connected to exactly two “raw” edges*
- (f) *Vertices inside the paper must satisfy Maekawa’s Theorem*
- (g) *Paper does not stretch, which means vertex positions are fixed and edge lengths are constant*
- (h) *Paper faces are flat, which means that each polygon’s vertices are coplanar and implies that only rigid folds are allowed*
- (i) *No rips or holes*
- (j) *Paper is infinitely thin, implying that paper “creep” is not taken into account*
- (k) *Only flat folds are allowed, meaning each vertex inside the paper must satisfy Kawasaki’s Theorem*
- (l) *Paper does not intersect itself*  
(Fastag, 2006, 3.1 [13])

Implementation of Maekawa’s and Kawasaki’s theorems of flat foldability[15] in addition to the other mentioned constraints could improve the reliability of Origrammer’s virtual folding and should be pursued further.

#### 6.1.4 Conclusion

Previous attempts at virtually folding paper for diagram creation have proven its viability. Unfortunately, they have either not been updated in years or never proceeded further than simple Valley and Mountain folds. Additionally, for most projects there is no detailed information available on how these programs work and how they actually implement the folding simulations. So for the Origrammer, all underlying processes should be explained and documented. This will give a starting point for other future approaches to the virtual folding. Furthermore, a set of constraints (similar to eGami 6.1.3) should be implemented ensuring that only valid folds can be made.

## 6.2 Requirements

The following aspects build the concrete requirements of what the virtual folding should do automatically and what has to be kept track of in order to implement this system of simulated folding.

The paper lines have to be stored in a way such that their relations and connections to each other are stored and preserved. Then, as seen in Figure 6.1, the folding arrows have to be placed automatically when folding parts of the paper. The folding line where the new crease is created should be placed automatically, as well. This crease line might also have to go through multiple layers of paper, which will have to be checked for each folding action. Lastly, the folded part of the paper has to actually change position and direction, and possibly show the differently colored side of the paper.

## 6.3 Current Status

So far, every line got stored within a single `ArrayList<OriLine>` and was otherwise completely independent of the other lines. An additional `ArrayList<OriVertex>` is tracked to efficiently iterate through all vertices that can be used as an input point for new lines, arrows, or symbols. All arrows and symbols got also stored within their respective `ArrayList<>` per folding step. Lastly, the folding steps themselves got stored within an `ArrayList<Step>` as well, thereby, making the saving of the full diagram in a .xml-file simple and efficient.

This approach worked reliably while there were no dependencies between the lines and vertices. But when folding virtually, several dependencies and other values have to be tracked. These include the edge lines, existing creases, newly made creases (all with their specific line types), correct layering when rendering folded paper, correct colouring of different paper sides, and the rendering with slight distortions to show hidden layers/flaps (Lang, 2011, para. *Distort the model to show all the layers [8]*), among others.

## 6.4 Reworked System

The new implementation works by splitting the actual paper model into `OriPolygons` that can be folded, split up, and rendered (see Listing 1).

Listing 1: `OriPolygon`

```
OriPolygon {
    OriVertexList vertexList;
    ArrayList<OriLine> lines;
    int height; /*height of the polygon (rendering order)*/
}
```

Once the user inserts a new fold line, the Origrammer checks which `OriPolygons` are affected by the fold and subsequently splits them at the points where the folding line crosses the edges of a given polygon. The resulting line at the newly created crease is now a shared line between the two smaller, split up polygons. This is more closely visualized at Figure 6.5.

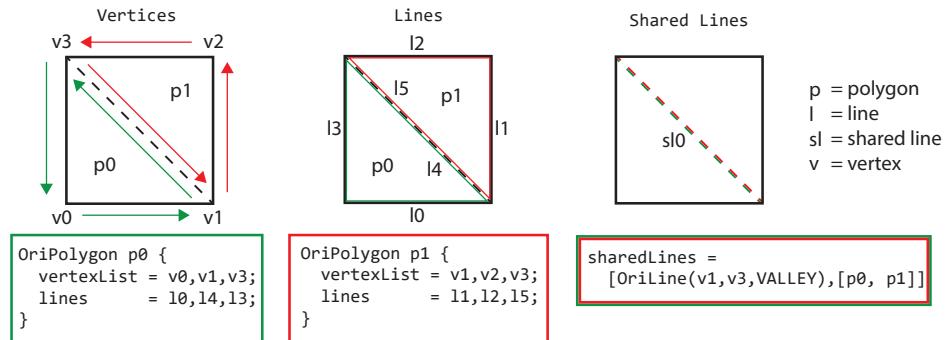


Figure 6.5: Reworked Paper Representation through `OriPolygons` and `sharedLines`

