



UNIVERSITAT DE
BARCELONA

Facultat de Matemàtiques
i Informàtica

DOBLE GRAU EN MATEMÀTIQUES
I ENGINYERIA INFORMÀTICA
Treball final de grau

Origami Simulator
Folding beyond paper

Autor: F. Xavier Colet i Guitert

Directora: Anna Puig

**Realitzat a: Departament de Matemàtiques
i Informàtica**

Barcelona, January 19, 2020

Abstract

This project aims to create a multi-platform application that allows the user to emulate the folding of a paper into an origami figure in a virtual environment. This application would start to solve the very time-consuming task in the origami world of diagramming, creating instructions for a specific model. In this proposal, a list of possible simple folds has been selected and each has been mathematically described with its constraints and requirements. A data structure that adapts to the complex origami folding process has been designed and a Unity application has been coded to allow the user to visualize and intuitively interact with it in a simple and clean graphic interface.

Resum

L'objectiu d'aquest treball és crear una aplicació multi-plataforma que permeti a l'usuari simular el plegat d'una figura de paper en un entorn virtual. Això començaria resoldre un problema de la comunitat de l'origami on es passa molt de temps generant el material gràfic per als diagrames, instruccions per a plegar un model concret. Amb aquest objectiu s'ha confeccionat una llista de plecs simples i s'ha descrit cadascun matemàticament amb els seus requisits i limitacions. S'ha dissenyat també una estructura de dades que s'adapta al complex procés de plegat i s'ha creat una aplicació en Unity que permet visualitzar el model i interactuar-hi de manera intuïtiva en una interfície gràfica clara i senzilla.

Resumen

El objetivo de este trabajo es crear una aplicación multi-plataforma que permita al usuario simular el plegado de una figura de papel en un entorno virtual. Esta aplicación empezaría a resolver un problema de la comunidad del origami, las incontables horas que se pasa dibujando diagramas, instrucciones para plegar un modelo concreto. Con éste objetivo se ha creado una lista de pliegues simples y se han descrito matematicamente con sus requisitos y limitaciones. También se ha diseñado una estructura de datos que se adapta al complejo proceso de plegado y se ha creado una aplicación en Unity que permite visualizar el model y interactuar con él de manera intuitiva en una interfície grafica clara y sencilla.

I would like to thank everyone that has helped to carry out this project: Marc Vigo, Joan Carles Naranjo and especially my tutor Anna Puig for her constant help in a very incident-full semester.

I would also like to thank my family's help and support during some frustrating moments. Irene for making me work in advance and until the late hours of the night. And finally to La Granja (Martí, Aleix, and Marina) for countless hours of company and doubt-solving, revising, sweet-eating and relaxing.

Finally, to the origami community for giving me the chance to learn and inherit such passion to face this project.

Contents

1	Introduction	1
1.1	Scope of the project	1
1.2	Motivation	1
1.3	General objectives	2
1.4	Specific objectives	2
1.5	Planing	3
1.6	Organization of the report	3
2	Analysis	5
2.1	Nomenclature	5
2.1.1	Origami	5
2.1.2	Other important concepts	7
2.2	Functional requirements	7
2.3	Technological requirements	9
2.4	Developing software justification	9
3	Antecedents	10
3.1	State of the art	10
3.1.1	Data structure	10
3.1.2	Mathematical foundations	10
3.2	Similar applications	11
3.2.1	Rabbit Ear [7]	11
3.2.2	Tree Maker [8]	12
3.2.3	Tess [4]	13
3.2.4	How to make Origami [1]	14
3.2.5	Non-software alternatives	14
3.3	Conclusion	17
4	Formalization	18
4.1	Finding the symmetry plane	18
4.2	The fold topological map	19
4.3	The fold geometric map	20
5	Design	21
5.1	Proposal	21
5.1.1	Model	21

5.1.2	Controller	25
5.1.3	View	27
6	Results and simulations	30
6.1	Fish	31
6.2	Flapping Bird	33
6.3	Features	37
6.3.1	File saving and loading	37
6.3.2	Play button	37
6.3.3	Deployment	38
7	Conclusions and future work	39
A	Technical manual	40
A.1	Installation	40
A.2	How to develop	40
A	User manual	41
A.1	Interactions	41
A.2	Fold requirement:	41

1 Introduction

1.1 Scope of the project

Origami is the Japanese art of folding paper. It consists of folding a piece of paper, usually a square, several times in different ways without cutting it or using glue to obtain a figure oftentimes representing an animal, an object or human figure. They range from the very simple (Figure 1) to a very high level of complexity (Figure 2).



Figure 1: Traditional spanish pajarita Figure 2: Ryu-jin dragon by Satoshi Kamiya.

As anyone who has ever folded any figure would have noticed, origami has an intrinsic geometry to it. The patterns that form in the process of creating a figure have lots of mathematical interest: symmetries, complex 2D and 3D fractals and tessellations. Endless engineering problems can be extracted from it.

Formalizing every fold that can be performed to a piece of paper poses a serious challenge that appears from the simplest of actions: folding a paper. It is not the aim of this project to do that but to create a simulator that can perform the most usual and traditional folds that allow to fold most figures. In this project only straight folds will be considered although curved ones are perfectly possible.

It supposes a challenge too in the computational area as it is not easy to visualize everything that happens during the folding of a paper and interact with it through a computer. But also the data structure that arises from a folded figure can be a rather complex graph connecting adjacent pieces sharing edges, co-planar, stacked on top of each other or sharing more vague properties like for example being part of the same leg of a model. It will be an important focus of this project to solve these problems.

Along with this project, the main mathematical tools will be those of Linear Geometry and some other basic concepts. In the engineering side, we will need some insight in Data Structure matters and software design and architecture as well as some notions of the use of computer graphics and user interface.

1.2 Motivation

The motivation for this project is very straightforward. In the origami community hours and hours are spent diagramming the models, which means writing down instructions, in a tedious and tiresome way, usually using tools like Illustrator or graphics generating tools to create the graphic parts. This tool facilitates this process and could be of tremendous use as it would be as simple as screen-shooting each step performed in the simulator. This

could be automatized further to get the diagrams directly after folding a model in this environment.

This project does not intend to archive this finished pipeline from simulation to diagram. It wants to be a first attempt or approximation to the first part, the simulation. We want to cover a few types of folds that allow folding very simple bases and figures.

1.3 General objectives

In order to solve that need this project aims to create a simulator with a graphic interface that allows the user to interact with a piece of paper as he would in real life. This means it has to be a 3D environment with non-ideal folds. For example, if a paper is folded in half both halves do not lie on the same plane, they are not parallel to each other and the angle formed by the fold is not 180 degrees but rather 179 or some smaller number. Considering the elasticity or endurance of the paper is beyond this project's scope but should be considered in more extensive work. The tool has to offer to the user enough diversity of folds to allow him to fold freely any figure he or she desires. Having this in mind, a list of different types of the fold will be compiled and we will work to implement most of them. This application must be a standalone and multi-platform tool and be able to run in any standard laptop.

This program intends to be a technical tool rather than a divulgatory or recreational one. It does not aim for any person to be able to fold at first try. This is a tool, for now, that aims at those who already know how to fold origami figures, those who are rather proficient in doing it. The interface and vocabulary are to be used by someone familiar with the type of folds that can be performed to a paper and the constraints they have.

Thus the interface is not designed to be intuitive for everyone but rather just for those who fold paper regularly.

1.4 Specific objectives

In order to attain our general goal we have to check some specific objectives to build upon:

- Analysis, definition and development of a finite set of valid folds formalized that allow the folding of simple figures.
- Founding and formalising a mathematical understanding of each and every fold, knowing the constrains that have to be applied to perform the fold as well as the data needed to perform it. Describing the mathematical tools that will be used in each one.
- Analysis, definition and development of a valid data structure that supports whatever figure the user aims to fold.
- Creating a valid visualization of the paper that matches what would actually happen in a real piece of paper
- Designing a user-friendly interface that allows the user to interact with the paper with ease and intuitively.

1.5 Planing

In order to accomplish our goals we have divided and scheduled the tasks as follows (See Figure 3) :

- Task 1: Learning how to create a simple Unity application and researching the mathematical foundations of origami
- Task 2: Listing the requirements of our application, our data structure and interface
- Task 3: Research in data structures and designing a data structure that checks all our requirements
- Task 4: Creating the representation/rendering of the paper and the interaction with it as View and Controller
- Task 5: Formalization of the mountain and valley fold and implementation
- Task 6: Implementation of the Finite State Machine that controls the flow of the application and manages the types of fold.
- Task 7: Expansion of the possible folds and interactions

1.6 Organization of the report

This report is divided in the following sections:

- **Introduction:** In the Introduction chapter it is intended to contextualize the project commenting shortly its main objective.
- **Analysis:** In the Analysis chapter all the requirements are detailed .
- **Background:** In the Background chapter we delve into the applications and models that already exist and find their strong and weak spots to improve our own.
- **Formalization:** In the Formalization chapter, the mathematical basis for the folds is developed.
- **Design:** In the Design chapter, the architecture and interface are described.
- **Results and Simulations:** In the Results and Simulation chapter the results of the developed project are shown.
- **Conclusions and future work:** In this last chapter we get to the conclusions and the project's state looking forward.
- **Appendix A : Technical Manual:** In this appendix all the steps to install or develop the application are detailed.
- **Appendix B : User Manual:** In this appendix it is explained how to use the application.

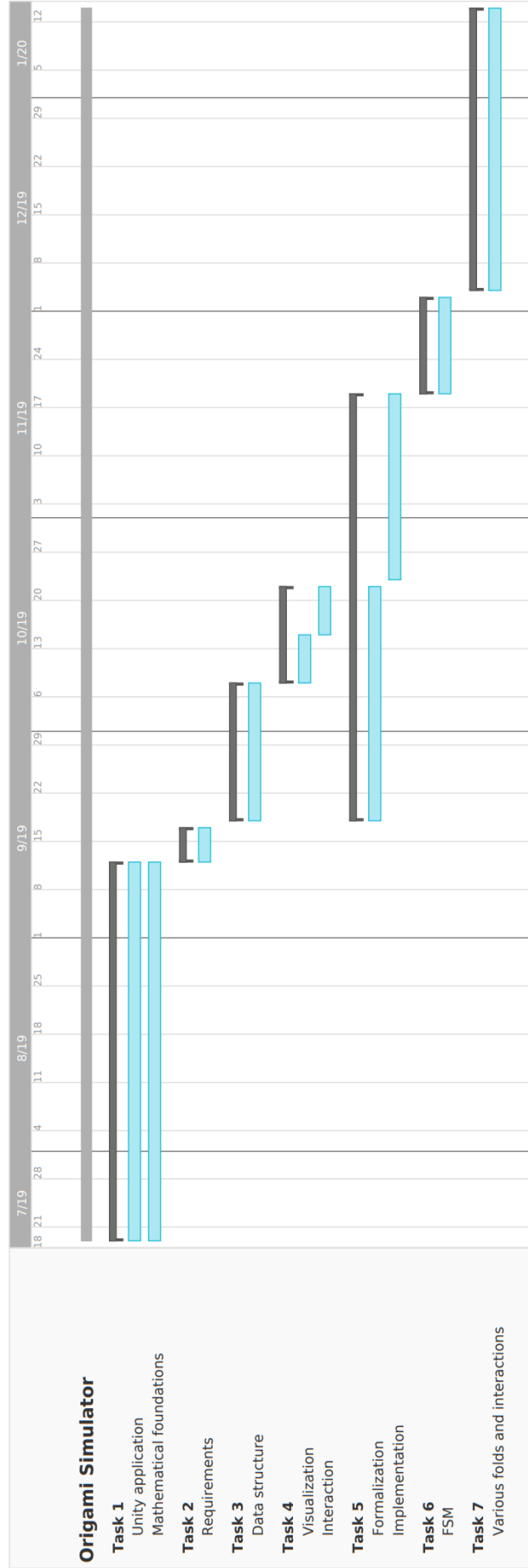


Figure 3: Time planning.

2 Analysis

2.1 Nomenclature

2.1.1 Origami

The world of origami is pretty specific and has its own vocabulary that will be needed to understand this report:

First of all, we need to define origami, which is the Japanese art of folding paper, the aim of it is to start with a flat square of paper (sometimes other shapes), to obtain a final model often representing an animal, human figure or object. A base of a model (Figure 4) is an intermediate state of the model that already has the flaps and tips of the complete model¹ (Figure 6).

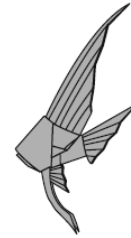
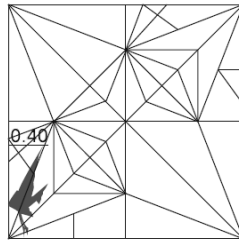
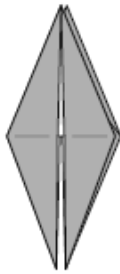


Figure 4: Base of the model Figure 5: CP of the model. Figure 6: Finished model.

Then, we need to differentiate different parts of a model. It has flaps and tips, long slim parts of the model, usually, a tip ends in one point and consists only of a triangle and a flap has a more complex shape. A crease is a previous fold that has been flattened, a mark remains and can be refolded but in the exact moment lays flat. A CP is the collection of creases of a model or base (Figure 5).

Moreover, we need to understand the different types of fold: the most simple two are mountain fold and valley fold (Figure 19). These are the most intuitive ones, the valley fold is a fold along a straight line that brings the paper to the folder, the mountain moves away from the paper from the folder.

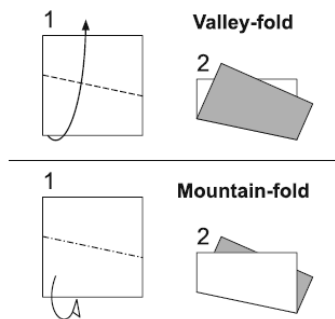


Figure 7: Mountain and valley folds.

¹All the images in this section are extracted from [5]

Additionally, we have 2 symmetric folds inside reverse and outside reverse (Figure 8) that have to be applied to a tip or flap and bring the tip inside itself or outside it symmetrically in both sides (in one side there is a mountain fold and in the other a valley and vice versa).