Every `OriPolygon` in turn consists of a `OriVertexList`, which contains all vertices of aforementioned polygon in a counter-clockwise order. Additionally, an `ArrayList<OriLine>` is required to keep track of the edge lines of the paper. As an `OriLine` also includes the line type, the lines within the `OriPolygon.lines`-list can have the type `EDGE` or `SHARED`.

For the case that a polygon is sharing a line with another polygon, the `HashMap<OriLine, ArrayList<OriPolygon>> sharedLines` was added.

Listing 2: New Implementation of Paper Model

```
ArrayList<OriPolygon> polygons
HashMap<OriLine, <ArrayList<OriPolygon>> sharedLines
```

This *shared lines*-list contains all the shared lines in combination of which polygons are sharing the given line. Additionally, the real line type is stored here, which is later used for rendering. If the **OriPolygons** stored the actual line type instead of the mockup type **SHARED**, then there is no programmable connection between several **OriPolygons**. This would make folding lines that go through multiple polygons at the same time harder to calculate and progress.

Alternatively, there could be a flag for each **OriLine** that would signalize if it is also a shared line. But then one would have to iterate through all **OriPolygons** and check each individual line in them when looking for the connected polygon of a given shared line. So even though the **HashMap** of shared lines has to be updated and kept track of at all times, it is still more flexible and efficient compared to the system of flagging an **OriLine** as a shared line.

Listing 3: OriVertexList, OriVertex, and OriLine

```
OriVertexList {
    int n; /*vertex count*/
    OriVertex head;
}

OriVertex {
    Vector2d p; /*Vector2d(x, y)*/
    OriVertex prev;
    OriVertex next;
}

OriLine {
    OriVertex p0;
    OriVertex p1;
    int type;
    /*NONE, EDGE, MOUNTAIN, VALLEY, XRAY, CREASE,
     (SHARED), (DIAGONAL)*/
}
```