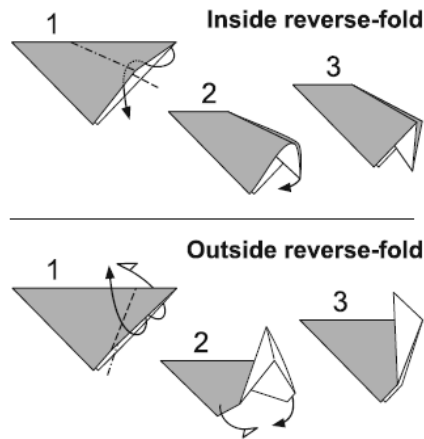


Figure 8: Inside reverse and outside reverse folds.

An open sink fold (Figure 9) is where you take a closed tip and fold it inside itself, all layers should stay in the same order they were before.

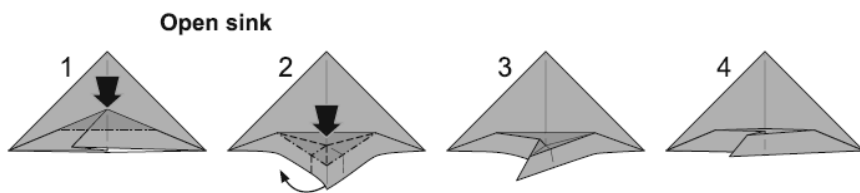


Figure 9: Open sink fold.

Finally we have the pull and move (Figure 10). Pulling is moving a tip of the paper creating a new crease in the direction of the pull. Moving is using an existing crease to move a part of the model but no creating any new fold.

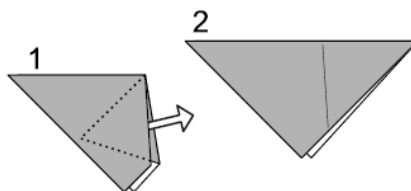


Figure 10: Pull or move.

There are more folds such as the swivel folds, twists, closed and mixed sinks, and petal folds or double ear rabbit. These will not be discussed as they are out of the scope of this project, however, the data structure will be designed having those in mind so they could

be supported in a future version. The list of folds presented are the ones that come to mind when folding a traditional figure and should be enough to fold simple figures.

Furthermore, there are operations that are not folds *per se* but can be seen as operations on the paper such as rotations, zooms in and out and more importantly unfolds and creases. We see an unfold as reverting the model to the previous state leaving the creases of the previous fold on the model. We define a crease as the operation made to make a mark in the paper as when you are folding but not moving any part of the paper.

2.1.2 Other important concepts

Finite State Machine It is an abstract machine that can be in exactly one of a finite number of states at any given time. The FSM can change from one state to another in response to some external inputs and/or a condition is satisfied; the change from one state to another is called a transition. An FSM is defined by a list of its states, its initial state, and the conditions for each transition [14].

Specular symmetry A specular symmetry is a map in linear algebra usually performed in a 3D space. It is a continuous isometry, meaning it maintains distances between points. It works in relation to a plane. It can be imagined as the same that happens in a mirror: everything gets to the other side of the plane (the mirror) but at the same distance from it as it was before.

2.2 Functional requirements

In order to achieve the desired results we need to have certain functionalities included in our application:

- Interface: In order to perform the desired operations we will need an adequate environment like the one presented in Figure 12. A window that allows users to see the 3D model, rotate and zoom in and out and move around it. We will need a menu that allows like the one presented in figure users to select which kind of fold we want to perform.
 - Rotate camera
 - Zoom in and out
 - Center camera

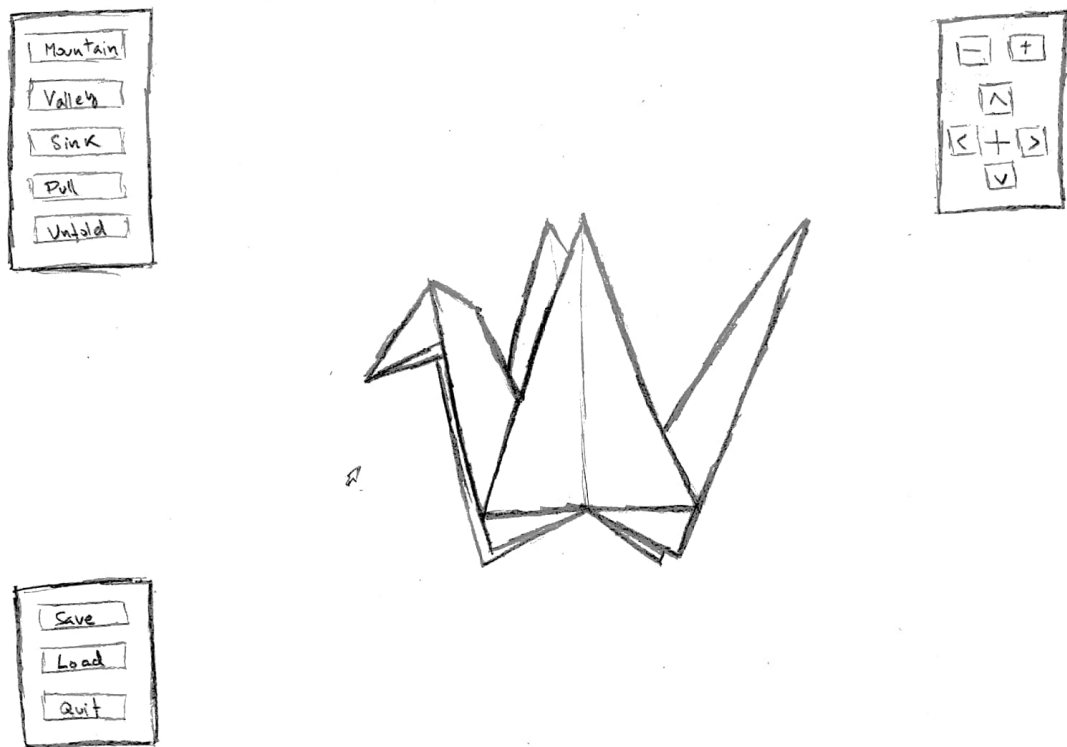


Figure 11: Early concept of the interface .

- Interaction: We need to be able to distinguish each part: vertices, edges and faces. We need to be able to select them and select points outside the model. There has to be a visual feedback to this selection.
 - Select vertex
 - Select edge
 - Select random point
 - Deselect
- Folds: we need to be able to use the folds described in the section above and use them easily and consistently (the same fold has to have the same result). Also we need the main window (the one containing the model) to represent the folds as users mark them and then needs to change (with or without animation) into the folded model.
 - Mountain fold
 - Valley fold
 - Open sink
 - Unfold
 - Pull
- File related operations: It has to allow you to save and load a project so the model can be shared conserving the data structure that allows you to view the full process in succession. We also want to undo and redo our actions.

- Save
- Load
- Undo
- Redo
- Play

2.3 Technological requirements

Regarding software we will need a relatively modern laptop or desktop computer with either Windows, MacOS or Linux running. It does not need to be extremely powerful or have a very modern graphics card. Having a mouse will help to better the user experience but using a trackpad is perfectly fine.

Unity requires the following stats to run any game (Figure 12):

System	Minimum requirements
Desktop	
Operating system	Windows 7 SP1+ macOS 10.12+ Ubuntu 16.04+
CPU	SSE2 instruction set support.
GPU	Graphics card with DX10 (shader model 4.0) capabilities.

Figure 12: Specific requirements [12].

2.4 Developing software justification

In order to accomplish the functional and technological requirements Unity [13] with C# has been chosen. Unity facilitates a lot the rendering of the model and the UI. It already has ways to deal with mouse clicks and positions, it handles keyboard events and has a very easy way to implement buttons. It makes it very easy to make any kind of application including desktop for all OS which is our goal. It directly generates the .exe file. But most of all it has already scene creating capabilities including the creation, texturing, and rendering of simple 3D objects such as spheres, cylinders, and meshes. It also creates the camera and lighting sources easily and lets you interact with these objects through C# code.

Regarding C# it is object oriented and offered by Unity so it was the reasonable choice to make.

3 Antecedents

3.1 State of the art

Before going into more technical aspects I would like to point out that the articles presented here are not the only source of knowledge that is applied in this project. Some works have been particularly helpful in building the technical understanding such as [9] written by Robert J. Lang and some other very inspiring in pursuing this art such as Román Díaz, Quentin Trollip, Eric Joisel, Nicholas Terry and of course Yoshizawa Sensei. As it is said I have been able to get here because I was standing on the shoulders of giants.

3.1.1 Data structure

The field in which researching has proven more useful is in the structuring of the information contained in a model.

This paper [16] describes a three part data structure consisting of a binary tree for the faces, a list of binary trees for the edges and a list for the vertices.

This model has very strong points as it allows to recover all the history of the model and links the elements of the model so it is possible to know the origin of each one. You can know the original edge from whence an edge comes. However the vertices are totally unrelated to each other having in mind that they do share things. When a vertex v_1 is moved we create a new one v_2 unrelated to the first one. In an unfolding scenario knowing that the previous position of the v_2 to be v_1 it would be very helpful.

The paper also discusses the curved folds and how they relate to tensions applied to the paper, in the introduction these aspects have been dismissed as they dive too deep in physics of origami and curved folds so we will not be using it.

3.1.2 Mathematical foundations

This papers [15] [11] illustrate the constraints in concurring valley and mountain folds into a single vertex. It contemplates not only planar and quasi-planar folds but folds in all angles in a rigid origami system and could be used for further work.

This first one [15] does so through quaternion algebra. It discusses the foldability of several folds concurring in a single vertex with folds that turn not necessarily 180° . It checks for face intersection and analyzes the 3D result.

This second one [11] recurs to lineal algebra, more specifically to affine transformations to discuss the foldability of a CP. It analyses if a CP is foldable (if it is possible to fold it without tearing any face or them intersecting each other) and whether or not it lays flat once folded.

This papers [15] [11] though very interesting do not concern simple folds, just collapses of more complex and simultaneous folds and are beyond the scope of this project.

Some other papers have been reviewed [17] [2] discussing the constructions and numbers, work related to constructibility problems that are not relevant for our work at this point. It centers in algebraic questions and problems and do not help at all with the task at hand.

3.2 Similar applications

In this section, we will explore some applications that perform some of the tasks that we want our simulator to accomplish. There will be many interesting things to be learned from them. Their strong points can be incorporated in our simulator and the weak ones can serve as things to avoid.

3.2.1 Rabbit Ear [7]

Named after a famous fold in origami, Rabbit Ear (Figure 13) is maybe the closest thing to what I want to achieve. It is created by Robby Kraft, a relatively well-known artist in the origami community. It is a web application that allows the user to fold a flat paper using mountain folds and shows the progress, the final shape, and the CP. It has a very nice and clean interface. It lets users download the .fold file, a custom file type that is allowed to later be loaded in the application and also to print your progress. It presents a two-colored square paper that can be folded by dragging from any point on the screen. It is an open-source project.

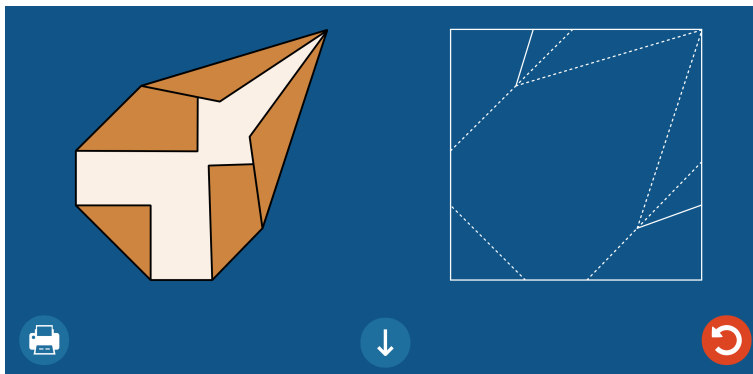


Figure 13: Rabbit ear web [7].

However, in the current version, it only allows mountain folds folded on a flat surface. There is no 3D view or more complex folds. It is also not reference-based, meaning that you do not fold from edge to edge or vertex to vertex. Users can only fold in a free-hand type of fold. This makes it really difficult to make any precise fold. In the end, it is a very successful first step to a very complete simulator. It is currently under construction so it may implement lots more of functionalities in the near future. However, in the current state, it is not very useful even if it is very usable. Some very simplistic and basic models can be folded from it but not any serious model. It also increases the feeling of limitation that there are no unfolds. If there were those and valley folds the application would be much more complete. The website shows some promising samples of code that is integrated into this simulator would make it much more powerful.

In conclusion, our simulator aims to create a program that expands the utilities of this one into a 3D world and lets you perform more folds. The CP real-time display is not at all a priority and the real-time adjust would be something nice to have but is not a priority either.

3.2.2 Tree Maker [8]

Tree Maker (Figure 14) is a software that helps origami designers in their way of finding a suitable base for their model. It lets the user introduce a stick figure representing the flaps, tips, and sections that they want their model to have and then it creates a crease pattern and folds it into a base showing the final result. It is a very powerful tool as it lets the user apply constraints to the flaps choosing if they want it from a tip, edge or the center of the paper and it is widely used by the origami community. It has been developed by Robert J. Lang, an engineer and one of the most famous origami artists in the world. The application is free to download and runs on any desktop computer and any OS.

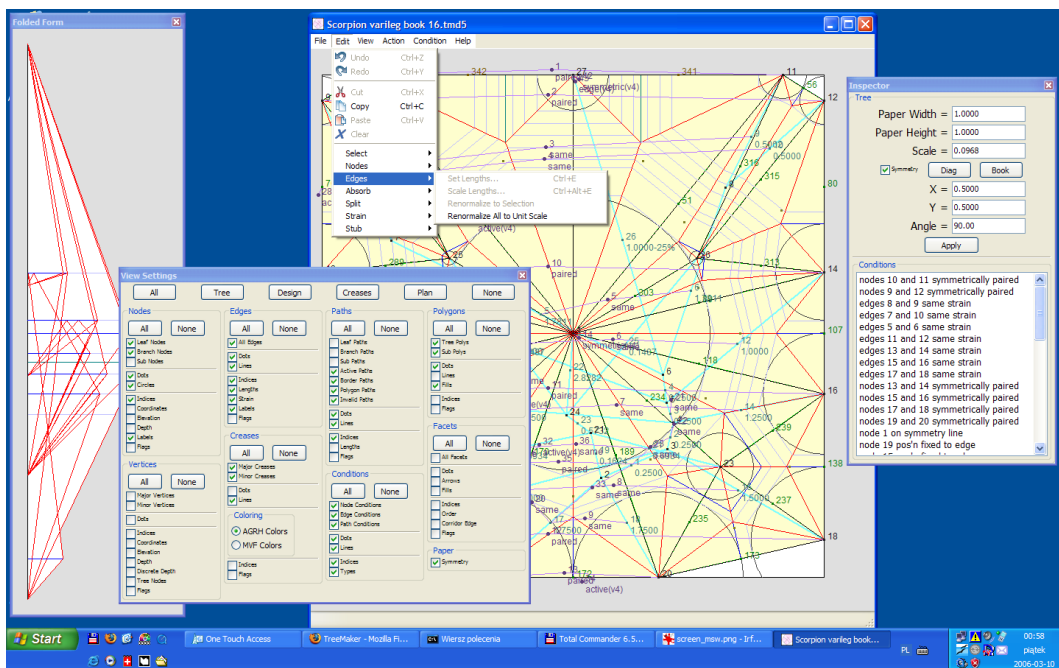


Figure 14: TreeMaker running [8].

Unlike the previous application this one allows for any kind of fold. It is a finished product, not an ongoing project so it is much more useful. It allows for very complex bases and models. However, it only allows creating bases (it can be seen in Figure 9 on the left) not complete models. There is no artistic component, it is purely a tool to facilitate the making of figures. It shows the CP but it does not show the steps to get to the base. The user usually has to recur to tools such as Reference Finder by the same Lang and Robby Kraft to be able to start folding. It is tedious work but sometimes the best way to get to your figure. From a mathematical and engineering point of view is a true wonder. However it only considers flat folding in a 2D space and it is not an easy tool to use, it gets time to understand it and being able to use it even if you are familiar with casual and even technical origami.

In conclusion, this is a much more complete application than Rabbit Ear, however, its intent is further away than what we want. It does not try to simulate any real-life folding experience, it is just a tool to design figures. However, it includes a wide variety of folds and infinite possibilities.

3.2.3 Tess [4]

Tess (Figure 15) is an application created by Eric Gjerde, one of the most recognized tessellation creators in origami. This application lets the user create origami tessellations (repetitive geometric patterns). It lets the user choose from the different possible basic shapes combinations, merge them, distort them and resize them to obtain your final piece. Then it can show the user the tiled version (how it would look when finished in 2D), the CP and the light pattern, which is how it would look when placed in front of a light given that the paper has some transparency and it is a very important aspect to consider when designing a tessellation because of its aesthetic impact. It is a desktop application available for free in all OS.

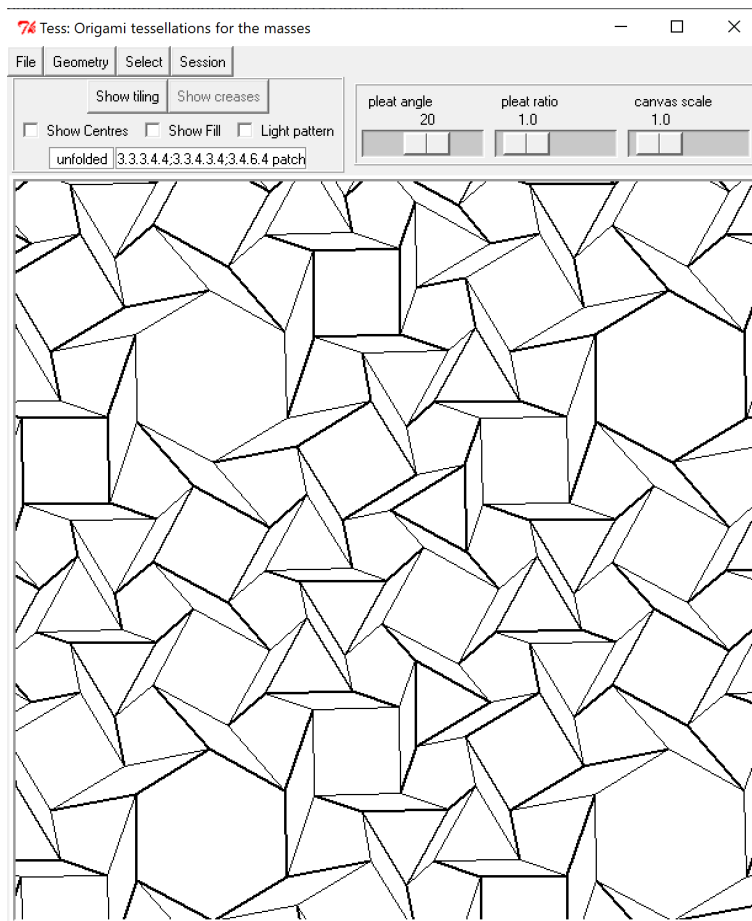


Figure 15: Tess running [4].

This application is as the previous one a complete product, very useful although, like the previous one not super usable. It permits a very wide range of possibilities and the 3D limitation results omissible as most tessellations are completely flat. It contemplates all kinds of folds and as folding a tessellation is not very conceptually difficult, although it may be hard to do, omitting the steps is not a handicap. However, it lacks too the simulation experience. It is again just a tool previous to the folding experience opposed for example to Rabbit Ear.

3.2.4 How to make Origami [1]

How to make origami (Figure 16 and 17) is one in a lot of mobile apps that teach the user how to fold certain figures. This one has been chosen between the lot that exists as it has the highest rating. They do not have many conceptual differences, just presentation so this one will do just fine. In this kind of app, the user selects a model that he or she desires to fold and then the app shows a series of steps or animations to follow the process along. This one, in particular, shows 3D animation that looks like it is made from a series of processed photos.

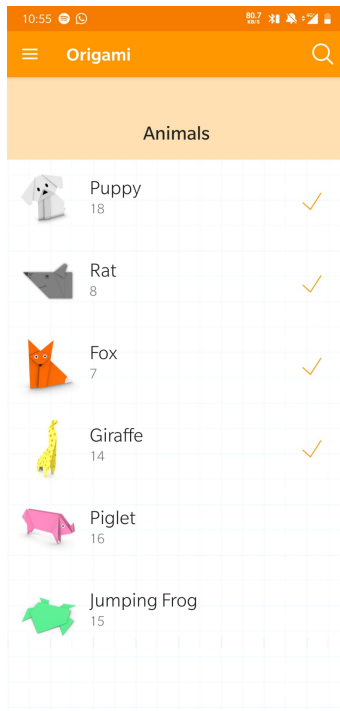


Figure 16: Menu with the available models

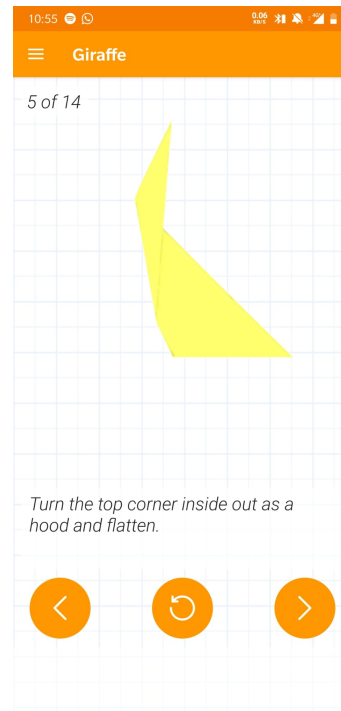


Figure 17: Step 5 of a giraffe model.

This is all well and good but it lets the user no freedom. The visualization is very clear 3D and fluid. However it is limited to a few basic models and has no real interaction with the paper.

3.2.5 Non-software alternatives

Even if this is a computer science project it is interesting to argue the need for this kind of simulator. Nowadays there are much easier and more spread ways to convey an origami model. Paperback diagrams, be them photographic, manually or digitally drawn are much more common and they are easy to understand thanks to a very useful set of sings improved during decades. Video tutorials are very popular tools too. And of course, real teachers and one's trial and error are sometimes the best and most satisfying ways to learn. So let us argue the need for this tool opposed to this more traditional methods.

Paperback diagrams

For the last decades the origami community has spread most of the models through diagrams (Figure 18), most of them drawn using computer drawing tools. They are clear thanks to a lengthy set of symbols and signs that have been developed and perfected during the years. They are very useful and easy to spread being in digital or paper form. They do not necessarily need to have written instructions but consume a lot of time to draw and in slightly complex models can be very difficult to draw. Moreover, the reader cannot turn the model or change its angle to recognize it in his paper.

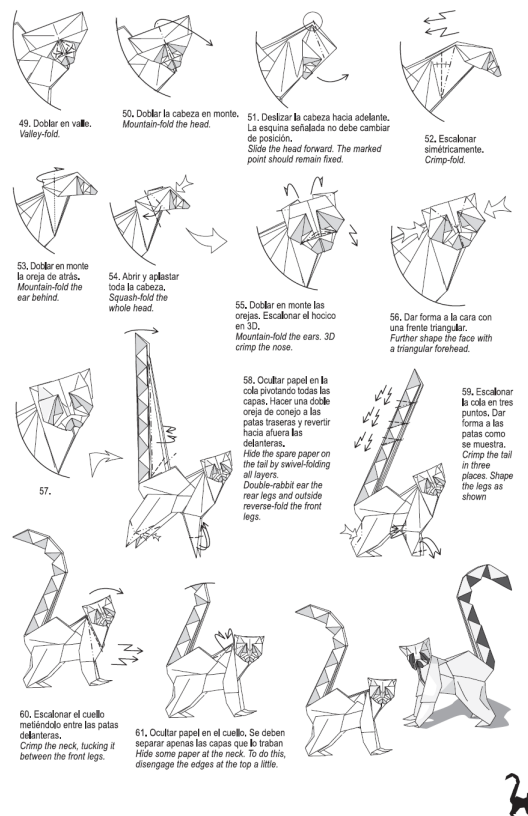


Figure 18: Typical diagram instructions [3]

Video tutorials

Another way to teach models these last decades has been video tutorials (Figure 19). They range from very amateur to professional and they are sometimes better and less time-consuming than diagrams as it just takes to record them the time it takes to fold the model. However there are inconvenient to them, the most important of them is that oftentimes the hands of the folder interfere in the vision and understanding of the spectator. Good YouTubers reduce this effect a lot by changing the angles of the camera and paper and zooming in and out. It is still a problem most of the time. As it happens with diagrams the user can not see the model from all points of view and sometimes that can be a problem.



Figure 19: Video tutorial for a squirrel [10].

Real folding

Though maybe evident this is the easiest and best way to learn how to fold a model through direct learning (Figure 20) from someone who already knows how to fold it. It has very clear inconveniences such as having to be in the same place as the teacher and the amounts of times that require to fold some models make it nearly impossible. However, there are more subtle inconveniences especially when you are test-folding a model (folding it in a less attractive, usually less expensive paper to work out the harder parts) most of the times you fold some parts over and over to achieve the best result and the paper wears out and tears, it is sometimes very hard to find a big enough paper or the space to fold it comfortably. Added the waste that can be generated from all the attempts in designing a figure and the space that takes to keep them. However when possible people still gather to fold together in conventions and events.



Figure 20: Origami convention class.

In conclusion, it may be very useful to have a tool that lets you fold anywhere, any model, any size, any type of paper and save it, diagram it, load it and send it. It would make the lives of all us who love to fold paper much simpler.

3.3 Conclusion

In this section we will compare all the applications and methods seen and draw their strong points to help design our simulator. We can see that in the following table that the more technical software lacks user-friendliness and 3D representation and the apps that mean to teach specific models to stay too simple and with very limited models.

Regarding the more traditional methods although they have very strong points they have serious weaknesses. Paperback diagrams lack a 3D and are very time consuming, video tutorials often obstruct the view and real folding has the inconvenience of having to be present.

	3D	All kinds of fold	Easy to share	Unlimited models	User friendly	Clear view	Fast to create
Rabbit ear	✗	✗	✓	✓	✗	✓	✓
Tree Maker	✗	✓	✓	✓	✗	✓	✗
Tess	✗	✓	✓	✗	✗	✓	✓
How to make origami	✓	✗	✓	✗	✓	✓	✗
Paperback diagrams	✗	✓	✓	✗	✓	✗	✗
Video tutorials	✓	✓	✓	✓	✓	✗	✓
Real folding	✓	✓	✗	✓	✗	✗	✓
This project	✓	✓	✓	✓	✓	✓	✓

The proposed software will have to solve the problems all these methods have to present a product useful, easy and fast to use and with very few limits so the folder can use it as he or she would a piece of paper.

4 Formalization

In this section we are going to discuss directly the mathematical tools that intervene in this simulator. We will discuss the methods used to find the symmetry planes in the folds, and why we are using symmetries in the first place. We will look into the fold itself as a map and to other aspects concerning the mathematical and physical constraints behind origami.

In most of the simple folds (mountain, valley, inside reverse, outside reverse, and open sink) the vertices move according to a specular symmetry, the plane of this specular symmetry is computed as it is mentioned in the next section and slightly tilted depending on the fold we are performing.

This slight tilt is due to the thickness of paper if it were a perfect paper with no thickness to it the symmetry would be concerning the plane without the tilt and the faces would stand in the same plane when executed.

In mountain folds we need the moved face to be closer to the camera than the one that remains still. To achieve that we need to add to the normal of the symmetry plane a small vector that has the direction from the model to the camera. In valley folds, we proceed in the same way but with the opposite vector. In open sinks, there is no tilt as the sunken tip will remain inside the model, between the furthest and closest face.

4.1 Finding the symmetry plane

In order to execute our fold we need to find the symmetry plane π and in order to find it we need 2 things: a vector \vec{n} normal to the plane and a point p that belongs to it.