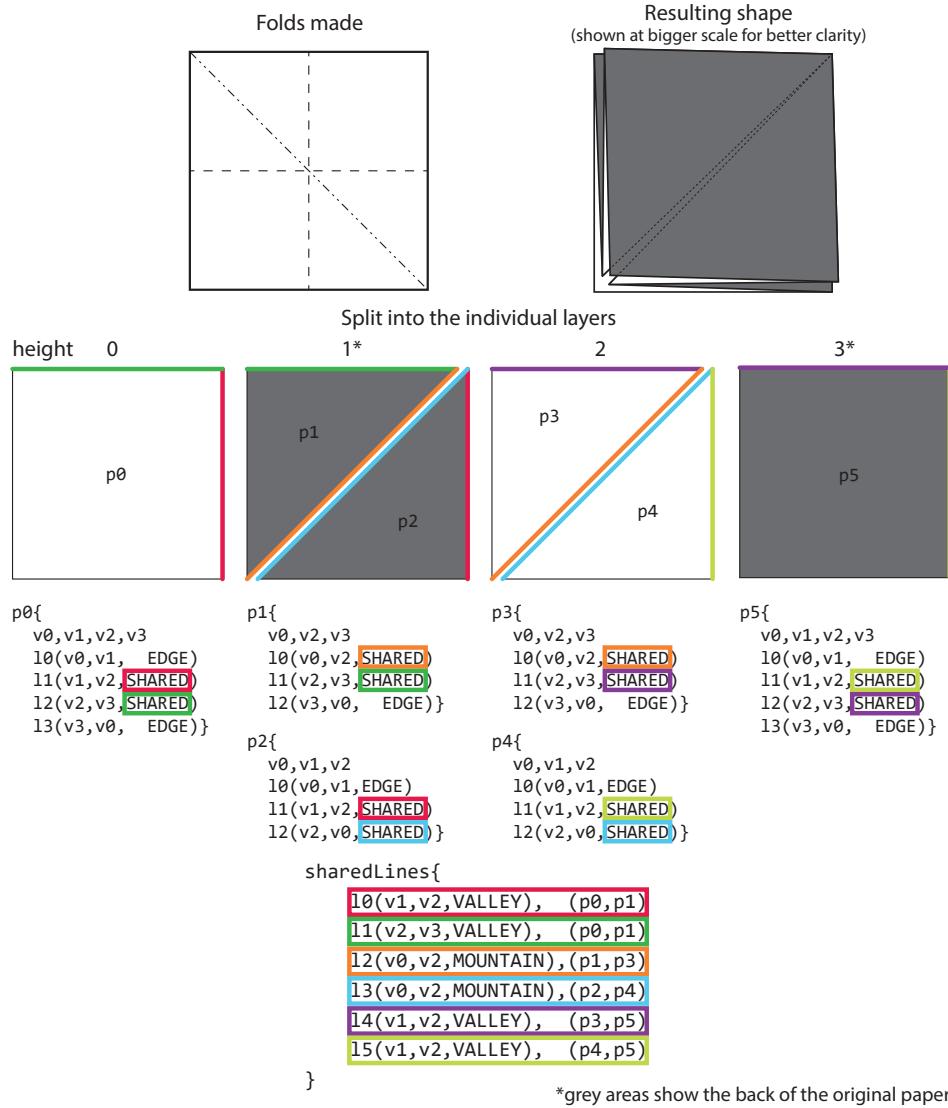


Figure 6.6: A more complex example of splitting polygons and multiple layers

The example in Figure 6.6 shows a slightly more complex example how the Origrammer splits the polygons and creates the correct shared lines. Additionally, in this case multiple layers of the paper have to be rendered and kept track of. This height value is only preserved for each `OriPolygon` and is not used in the vertices, lines, or shared lines. It appears worth highlighting that for each shared line there can only be two `OriPolygons` that share one edge. So when making a fold on one polygon and this fold crosses a shared line of this polygon, the Origrammer only has to search for

the counterpart polygon, that shares the same line. This process can be seen in Figure 6.7, where a fold through multiple layers is carried out. Internally, the Origrammer starts with a simple folding line on the top polygon that goes to its edges. If one or both endpoints of the folding line end in a shared line, it is clear that other polygons are also affected by this fold. Then one has to only iterate through the list of shared lines and find the connected polygon/s. Once the new `OriPolygon`/s have been found, the same process can be repeated subsequently.

1. Create the folding line on the top layer
2. Check both endpoints if they are ending in a shared line
3. Find the connected polygons on the sides that have a shared line
4. Repeat step 1-3 with the next highest layer until on layer 0 or no other polygons are found

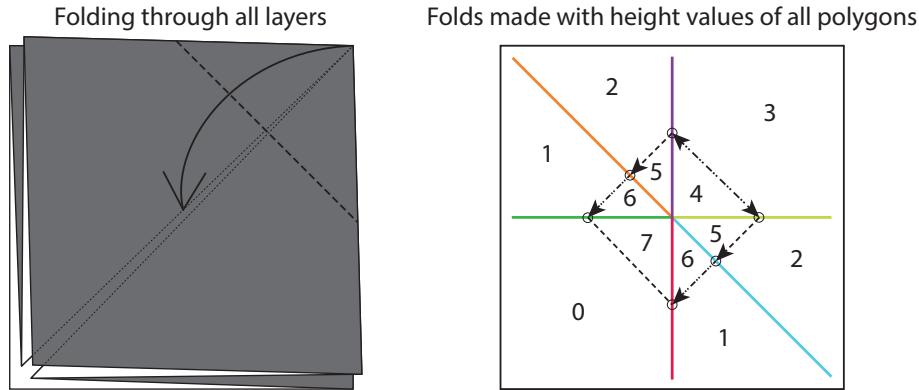


Figure 6.7: The process of folding through multiple layers and the updated height values

Figure 6.7 additionally highlights that the colored lines (shared lines) will have to be split up when applying the mountain and valley folds. The marked crease lines also represent newly made shared lines, as they split existing polygons.

### 6.4.1 Current Limitations

The current implementation of the virtual folding still inherently contains limitations. While the user can freely fold the paper, even through multiple layers and in different directions, several more complex folding procedures can - yet - not be carried out.

#### Reverse Folds

Reverse Folds allow the paper to change directions. Inside Reverse Folds are made when pushing a corner of two layers in between these layers. At the same time, the original folding line between the two layers is being reversed, which explains the origin of its name.

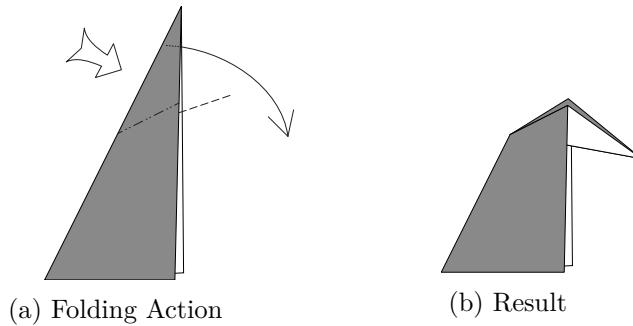


Figure 6.8: Inside Reverse Fold

While for Inside Reverse Folds the paper is sunken into the layers of the paper, for Outside Reverse Folds, the fold wraps over the paper.

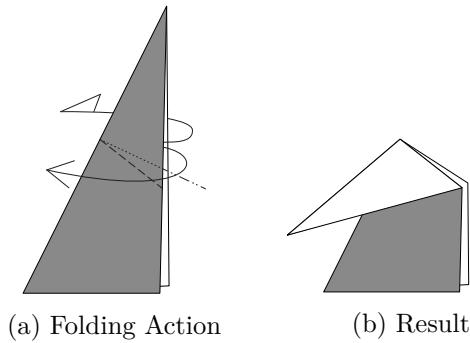


Figure 6.9: Outside Reverse Fold

The Reverse Folds require a mountain as well as a valley fold at the same time, while also changing the height values of the affected layers. As the

Origrammer can currently only carry out one fold at a time, it can not make Reverse Folds yet.

### Pleating and Crimping

Pleats can be created by the Origrammer, as the folding process only requires two different, independent folds (see Fig. 6.10a). As these folds don't intersect or affect each other, they can be carried out one by one. Though this folding procedure could still be made more straight forward by offering a specific function for pleats. This way the user would only have to create one instead of two separate folding actions.

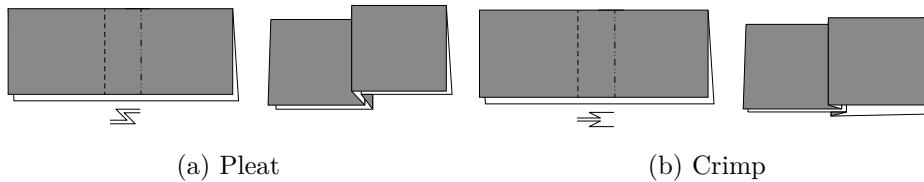


Figure 6.10: Pleats and Crimps

Crimps on the other hand require one additional step. They can be started by a simple pleat followed by immediately folding it in half, thereby, creating a crimp. It needs to be highlighted that one part of the crimp partially encapsulates the other side. This can be seen in Fig. 6.10b where the right part of the paper surrounds the left. Even though a crimp could technically already be carried out by the Origrammer in three separate steps, it is desirable to create a dedicated crimp-feature. This is especially important when the paper is already folded in half and a crimp has to be folded from there. Folding a crimp in this vein is currently not possible as the Origrammer would have to carry out two mountain and two valley folds at the same time.