We can fold from:

- **Vertex v_1 to vertex v_2 :** In this case $\vec{n} = (v_1 - v_2)$ and $p = (v_1 + v_2)/2$
- **Vertex v_1 to point q :** We proceed in this case taking $v_2 = q$ and operating as in the previous case.
- **Edge e_1 to edge e_2 :** When we fold from edge to edge it gets a little more difficult. First of all both edges must be co-planar, and they can be either parallel or intersect.
 - If $e_1 \parallel e_2$, let π_e be the plane formed by the two edges and \vec{n}_e its normal vector then $\vec{n} = \vec{e}_1 \times \vec{n}_e$. The base point can be the midpoint between a vertex of each edge.
 - If e_1 intersects e_2 then \vec{e}_1 in the previous case needs to be the bisector \vec{b} of e_1 and e_2 . To get this bisector, given that the edges do not have a direction, we need to find the point of intersection of the prolonged edges q and then form both vectors \vec{e}_1 and \vec{e}_2 from this intersection point to a vertex of the edge and then we get $\vec{b} = \vec{e}_1 + \vec{e}_2$. In this case $p = q$.

Or we can crease from:

- **Vertex v_1 to vertex v_2 :** In this case $\vec{n} = (v_1 - v_2) \times \vec{n}_f$ where \vec{n}_f is the normal of the face if the two vertices share one, if they do not we will need to use the vector from the observer to the model, which may be the closest approximation we have. The base point p can either be v_1 or v_2 .

- **Vertex v_1 to point q :** We proceed in this case taking $v_2 = q$ and operating as in the previous case.

4.2 The fold topological map

In this simulator the paper is represented as we have seen as a list of vertices, edges, and faces. However, as we are operating with a symmetry that maintains the distance between the vertices and thus we do not need to save the lengths we see this structure as merely topological. So when we fold and unfold we only need to change the topology of the structure and apply the symmetries.

When a new fold is performed (a new crease is added to the paper), the topology changes. We need to add vertices, edges, and faces in the following way (See Figure 21):

The new vertices are created in the intersections between the plane of symmetry and edges. Anywhere this happens we add a vertex that splits the edge and becomes origin and end of each half of the previous edge. To create edges we need to check if the generated vertices share a face pairwise. If they do we create an edge that joins them and we split the face in two. The vertices that fall in one side of the symmetry plane and those in the new edge create a face, the ones that fall in the other with those in the new edge create the other.

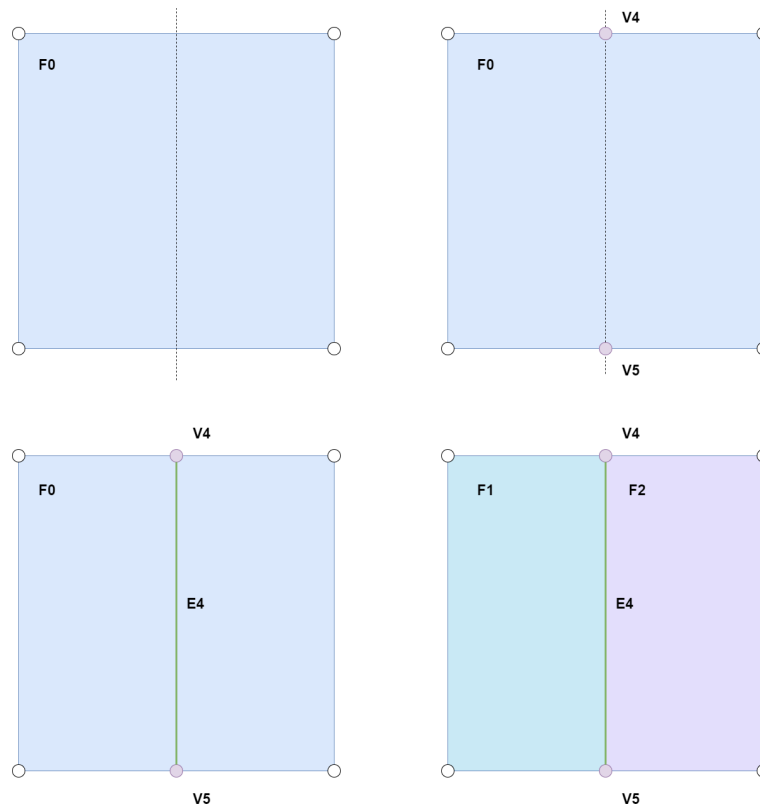


Figure 21: Steps of the folding process.

When this process has ended we will apply the symmetry to the vertices in one side of the plane.

4.3 The fold geometric map

Before starting we need to remember some linear geometry concepts:

Proposition 1: Having defined the original basis as the canonical and changing into an orthonormal one, the matrix of the change of basis M has the property that $M^{-1} = M^T$.

Annotation: In this program we are working with points in \mathbb{R}^3 but we will treat them as if they were in \mathbb{A}^3 as it is a direct inclusion and it will lighten the notation.

The fold itself as an map is defined as follows:

$$\begin{aligned} F : \mathbb{A}^3 &\longrightarrow \mathbb{A}^3 \\ p &\longmapsto q \end{aligned}$$

The map can be decomposed as the following:

$$\begin{aligned} F : \mathbb{A}^3 &\xrightarrow{C^{-1}} \mathbb{A}^3 \xrightarrow{P} \mathbb{A}^{3^-} \xrightarrow{i} \mathbb{A}^3 \xrightarrow{C} \mathbb{A}^3 \\ p &\mapsto C^{-1}p' \mapsto SC^{-1}p' \mapsto SC^{-1}p' \mapsto CSC^{-1}p \end{aligned}$$

\mathbb{A}^{3^+} and \mathbb{A}^{3^-} define the positive and negative affine spaces ($x < 0$ and $x > 0$)

Where C and C^{-1} are a base change that bring the space to the desired symmetry plane and S is a specular symmetry.

$$S = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, C = (V_n, V_{d1}, V_{d2}, p)$$

V_n is the normal vector to the plane of symmetry, V_{d1} and V_{d2} are two of its director vectors and p is one of its points. The map P is defined as follows:

$$\begin{aligned} P : \mathbb{A}^3 &\longrightarrow \mathbb{A}^{3^-} \\ p &\longmapsto Sp \quad \text{if } p \in \mathbb{A}^{3^+} \\ p &\longmapsto p \quad \text{if } p \notin \mathbb{A}^{3^+} \end{aligned}$$

This map preserves the distance to the symmetry plane, and thus can be applied to the physics of origami. I want to remark this map only moves half of the space, the other half remains still. Thus it cannot be looked as an homeomorfism.

5 Design

5.1 Proposal

The design of the main application is based on the classic software design pattern MVC (Model-View-Controller) (Figure 22). The Model will contain the figure, the position of vertices, edges and faces; all their important information and global information of the figure. The Controller will manage all the interactions of the user with the program and control the state of the simulator. It contains the data structure that holds the model, a camera and a plot that contains the view. The View contains all the element objects, the camera and all the other objects in the scene (lights, canvas, panel, and buttons). We will describe each module separately.

5.1.1 Model

To support the specified folds and objectives we need to design a robust, fast and memory-efficient data structure that represents our origami model in every one of its stages. A model consists, mathematically, of 3 distinct and simple elements: vertices, edges, and faces. Each of these require a different data structure and the three must be connected in specific ways in order to make searches and modifications to the structure easy and fast, as a complex model one fold may contain hundreds of elements.

The proposed model will have a tree representing the faces of the model, a list of binary trees representing its edges and a list of lists representing the vertices (Figure 23).

Faces

As in each fold each face may only be divided, no faces are created, a tree is an ideal way to store them. Considering that we start with only one face and that the new faces will only be divisions of it. It not only gives us the relation between faces but also can store the folding process as each new level in each branch represents a new fold.

In each node must be stored the reference of the face, the plain it represents, and the vertices that bound it. It has to contain too in which step it was created in order to be able to recover in case of an unfolding. It must contain too, a list of provisional vertices that is meant to be used in the `generateCrease` function.

Edges

Similarly as it happens with the faces each time that a fold crosses an edge it divides only in 2, however, in this case, edges can also be generated. With this in mind the best solution is for it to be a list of binary trees that behave as does the one that contains the faces.

Each node of the trees must contain the information about an edge meaning its reference number, the two faces that it bounds and the vertices where it starts and ends. It has to contain too in which step it was created in order to be able to recover in case of an unfold.

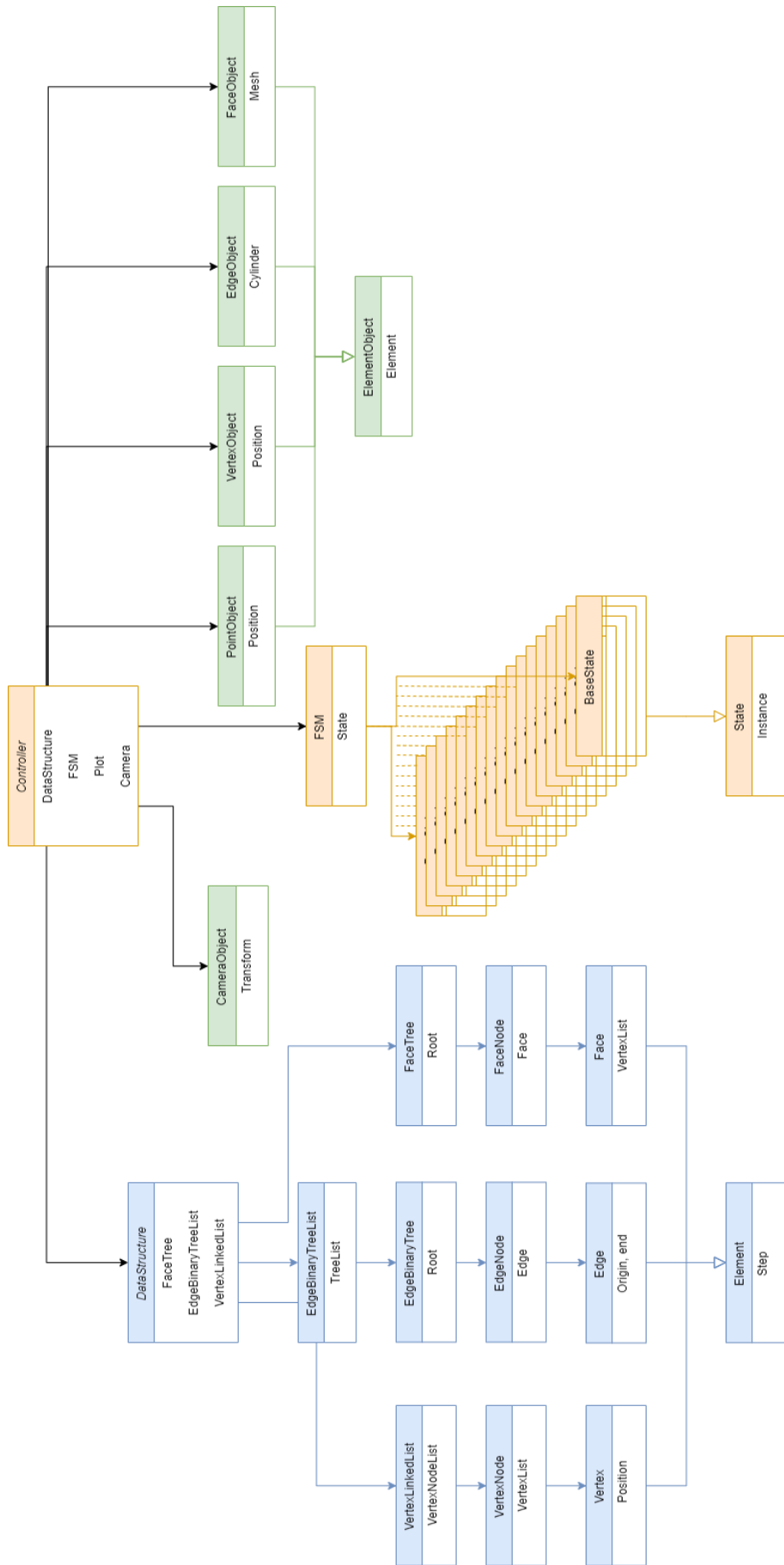


Figure 22: Model (blue), Controller (orange), View (green).

Vertices

In the case of vertices they do not follow any specific order. Each time one is created it will be in the line of the fold, thus it will not be moved in that step. However, we will save a vertex in its list in a node when we change its position and only the last element of each list will be displayed. In this fashion we can retrace the position of each vertex along the folding sequence.

For each vertex node we will need to save the list of positions and the edges that start there. We need also to save the step in which position is created for retracing purposes.

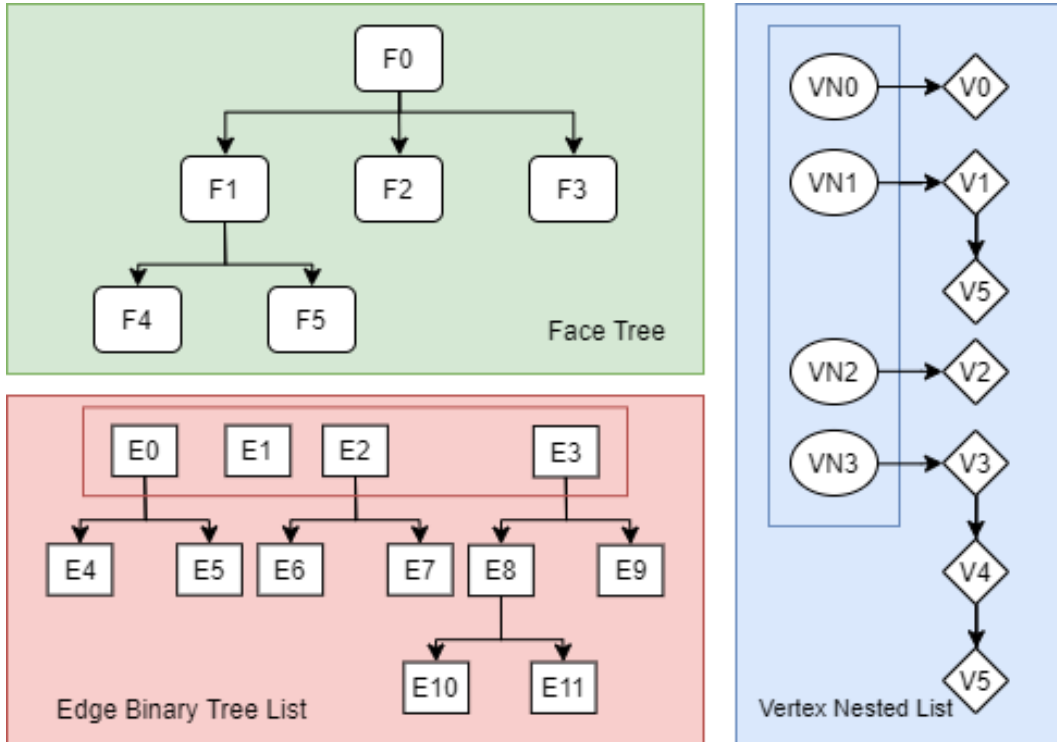


Figure 23: Example of a possible data structure.