### Sinks

The most complex folding procedures to automatically carry out by the Origrammer are the so-called sinks. The examples in Fig. 6.11 show the three different types, the open, the mixed, and the closed sink. A basic open sink is performed by pushing a corner of paper (with multiple layers) downwards and reversing the direction of all folds within the pushed-in corner at the

same time. This reversing of existing creases is visualized in the so-called *Crease Pattern* from Fig. 6.11. A crease pattern displays an unfolded view of the paper, but still shows all existing mountain and valley folds. Through this approach, even the hidden creases can be shown to give a full overview. But as the crease patterns show, multiple existing crease lines have to be reversed and almost all height values of the layers change, as well. This makes the calculations for the automated folding more complex - especially when sinking multiple layers at the same time.

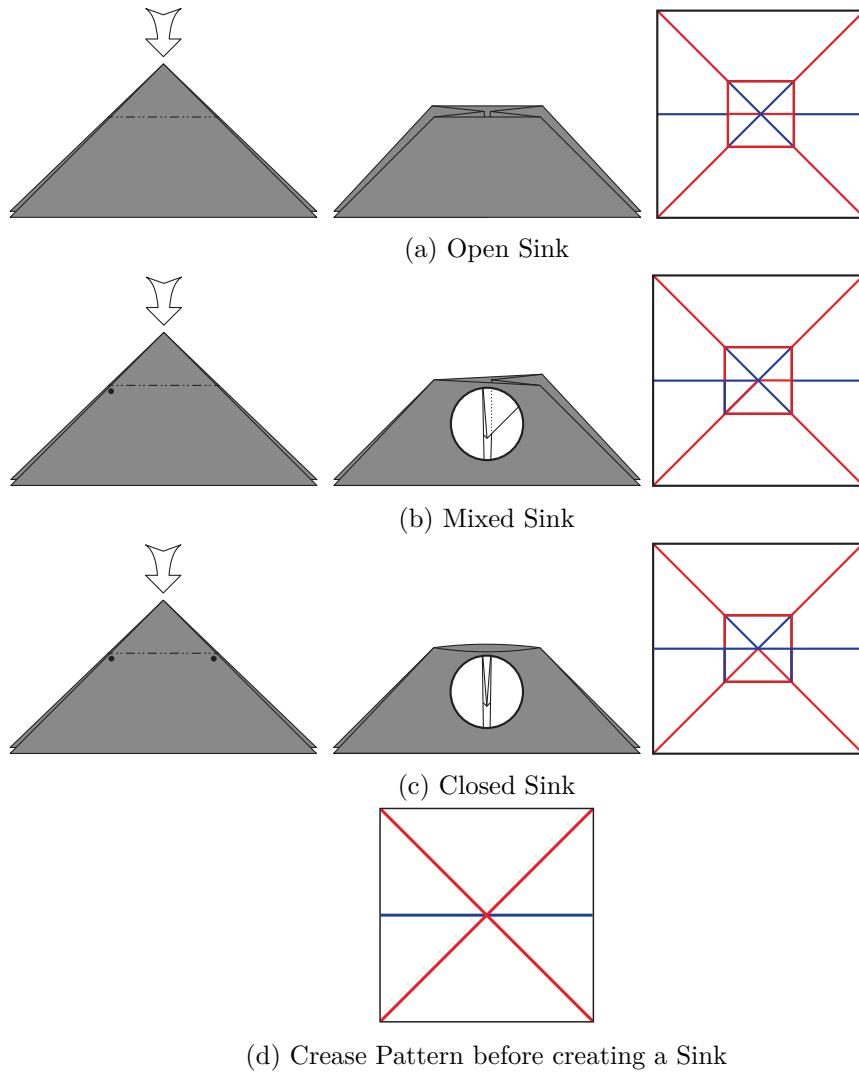


Figure 6.11: Different forms of Sinks<sup>7</sup>

<sup>7</sup>red lines = mountain folds; blue lines = valley folds

## 7 Problems and other Findings

This section discusses decisions and problems during the development process.

### 7.1 Virtual Folding

Simulating the folding of paper for origami models by the Origrammer has introduced a multitude of edgecases and constraints that have to be tracked, to present a correct and predictable result.

#### 7.1.1 Folding over and over

Making simple valley or mountain folds with only one layer of the paper being folded (e.g. folding the corner of the paper), does not encounter many constraints. Internally, the Origrammer just has to split the existing polygon on the folding line and then update the position (fold) of one of these split-up polygons with the folding line as the base.

Though, as soon as there are multiple polygons connected to the newly made crease line, more calculations will have to be carried out. Especially, once the user wants to fold over and over to create multiple layers of paper (as seen in Figure 7.1), all the affected polygons are influencing the resulting layer height of each other.

Listing 4: OriPolygon

```
ArrayList<OriPolygon> sortedPolygons;
int curLowHeight = hmin; //1
int curUpHeight = hmax; //7
boolean justDidUpper = false;

for (OriPolygon curP : sortedPolygons) {
    if (!justDidUpper) {
        curP.setHeight(curUpHeight);
        curUpHeight -= 1; //goes lower for every 2nd folded polygon
        justDidUpper = true;
    } else {
        curP.setHeight(curLowHeight);
        curLowHeight += 1; //goes higher for every 2nd folded ←
                           polygon
        justDidUpper = false;
    }
}
```

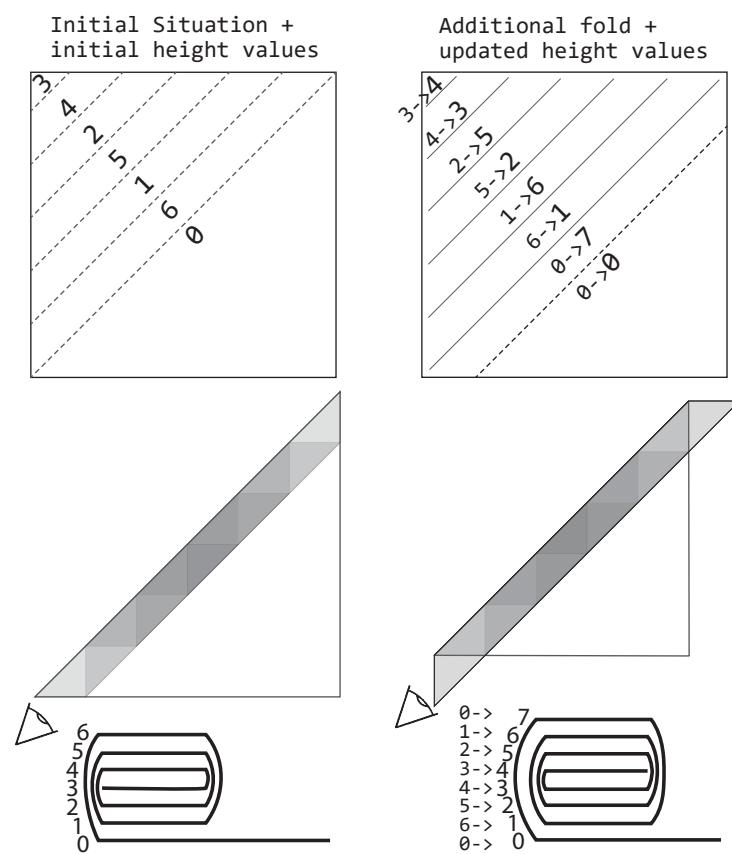


Figure 7.1: Folding the paper over and over - Folding Polygons that don't share a line with the folding line

## 7.2 Folding multiple layers

Another problematic scenario arises once the user wants to fold through multiple independent layers. In this case, independent from each other means that the polygons representing these layers can be folded independently from each other. This scenario is illustrated in Figure 7.2 where the two corners facing the top right are to be folded downwards.