Main functions

The constructor of this class initializes all the model creating a paper square on 2x2 units (like the one seen in Figure 24). Creates the 4 vertices, 4 edges and the square face. Initializes the lists and trees so everything is ready to start folding.

Then we have all the folding functions: mountain, valley, sink, pull and move. They all need to generate a symmetry plane π , generate the new creases and apply the symmetry. let us review each step:

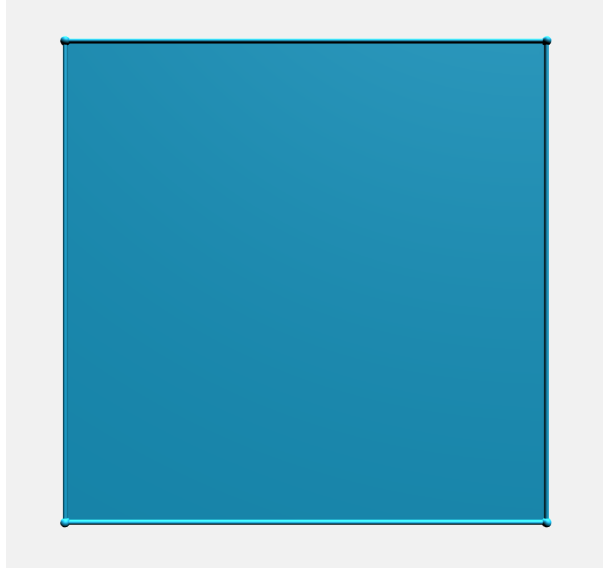


Figure 24: Initial structure.

Similarly as it happens with the faces each time that a fold crosses an edge it divides only in 2, however, in this case, edges can also be generated. With this in mind the best solution is for it to be a list of binary trees that behave as does the one that contains the faces.

Each node of the trees must contain the information about an edge meaning its reference number, the two faces that it bounds and the vertices where it starts and ends. It has to contain too in which step it was created in order to be able to recover in case of an unfold.

When generating a plane (Figure 25), as commented in section 4, it can be from 2 vertices, 2 edges or a vertex and a point. This function is overloaded and always returns a plane.

The function `generateNewCreases` that generates the new creases is the most complex in the program (Figure 26). First of all, we get all the edges $\{e_i\}$ that get cut by the symmetry plane π . Then we generate vertices $\{v_i\}$ in the intersection points and split the edges $\{e_i\}$ that cut it into $\{e_{1,i}\}$ and $\{e_{2,i}\}$. We refresh the origins and ends of the edges and the lists of edges that stem from all vertices affected. We also need to add the new vertices to the list of provisional vertices in the faces that the edge bounds. We have to consider that if π cuts a vertex $\{v_k\}$ we do not have to generate a new one and we have to consider it in the process that follows. Now we need to create the new edges, the new creases made in the paper. For each pair of the new $\{v_i\}$ and the cut vertices $\{v_k\}$ we check if they share a face F . This is where that provisional vertex list comes in handy. If they do we need to create an edge that goes from one to the other. Then we need to split the face F that they share. The origin and the end of the edge will go to both faces and the rest only to one depending on the side of the symmetry plane they are in. Then we need to refresh the faces on each edge. If they had the parent we need to substitute it for the correct child. Once we have done all this, we need to introduce all these new generated elements to their respective structures.

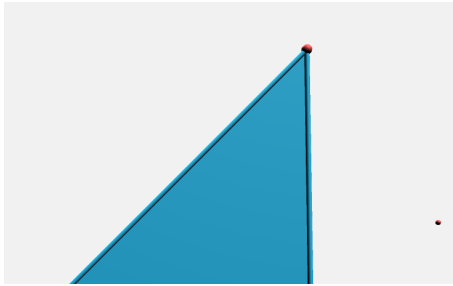


Figure 25: The reference points for the fold are selected.

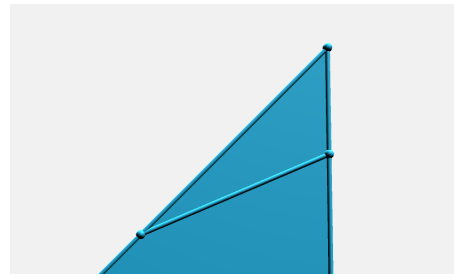


Figure 26: The vertices and edges are created and the faces split.

The function that applies the symmetry (Figure 27) works as described in the section 4 and is used on all the vertices in one side of the plain for mountain, valley, and sinks but for only one vertex in the pull and move.

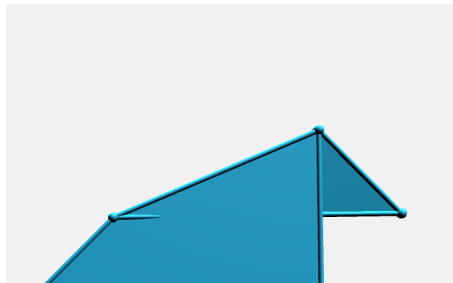


Figure 27: Complete fold.

Then we have the `crease` function works a little bit differently in that the symmetry plane π has to be computed differently as stated in section 4 and that it does not apply the symmetry like the other folding functions. It would be the same as folding and unfolding. It is very useful in some cases.

The `unfold` function is a tricky one. Unfolding once is an easy task, you just need to get the $(n-1)$ -th element in the nodes that were changed in the last step. However, if you want to unfold more than once there will be vertices that did not exist in that step but you have to keep. This problem has not been solved in this project but should be solved in future work by calculating the distance to the neighbor vertices and making it lay in the line between those preserving those distances.

Then we have the get functions that return the active vertices, edges, and faces. In the trees, we need to get all the leaves and this will be enough, in the vertices we have to get the last added element to each vertex node. We also have the functions that get the active elements in the n -th step.

5.1.2 Controller

The controller is the hub of the code. It contains an instance of the data structure and the finite state machine (FSM as an abbreviation), it controls the view as it creates and destroys the `gameObjects` in a plot variable and finally, it holds the data of the interaction with the user.

Finite State Machine

To achieve a fluid interaction with the user this application implements a finite state machine that controls the actions of the user in the interface and directs the program's flow. Each interaction of the user with the machine is viewed as an input and is managed by the machine to achieve a new state. Each state represents a sequence of interactions, a step that leads ultimately to a fold or some other action.

The machine takes the controller as an argument to give it to the states so they can interact with it. During the update cycle of the controller, it executes the machine that executes its current state. So as long as the program is running the FSM is running too and so is its active state.

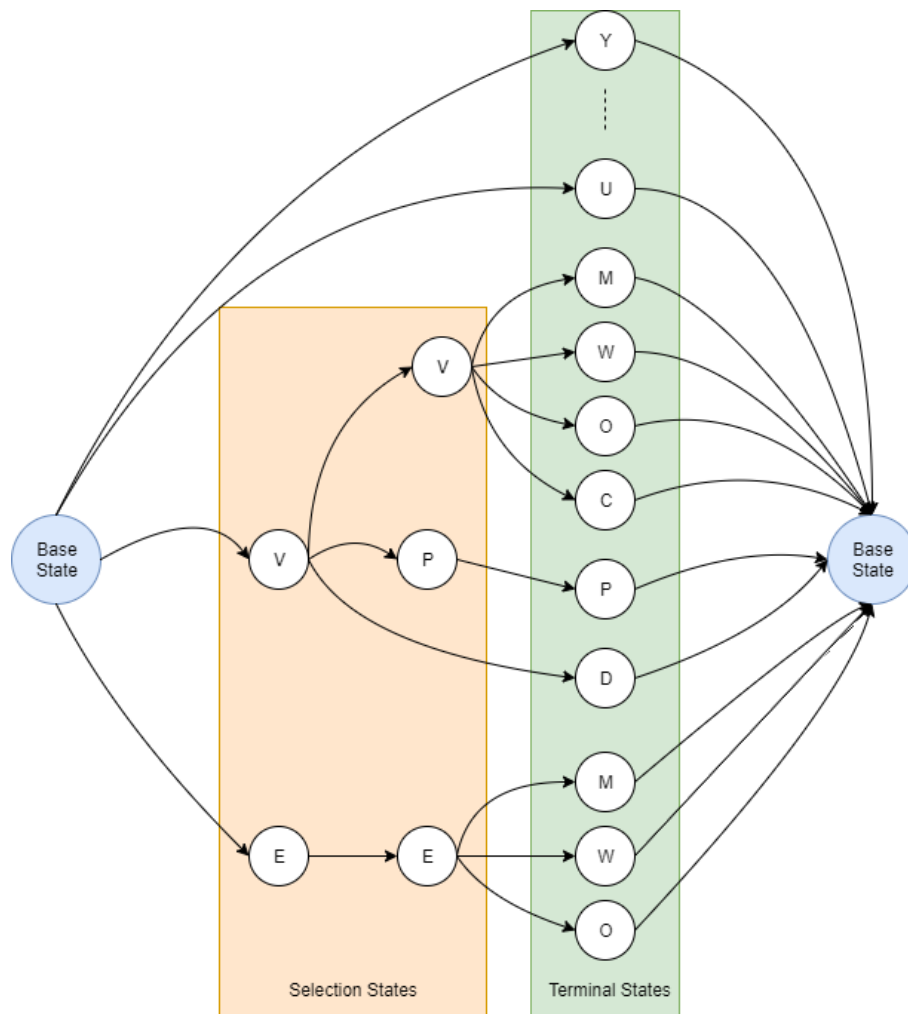


Figure 28: A glimpse into the state machine. The letters in the terminal states correspond with the action shown in the User's Manual. V, P and E are the states where a vertex, a point and an edge have been respectively selected.

We have 3 types of states (Figure 52). The base state in which nothing is selected and from which all folds can be achieved. We can return to this state on any occasion we want and all the selections made will be forgotten. This state can last for as long as it needs, it will hold until the user interacts with the application. The intermediate step is

only used to store the information and type of information the user gives. This state can last too for as long as it needs, it will hold until the user interacts with the application. And finally the terminal states. In these, the program has all the information that needs to perform a fold or some other actions and once it is done it will refresh the view, forget the accumulated information and get back to the base state .

Plot and destroy

The functions that translate from model to view are plot and destroy. In the plot function, the controller gets the active elements from the model and then generates `gameObjects` and matches to them the correct class. It assigns them their corresponding element and saves them in the plot list. To "animate" the transition when a fold happens the controller destroys every element in the plot and then gets all the active elements and plots them with their new positions.

Save and Load

In order to save and load we serialize all the data structure and then convert it to json and save it in a json file. We recover the file and translate it back to a `DataStructure` object in order to load. We have chosen json because it is a readable format outside the application, it can be understood in a plain text editor, it can be edited manually if desired and ultimately it could lead to creating or modifying the figure outside the simulator's environment.

State control functions

There are a lot of functions that serve to control the selected items by the user. They save them when needed, clear their data and reject adding them when there is no more room. Some functions control when the buttons are pressed. For each button a state in the FSM is triggered just as what it happens with the keys in the keyboard.

5.1.3 View

Elements

There are four elements in the view: `PointObject`, `VertexObject`, `EdgeObject` and `FaceObject` (seen in Figure 29). All 4 inherit from `ElementObject` which contains the methods to paint them and reset them. In the case of the point, it only paints red or disappears. In the other three cases, it paints them red (Figure 31) when selected and back to blue when not (Figure 30). It has been decided to use spheres and cylinders instead of lines and points as it is a 3D simulator and using 2D elements may have caused difficulty in seeing them from certain angles.

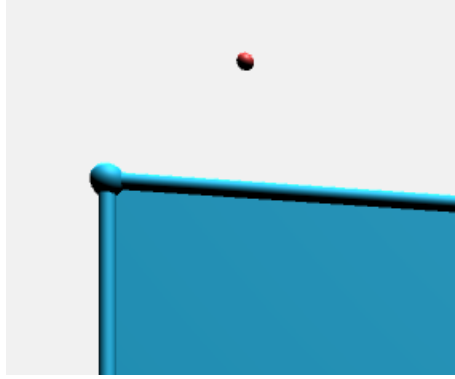


Figure 29: All four elements displayed.

The points are represented by a small sphere. They are used to represent a random point selected by the user. These points are where the paper will go when folded. Once they are unselected or folded they need to disappear.

The vertices are represented by a sphere, a slightly bigger sphere. When clicked they do not only need to change color but their corresponding vertex in the model has to be sent to the controller.

The edges are represented by a cylinder. To create the cylinder in the correct spot we need to calculate the vector from end to origin and then transform it to a quaternion rotation from the original position $(0,1,0)$ to the desired one. When clicked they do not only need to change color but their corresponding edge in the model has to be sent to the controller.

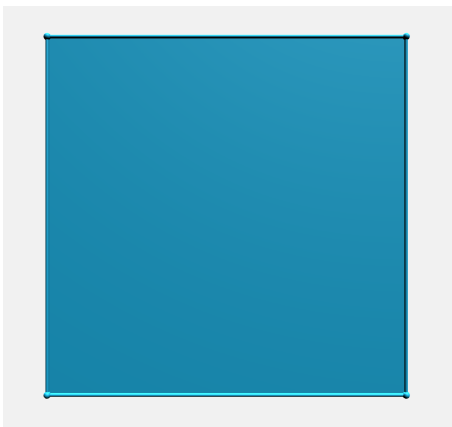


Figure 30: No elements are selected.