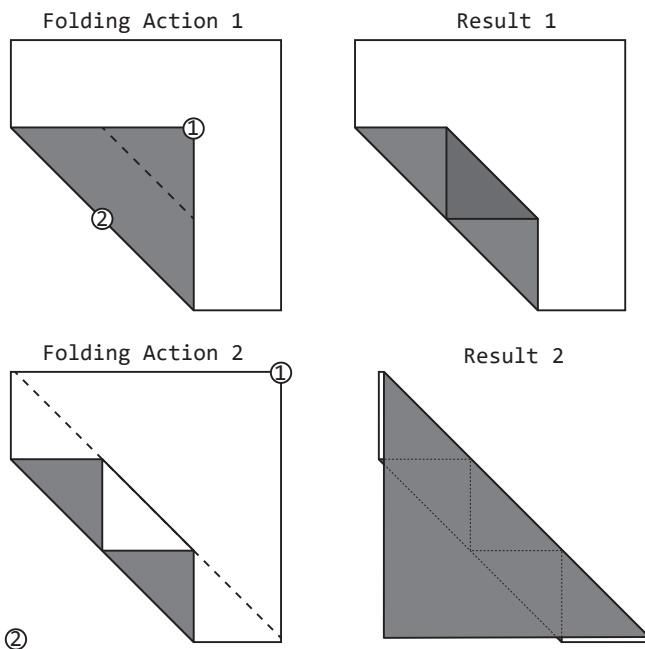


Figure 7.2: Folding through multiple independent layers at the same time

Currently, the user would have to carry out these folding actions one by one and the Origrammer would create 2 separate folding steps. This situation arises because the Origrammer only folds all connected layers looking from the top layer. But as the inputted folding line crosses edge lines on both sides of this top polygon, it is the only layer being folded.

Additionally, in this specific example there is no clear reference point where the second folding action should end (point 2 at Folding Action 2). This is compared to the first folding action, where the corner simply gets folded towards an outer edge line.

In order to increase efficiency and to remove this edgecase of missing reference points, a feature should be implemented, providing the user with the opportunity to specify how many layers the folding step should fold through.

## 8 Outlook

The automation of processes in the Origrammer has changed the workflow of creating diagrams as a whole. Instead of placing lines, arrows, and symbols one by one, the user can simply fold the paper with two mouse clicks, thereby folding the paper automatically. Furthermore, the correct folding arrow is placed without any additional user input and the following step is started automatically.

With the use of the step navigation, the user can navigate more efficiently and flexible. Steps can be moved, duplicated, deleted and created freely. Moreover, the preview images of said steps shows the user where he is in the diagram.

Another important improvement is made for the input of lines, arrows, and symbols. During the input process dynamic previews are being shown to guide the user which input sequences have to be strictly followed . This will help especially novices that don't necessarily know the specific input structure of all objects.

Future development could improve on the virtual folding, especially for more complex folding actions like reverse folds, crimps and pleats, and opened/mixed/closed sinks. Another issue of the current folding system is the inability to display curves and 3d-shaped parts of the origami model. This would be especially important during the shaping process at the end of folding a model.

Additionally, the user interface still contains several usability issues, even though most of them were already discovered during the heuristic evaluation. Consequently, the user interface will have to be reworked and improved in the future. At this later point the involvement of real users appears valuable to maximise usability improvements.

## **9 Declaration of Authorship**

I hereby declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources. According to my knowledge, the content or parts of this thesis have not been presented to any other examination authority and have not been published.

21.09.20 Gummersbach J. Körny

Date, City, Signature

## Glossary

**Origrammer** Origrammer is the name of the diagramming program developed within the scope of this project. [4](#), [6](#)

**Diagram** Origami diagrams give folding instructions for origami models. They usually consist of a representation of the model per folding step, the folding instruction in symbol form, as well as the folding instruction in text form. [4](#)

**Mountain Fold** The fold that results after folding one part of the paper behind the other. After unfolding you roughly get an A-shape. It is usually visualized by a dot-dot-dash line. [4](#)

**Valley Fold** The fold that results after folding one part of the paper over the other. After unfolding you roughly get a V-shape. It is usually visualized by a dashed line. [4](#)

**Origami** (jpn: *ori* = *folding* and *kami* = *paper*) is the art of folding paper into models of animals, people, or other objects. [5](#)

**Crease Pattern** A Crease Pattern is “a set of lines that is a representation of some subset of folds in an origami shape, real or imagined” (Lang, 2015, para. 1[[16](#)]). [38](#)

## List of Figures

2.1	Crane Diagram . . . . .	5
3.1	Menu Bar (Purple), Top Panel (Yellow), Side Panel (Red), Editing Panel (Green), Navigation Panel (Blue) . . . . .	7
3.2	Options when creating a new diagram . . . . .	8
3.3	Origrammer Preferences . . . . .	9
5.1	Unwanted scaling when rotating the wrong, always square hitbox, regardless of arrow shape . . . . .	20
5.2	Example on how the Filling Tool works after being reworked . . . . .	21
5.3	Family Tree of the standard bases. Top to bottom and left to right: Cupboard Base, Windmill Base, Kite Base, Fish Base, Preliminary Fold, Bird Base, Waterbomb Base, Frog Base . . .	23
5.4	Example diagram of a 5x5 grid created with preset feature of the Origrammer . . . . .	24
6.1	Current workflow (top) & Desired workflow (bottom) . . . . .	26
6.2	Origami Simulation by Robert J. Lang . . . . .	27
6.3	Foldinator by John Szinger . . . . .	28

---

6.4	eGami by Jack Fastag . . . . .	29
6.5	Reworked Paper Representation through <code>OriPolygons</code> and <code>sharedLines</code> . . . . .	32
6.6	A more complex example of splitting polygons and multiple layers . . . . .	34
6.7	The process of folding through multiple layers and the up- dated height values . . . . .	35
6.8	Inside Reverse Fold . . . . .	36
6.9	Outside Reverse Fold . . . . .	36
6.10	Pleats and Crimps . . . . .	37
6.11	Different forms of Sinks . . . . .	38
7.1	Folding the paper over and over - Folding Polygons that don't share a line with the folding line . . . . .	40
7.2	Folding through multiple independend layers at the same time	41

## References

- [1] Jakob Nielsen. 10 usability heuristics for user interface design, April 1994. <https://www.nngroup.com/articles/ten-usability-heuristics/> [Online; last accessed July 25, 2020].
- [2] Akira Yoshizawa. *Atarashi origami geijutsu*. Origami Geijutsu-Sha, Tokyo, 1954. no ISBN.
- [3] Joseph Wu. Origami: A brief history of the ancient art of paperfolding, 2006. <http://www.origami.as/Info/history.php> [Online; last accessed July 20, 2020].
- [4] Koshiro Hatori. K's origami: History of origami. <https://origami.ousaan.com/library/historye.html> [Online; last accessed July 20, 2020].
- [5] Anrew Hudson. Crane, 2011. <https://ahudsonorigami.files.wordpress.com/2011/11/tsuru.pdf> [Online; last accessed July 16, 2020].
- [6] Julian Hardtung. Origami diagramming: Development of a desktop application for creating origami diagrams, April 2020.
- [7] Jakob Nielsen and Rolf Molich. Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '90, page 249–256. Association for Computing Machinery, 1990. <https://static1.squarespace.com/static/57d1d882e58c62c988fb0254/t/5ae8fa3b6d2a73ce72214d91/1525217854980/nielsenheuristicsCHI.pdf> [Online; last accessed September 3, 2020].

- [8] Robert J. Lang. Origami diagramming conventions, 2011. <http://www.langorigami.com/article/origami-diagramming-conventions> [Online; last accessed August 21, 2020].
- [9] Robert James Lang. *Origami Design Secrets: mathematical methods for an ancient art*. CRC Press, second edition edition, 2012. ISBN: 978-1-56881-436-0.
- [10] Robert J. Lang. Family tree of the standard bases. Published in: *Origami Design Secrets: mathematical methods for an ancient art*; p. 61.
- [11] Robert J. Lang. Origami simulation, 2015. <http://www.langorigami.com/article/origami-simulation> [Online; last accessed August 16, 2020].
- [12] John Szinger. The foldinator project, 2009. <http://zingman.com/origami/foldinator.php> [Online; last accessed August 10, 2020].
- [13] Jack Fastag. egami: Virtual paperfolding and diagramming software. In Robert J. Lang, editor, *Origami 4*, pages 273–284. A K Peters, Ltd., 2006.
- [14] James D Foley, Foley Dan Van, Andries Van Dam, Steven K Feiner, John F Hughes, Edward Angel, and J Hughes. *Computer graphics: principles and practice*, volume 12110. Addison-Wesley Professional, 1996.
- [15] Thomas Hull. On the mathematics of flat origamis. *Congressus numerantium*, pages 215–224, 1994.
- [16] Robert J. Lang. Crease patterns as art, 2015. <https://langorigami.com/article/crease-patterns-as-art/> [Online; last accessed September 15, 2020].