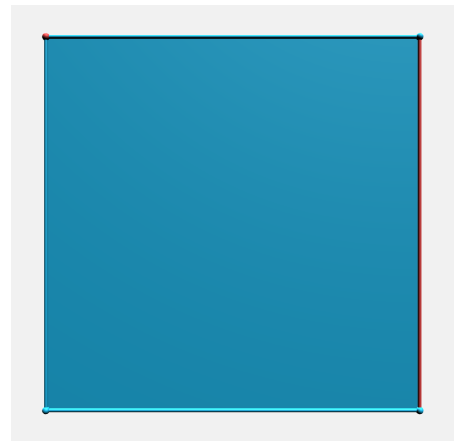


Figure 31: Vertex and edge are selected.

The faces are represented by a mesh. Two meshes, one for each side of the paper. In order to create them the vertices of the face are gone over by threes to create the triangles that cover the face. The faces are not clickable. They do not interact in any way.

Interactions

The interaction with the user is done via clicks on the screen and with the keyboard or the buttons. The keys and buttons double the same functions. The list of options is

in AppendixA the user manual. The buttons are in the interface (Figure 32) via Unity's canvas, panel, and button objects. They have been fixed in corners of the screen no matter where the camera goes.

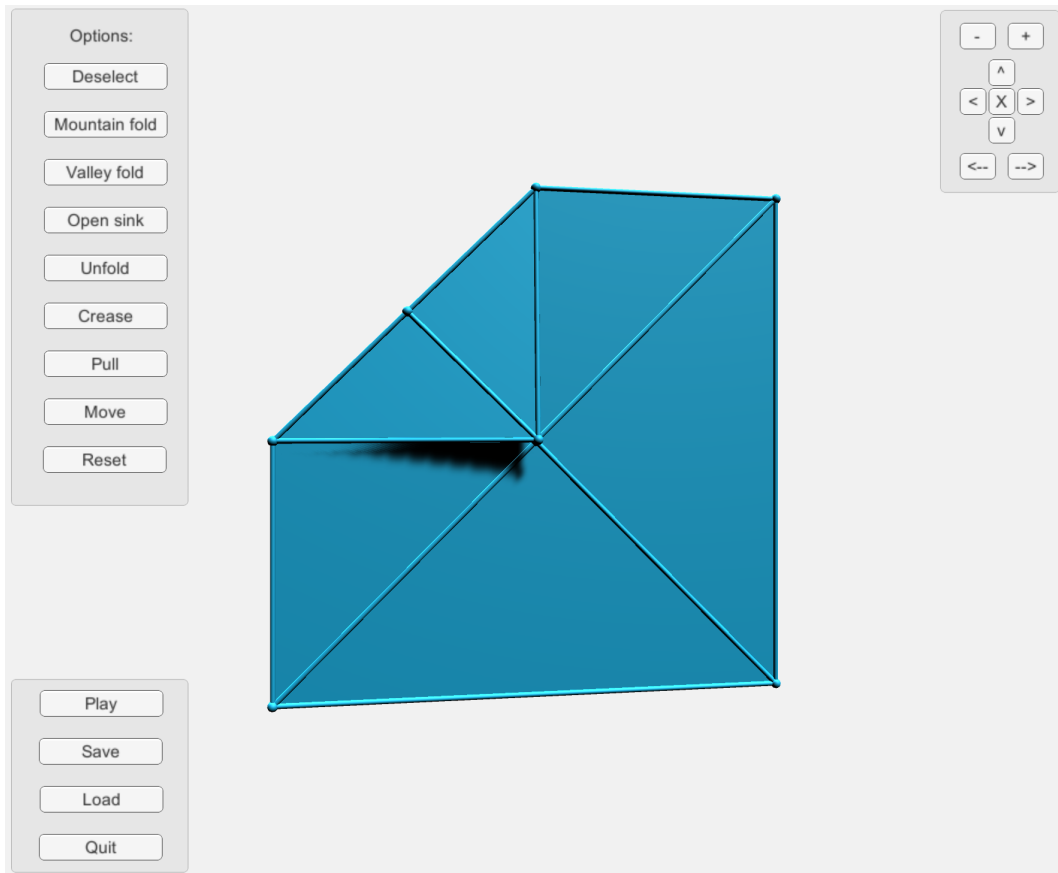


Figure 32: All buttons shown in three corners.

The camera has a script associated that lets it zoom in and out and rotate around the origin of coordinates. There is no worry in losing the model of view as neither the focus of the camera nor the model can be moved.

Scene

There are two sources of lighting in the scene: A primary one above the camera in the initial position and slightly tilted to the model and one behind the model in the initial position that is slightly dimmer.

Finally the materials, there is a material for all elements saved in the project and customized to look like it does. The background is plain white to help the user focus in the model.

6 Results and simulations

The results of this project are the following: First of all a working application that allows the user to fold simple origami figures with a clear and simple interface presented in Figure 33. Some simple figures have been folded to present the possibilities the software offers. First of all a fish folded from a kite base and then the traditional flapping bird from a bird base. The structure's data will be specified in each step along with memory consumption in the form of vertex count.

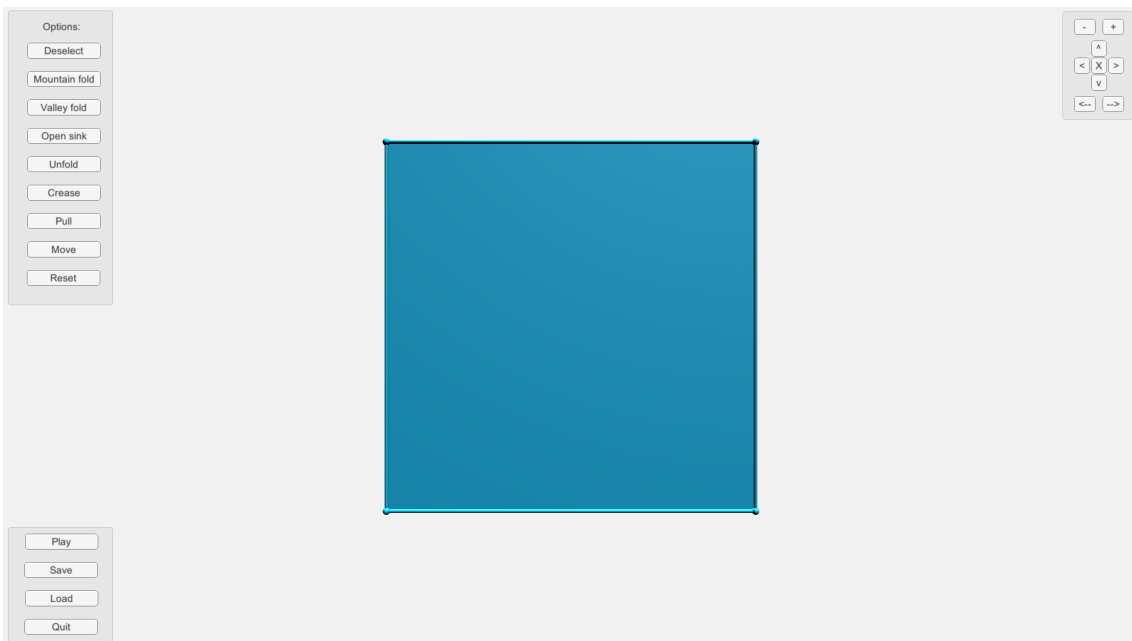


Figure 33: Finished interface with starting paper. 4 vertices, 4 edges and one face. 13.4K vertices.

6.1 Fish

This is the first figure folded with this simulator (Figure 34 to 38). It uses mountain folds, open sinks and pulls. It is only a 5 step figure but it showcases the basic functionalities of the simulator. Folding it in the simulator takes 15 seconds and making a photo diagram would take the time to screenshot and crop the photos in each step, maybe 2-3 minutes. It takes approximately 1 minute to fold it in paper and maybe 15 minutes to diagram it.

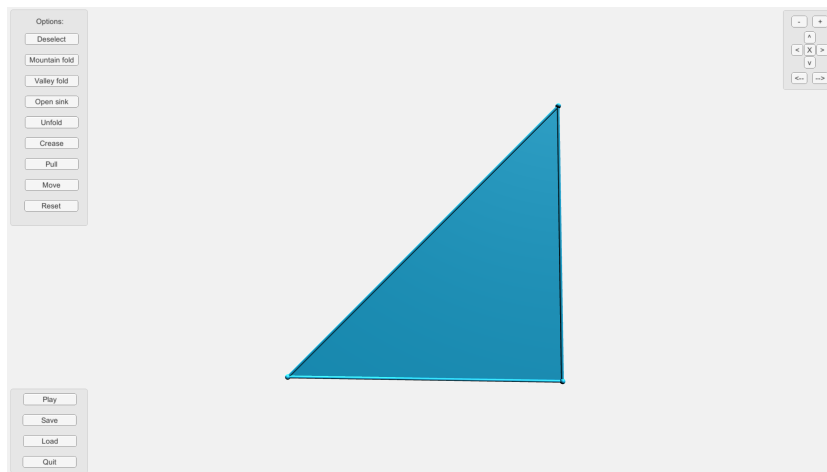


Figure 34: Mountain fold. 4 vertices, 5 edges and 2 faces. 13.9K vertices.

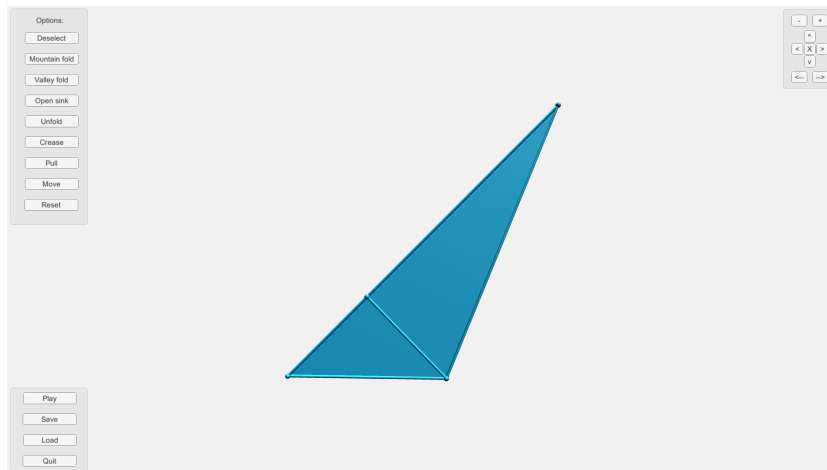


Figure 35: Open sink. 6 vertices, 9 edges and 4 faces. 20.9K vertices.

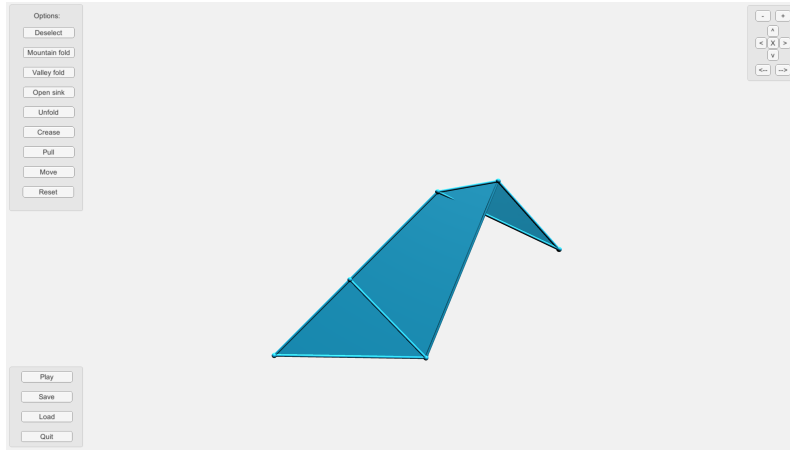


Figure 36: Pull. 11 vertices, 18 edges and 8 faces. 37.9K vertices.

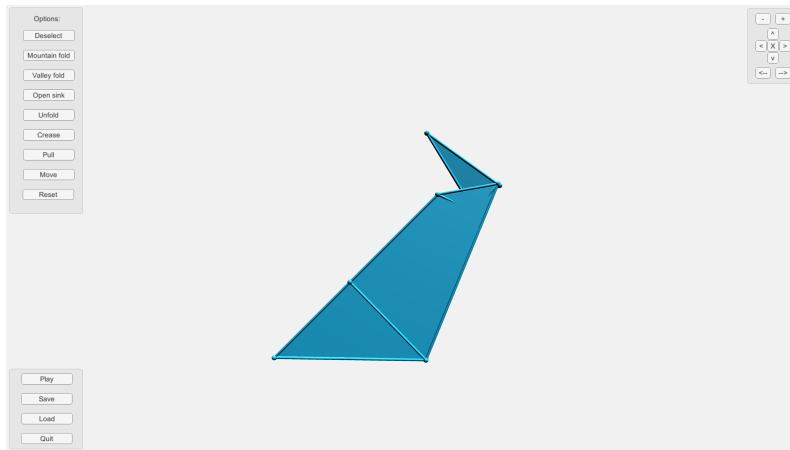


Figure 37: Second pull. 16 vertices, 27 edges and 12 faces. 13.4K vertices. 54.9K vertices.

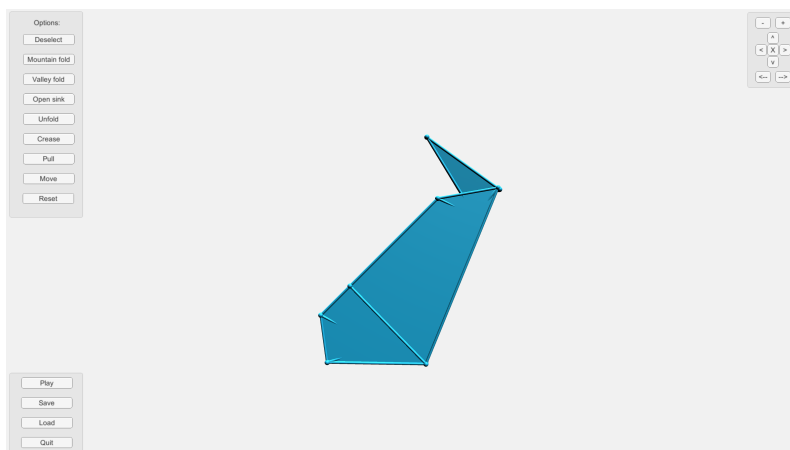


Figure 38: Finished fish. Open sink. 19 vertices, 32 edges and 14 faces . 64.9K vertices.

6.2 Flapping Bird

The flapping bird (Figure 39 to 49), though being a simple model, reaches a level of complexity superior to the fish. It uses creases and moves and has 12 steps. It shows the rest of the folds in the simulator and presents a much more iconic figure. It takes approximately 1 minute to fold it in the simulator and making a photo diagram would take the time to screenshot and crop the photos in each step, maybe 3-4 minutes. It takes approximately 3 minutes to fold it in paper and maybe 30 minutes to diagram it.

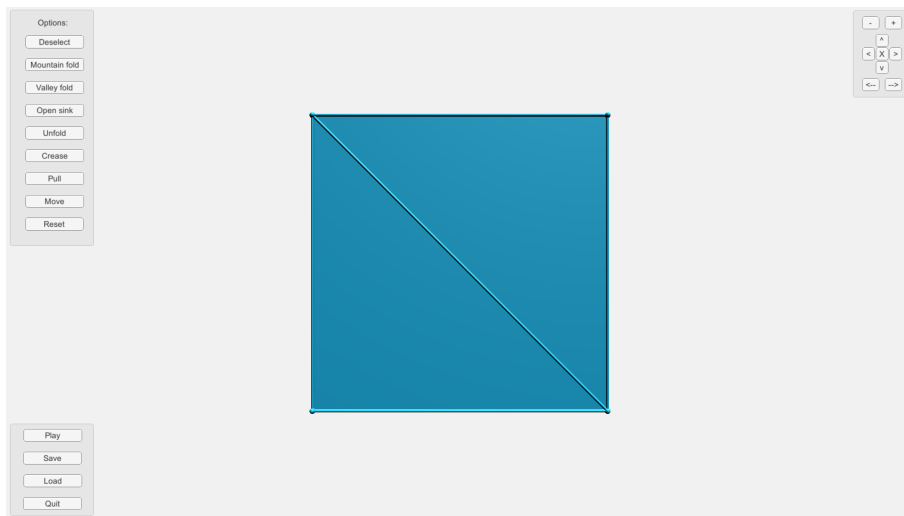


Figure 39: Crease. 4 vertices, 5 edges and 2 faces . 13.9K vertices.

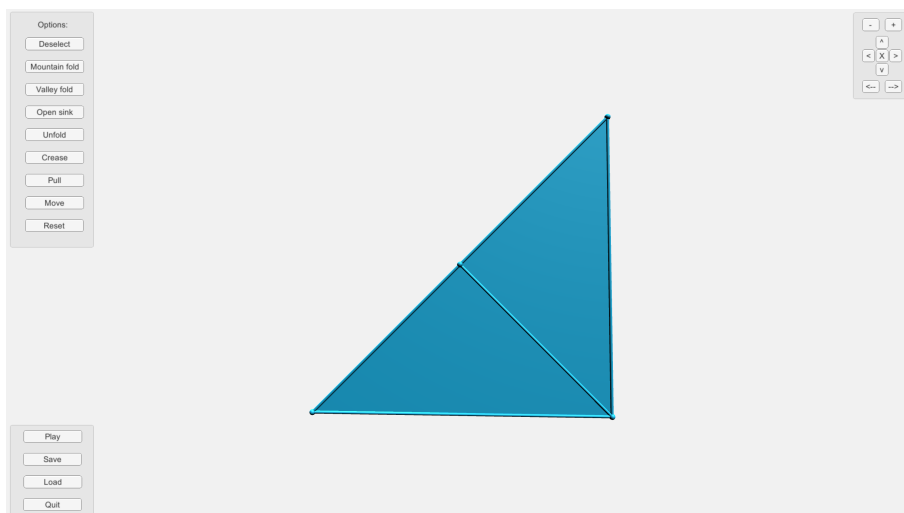


Figure 40: Mountain fold. 5 vertices, 8 edges and 4 faces . 17.9K vertices.

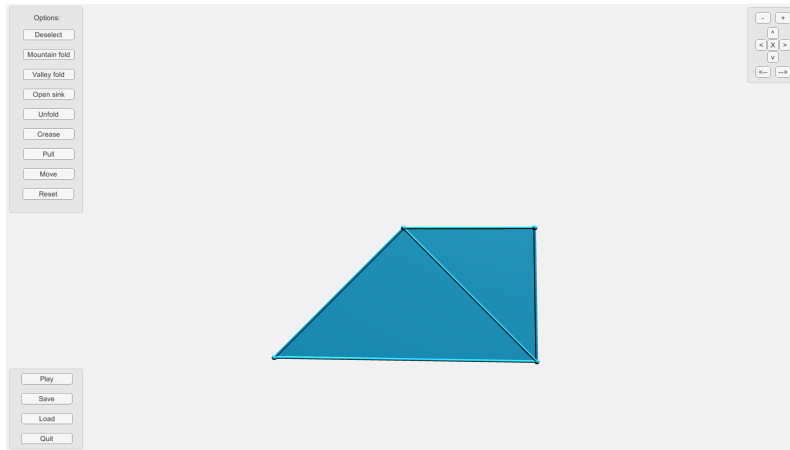


Figure 41: Open sink. 7 vertices, 12 edges and 6 faces . 24.8K vertices.

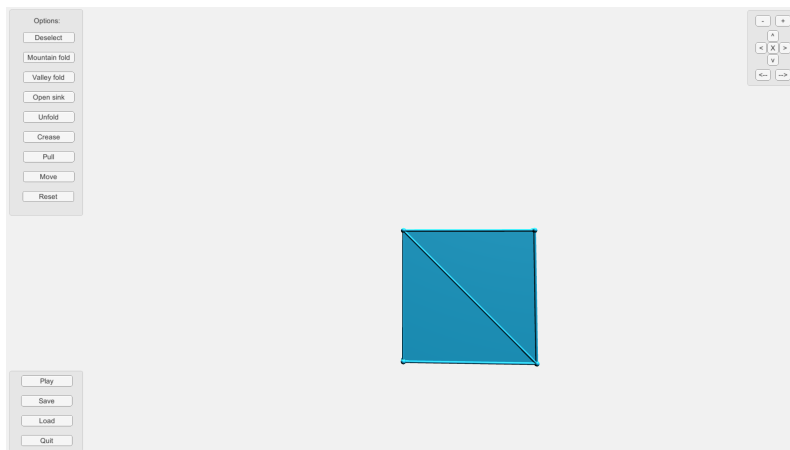


Figure 42: Open sink. 9 vertices, 16 edges and 8 faces . 31.8K vertices.

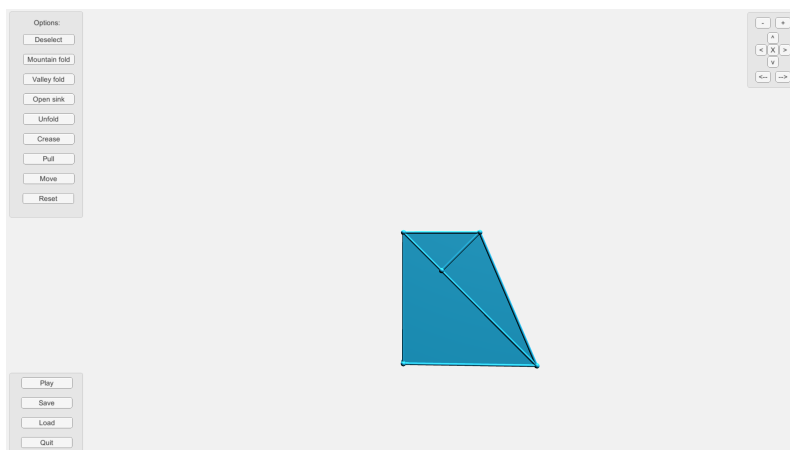


Figure 43: Open sink. 11 vertices, 22 edges and 12 faces . 39.7K vertices.

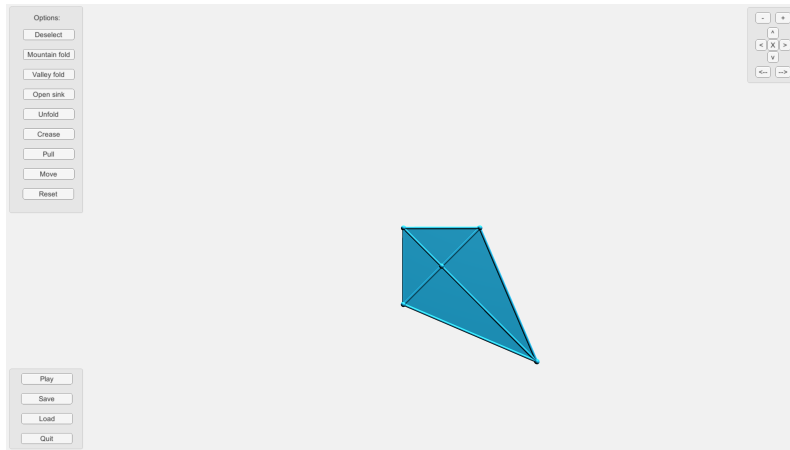


Figure 44: Open sink. 13 vertices, 28 edges and 16 faces . 64.9K vertices.

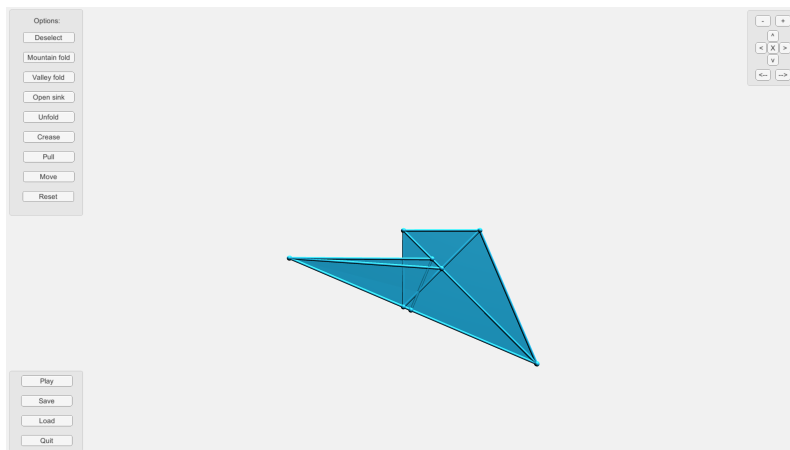


Figure 45: Pull. 16 vertices, 33 edges and 18 faces . 57.6K vertices.

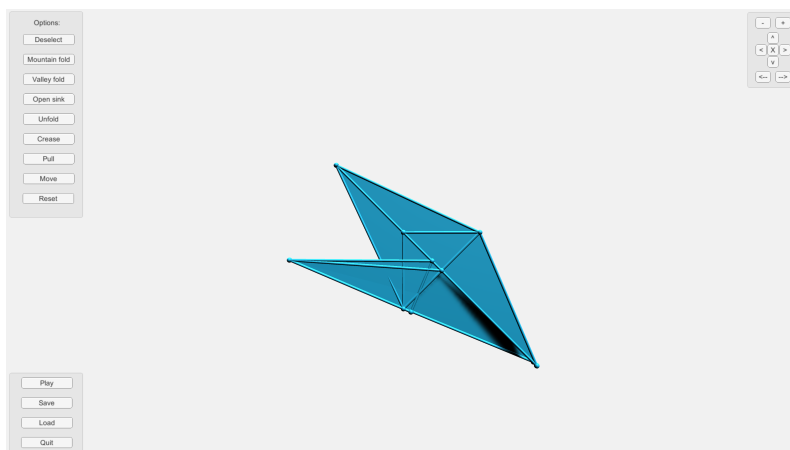


Figure 46: Move. 16 vertices, 33 edges and 18 faces . 57.6K vertices.

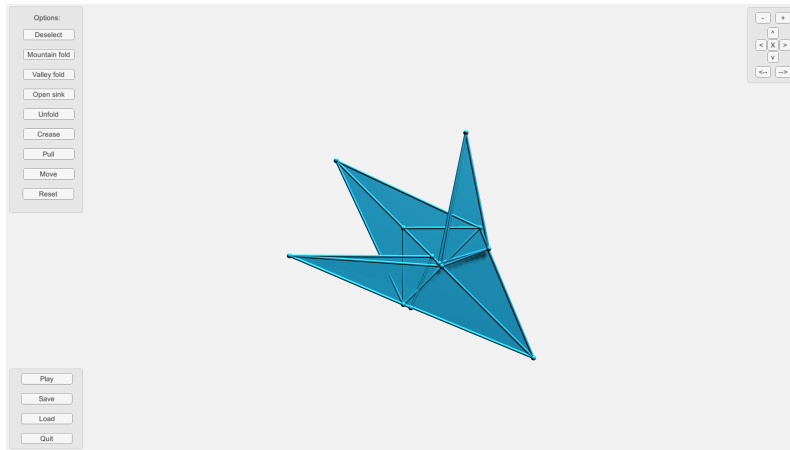


Figure 47: Pull. 19 vertices, 38 edges and 20 faces . 67.6K vertices.

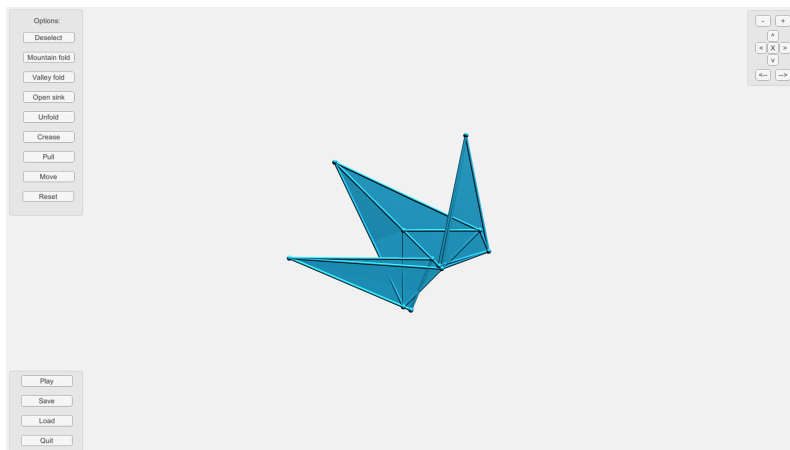


Figure 48: Move. 19 vertices, 38 edges and 20 faces . 67.6K vertices.

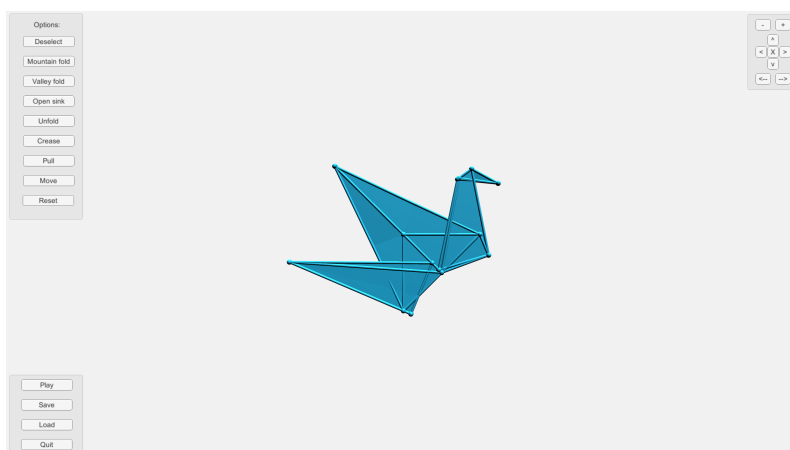
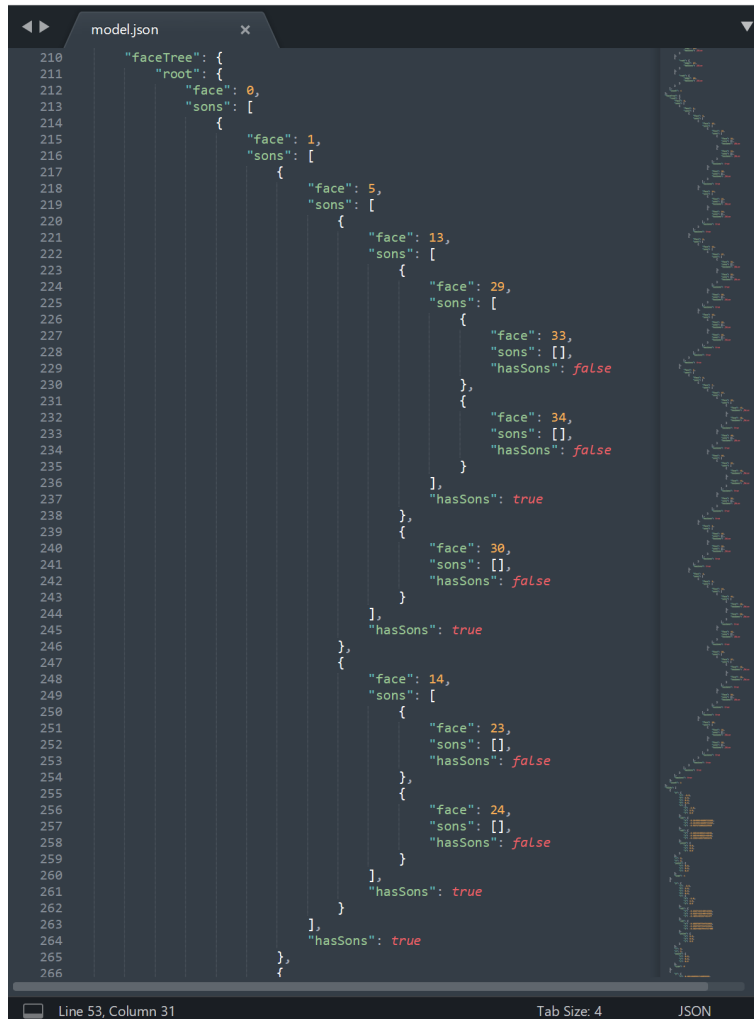


Figure 49: Finished flapping bird. Pull. 24 vertices, 45 edges and 22 faces . 83.4K vertices.

6.3 Features

6.3.1 File saving and loading

Having used the json format to save our models it means it is very easy to read, it can be opened with any text or code editor and read as is. It is a very important tool to be able to share models as it can be sent and opened in another simulator by another user. To exemplify this in Figure 50 that is approximately 3000 lines long and weighs only 88 KB we have a fragment of the flapping bird file.



```
210 "faceTree": {
211   "root": {
212     "face": 0,
213     "sons": [
214       {
215         "face": 1,
216         "sons": [
217           {
218             "face": 5,
219             "sons": [
220               {
221                 "face": 13,
222                 "sons": [
223                   {
224                     "face": 29,
225                     "sons": [
226                       {
227                         "face": 33,
228                         "sons": [],
229                         "hasSons": false
230                       },
231                       {
232                         "face": 34,
233                         "sons": [],
234                         "hasSons": false
235                       }
236                     ],
237                     "hasSons": true
238                   },
239                   {
240                     "face": 30,
241                     "sons": [],
242                     "hasSons": false
243                   }
244                 ],
245                 "hasSons": true
246               },
247               {
248                 "face": 14,
249                 "sons": [
250                   {
251                     "face": 23,
252                     "sons": [],
253                     "hasSons": false
254                   },
255                   {
256                     "face": 24,
257                     "sons": [],
258                     "hasSons": false
259                   }
260                 ],
261                 "hasSons": true
262               }
263             ],
264             "hasSons": true
265           }
266         ]
267       }
268     ]
269   }
270 }
```

Figure 50: Fragment of the json file of the finished flapping bird.

6.3.2 Play button

An added feature is that the models can be seen folded step by step in a video-like motion. It helps the user understand what is happening and understand the under-laying structure of the model much better. In the following link (Figure 51), there is a recording of this feature being used.

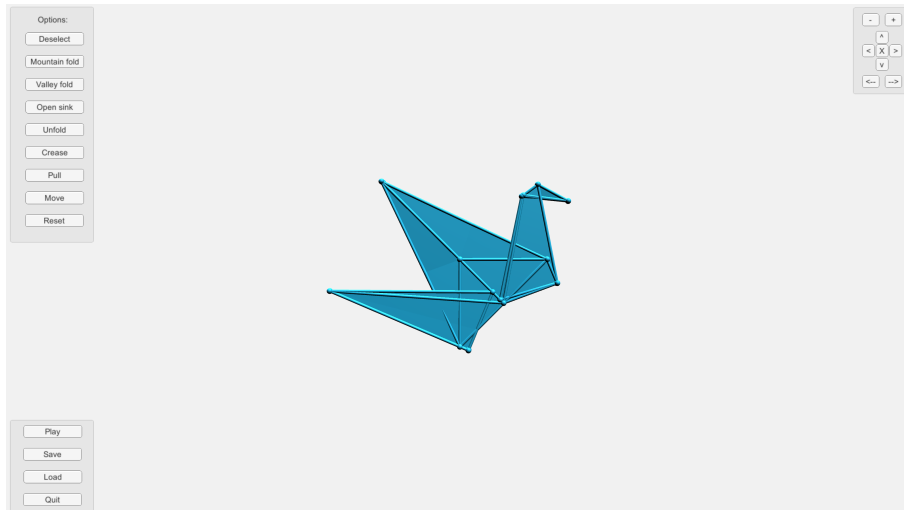


Figure 51: Link to the video: [\[6\]](#)

6.3.3 Deployment

Finally having implemented this application in a free version of Unity it only allows us to build the Windows application. However, it does not need any installation and weighs only 25MB so it is very easy to share and use right away.

7 Conclusions and future work

At the beginning of this project we set an informal objective with my tutor: "if at the end we can fold a flapping bird we will call it more than success". Why a flapping bird? Well, it is a relatively simple model, very recognizable and most people have to fold it at some point in their lives. Nonetheless, it has complicated and diverse folds and it poses a real challenge. Here we are:

In conclusion, we have achieved an application that checks all the objectives that were set. The program is capable of performing all the folds proposed and the move and crease. It allows for a realistic fold experience with a non-ideal paper. It can fold simple figures such as the fish or the more complex flapping bird. It has a clean interface with menus, a camera that rotates around the model zooms in and out and represents the model in a clear fashion. The user can interact with the model and fold as he or she wishes. The model can be saved, loaded and played so the user can see the whole process. The application is quite portable as it weighs very little and can be used in any Windows device.

It is very fast to use as it is way faster than folding paper and the diagramming process (as of now photo diagramming) is made much faster with this tool. In future iterations of the software the diagram would be automatically generated as it is stated in the introduction.

In future work it would be interesting to test the application with users so the interface and interaction can be improved and the rigidity of the system can be tested.

In development it would be interesting to improve the unfolding algorithm to let the user unfold several times instead of just one. It would also be nice to have a two-colored paper or textures and find a way to hide an object if it is intersecting with a face to get a more polished model view.

A Technical manual

A.1 Installation

This software does not need installation. The .exe file has to be run in the same folder as the other materials in the compressed file. However, the computer has to meet the requirements stated in section 2.3.

A.2 How to develop

In order to develop this application Unity's versions 2019.2.0f1 through 2019.2.11f1 have been used in Windows10 for 64 bits. Visual Studio 2019 has been used to manage the C# project. To be able to use it you have to meet the following requirements:

System	Minimum requirements
Operating system	Windows: 7 SP1+, 8, 10, 64-bit versions only macOS: 10.12+ Linux: Fixed at: Ubuntu 16.04, 18.04 and CentOS 7 Server versions of Windows and OS X are untested.
CPU	SSE2 instruction set support.
GPU	Graphics card with DX10 (shader model 4.0) capabilities.
Devices	
<u>iOS</u>	Mac computer running minimum macOS 10.12.6 and Xcode 9.4 or higher.
Android	Android SDK and Java Development Kit (JDK). IL2CPP scripting backend requires Android NDK.
<u>Universal Windows Platform</u>	Windows 10 (64-bit), Visual Studio 2015 with C++ Tools component or later and Windows 10 SDK.

Figure 52: Development requirements[12]

To continue developing the project has to be opened in Unity and the code edited with some compatible IDE. The source code is divided into 4 folders: model, view, and controller containing the classes in the class diagram in the design section. In the resources folder, there are the materials assigned to the paper. Then we have the camera script and the `starterObject` that creates the controller and hence everything else and lastly the Scene file which contains all the UI, lighting, and camera information.

A User manual

A.1 Interactions

To use this application there are 3 main ways to interact:

- **Mouse:** by clicking on an element in the screen (edges, vertices and any point in the space) it allows you to select it. It will go from blue or transparent to a bright red. This lets you interact with each part of the model. This interaction is done via a single left click.
- **Buttons:** on the top left part of your screen let you do 3 main things:
 - **Fold and unfold:** by clicking on any button related to a fold your fold will be executed given you have selected the required elements. The first selected element will be the one that will be moved during your fold.
 - **Control your selections and model:** The first button lets you deselect all the elements that are selected. The reset model resets the model to a square.
 - **Load,save and quit:** let you load the last model you saved, save the current model and quit the application.
- **Keyboard:** The keyboard lets you do the same things that the buttons allow much faster and easier and using the letters listed below:
 - 'M': mountain fold
 - 'W': valley fold
 - 'U': unfold
 - 'P': pull
 - 'D': move
 - 'O': open sink
 - 'B': return to base state
 - 'L': load model
 - 'S': save model
 - 'Q': quit
 - 'R': reset model
 - 'Y': play

A.2 Fold requirement:

For each fold there are some elements that need to be selected:

- Mountain, valley and open sink: 2 vertices, 2 edges, a vertex and a point
- Pull: a vertex and a point.
- Move: a vertex and a point.
- Unfold: nothing, it only unfolds once.

References

- [1] https://play.google.com/store/apps/details?id=com.mobilicos.howtomakeorigami&hl=en_US. Accessed 30/12/2019.
- [2] Roger C. Alperin. A mathematical theory of origami constructions and numbers. *New York Journal of Mathematics New York*, page New York J. Math. 6 (2000) 119–133., 2000.
- [3] Román Díaz. *Origami Essence*. Nicholas Terry, 2013.
- [4] Eric Gjerde. <http://papermosaics.co.uk/software.html>. Accessed 30/12/2019.
- [5] Vietnam Origami Group. *VOG2*. Nicholas Terry, 2014.
- [6] F Xavier Colet i Guitert. <https://youtu.be/kfX2hRX15UI>.
- [7] Robby Kraft. <https://beta.rabbitear.org/diagram/>. Accessed 30/12/2019.
- [8] Robert J. Lang. <https://langorigami.com/article/treemaker>. Accessed 30/12/2019.
- [9] Robert J. Lang. *Origami Design Secrets*. A. K. Peters Ltd., 2003.
- [10] Jo Nakashima. <https://www.youtube.com/watch?v=tY03IKW0vZo>. Accessed 30/12/2019.
- [11] Thomas C. Hull Sarah-Marie Belcastro. Modelling the folding of paper into three dimensions using affine transformations. *Linear algebra and its applications*, pages 273–282, 2001.
- [12] Unity Technologies. <https://docs.unity3d.com/Manual/system-requirements.html>. Accessed 17/01/2020.
- [13] Unity Technologies. <https://unity.com/es>. Accessed 07/01/2020.
- [14] Wikipedia. https://en.wikipedia.org/wiki/Finite-state_machine. Accessed 30/12/2019.
- [15] WEINA WU and ZHONG YOU. Modelling rigid origami with quaternions and dual quaternions. *PROCEEDINGS OF THE ROYAL SOCIETY A*, pages 2155–2174, 2010.
- [16] SHIN-YA MIYAZAKI TAKAMI YASUDA, SHIGEKI YOKOI, and JUN-ICHIRO TORIWAKI. An origami playing simulator in the virtualspace. *THE JOURNAL OF VISUALIZATION AND COMPUTER ANIMATION*, pages VOL. 7: 25–42, 1996.
- [17] Sheri Yin. *The mathematics of origami*. 2009.