EPITA

undergraduate (first year) project

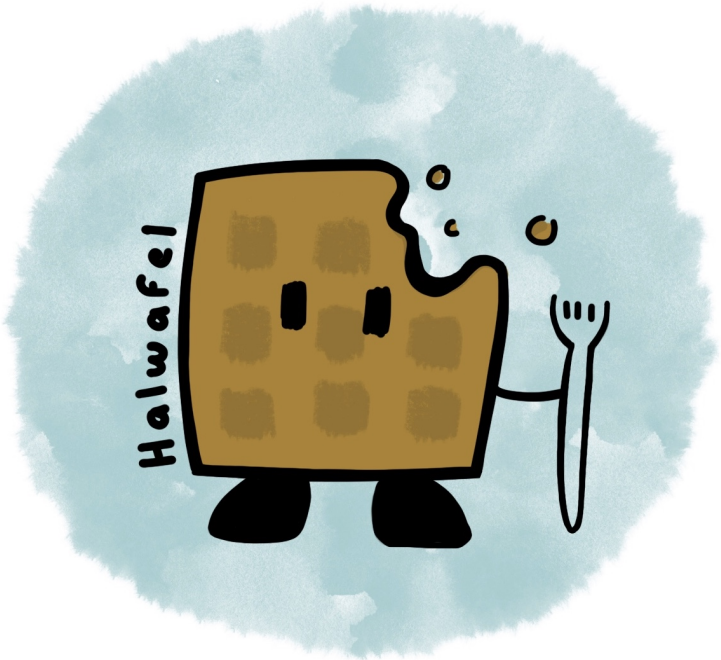
15.03.2018
Book of specifications

THE FRAME WORKSHOP

3D Animation software

Halwafel

Brian Bang, Etienne Benrey, Léane Duchet, Julie Hazan



Contents

Introduction

Halwafel is a group made of four members, Brian Bang, Etienne Benrey, Léane Duchet and Julie Hazan. It was founded on October 16th, 2017. The name of the group is based on a stylized contraction of the two words half and waffle.

Origin and type of project

In this section, the project idea will be briefly described and the process to getting to that idea will be detailed.

Object of study

This part deals with the goals and interests of the project, as a group and individually.

State of the art

This point will introduce examples of software which are close to what we want to achieve for our project.

Parts of the project

Finally, this section will present in more details what our software should do and how it is going to be divided between the members of the group and the different presentations.

1 Origin and type of project

1.1 Type of project

The Frame Workshop is an animation software, which will allow a user to make his own animations, the user will be able to import his own objects from Blender or other similar modeling programs, or use Unity's assets, since they are already handled by Unity. It should also be able to create a skeleton, that the user can then modify if necessary, to a given mesh. Moreover many tools would be provided to the user in a GUI (stands for Graphic User Interface) to manipulate every object as efficiently as possible : timelines for scenes, objects, cameras, visual effects, sound synchronisation; tools to modify skeleton at will, like adding or removing points of rotation, or even modifying the length of a link between two points of the skeleton.

1.2 Origin of the project

The first idea we had was a RTS (real time strategy) game. We then decided that we did not want to make a game especially since some of us do not play games regularly and hence lacked motivation for such a project.

The second idea was to construct a machine learning neural network. The most obvious problem that this method could solve would have been a maze. We gave up this idea after learning that this would be done in S3, as it would not be interesting to do it twice.

The third idea was making a GPS-like path finding system using A* method. This idea came up after we watched a series of computer file videos, which explain various path-finding methods. It was also interesting because it was correlated with maze solving, so went back to our previous idea. We gave up path finding because using a preexisting method meant that the project could be done too easily.

The next idea was map generation using fractals. This included the generation of mountains and forests. This main drawback to this idea was the computing power required to render these maps, we were afraid our computers would not be powerful enough.

Then, we thought we could try to do something with house automation. The idea was to process information from sensors placed in a house, use it to track problems (such as floods) and display them on a map of the house in order to be able to take action as quickly as possible. The drawbacks were that if we wanted to test our software in a real situation, we would have needed to buy sensors, which we heard was prohibited by the project rules.

We then had a spontaneous idea, which was to do a Tetris-like game. This would imply creating a 3D multiplayer Tetris in which players could affect other adjacent players' boards. We then realized that this was going back to our original game idea. Furthermore, this idea was too simplistic and had more creativity obstacles than programming obstacles.

Finally, we got to our final idea, which originated from one of our members' father's business project, which required the creation of a 3D or 2D animation video. We hence got the idea of developing a 3D animation-editing program. This seemed like a good contender for our project, since if this could be useful to one's business, it could be useful in a more general context. For example, our program could be used for trailers, marketing videos, animation films, advertisements or even for video-game cutscenes. Since we could use Unity, the 3D aspect of this project was feasible.

2 Object of study

One main aspect of the project is its nature as an editing program. Some of us having already delved into some program-based editing, most notably video-editing, we have a general idea of how to use such tools, but without ever truly understanding how they function. Creating an animation program will teach us to correctly code a complex GUI, and associate varying functionalities to the buttons we are so used to seeing work on their own. Furthermore, we will discover how to apply physics to 3D objects through scripts, so without using a handy GUI, since our aim is to make our own.

Another facet of this project is its strong association with 3D animation, which will force us to learn all about the different commonly used animation techniques. These include, but are not limited to, mesh layout and morphism, skinning, skeletal animation, or even inbetweening. It is a new subject we are discovering.

The final, and most important competence we will learn through this project is how to work in a highly organised manner, with a group of other people that will often have differing or even opposite points of view, and under the pressure of a quickly approaching deadline.

3 State of the art

The first 3D animation software was Alias1. This program was released in 1985 by Alias Research. It then became PowerAnimator in 1990.

Here are other notable 3D animation softwares :

Blender 

Blender is a free and open-source computer graphics software. It was developed by a dutch animation studio Neo Geo. Its primary author is Ton Roosendal. Its features include 3D modeling, UV unwrapping (projecting a 2D texture onto a 3D model), texturing, raster graphics editing (photo editing), rigging and skinning (skeleton animation), fluid and smoke simulation, particle simulation, soft body simulation (visually realistic physical simulation of motion), sculpting, animating, match moving (insertion of computer graphics into live action footage), camera tracking, rendering, motion graphics, video editing and compositing. It also has a game engine.

Pros :

- Open source development, free and full of features.
- Stable, as 3D programs are known to crash a lot.
- Keyboard shortcuts for anything.

Cons :

- Not industry standard, some features are hard to find or come slowly.
- Interface can be difficult to learn.

Autodesk 3DS max 

3DS max is a professional computer graphic program used in the making of 3D animation, models, games and images. It was first created under the name of 3D studio in 1988 (released in 1990) and then became 3D studio max before getting the name 3DS max. It has a built-in scripting language MAXScript (to automate repetitive tasks), a character studio which helps the user with animating, a scene explorer, a texture assignment and editing tool, a constrained animation tool which allows object to be animated along curves, skinning tool, Skeleton and inverse kinematic tools, cloth solver.

Pros :

- Great conceptual modeling tools.

- Large-scale environment creation.
- Render plugins available.
- Strong rendering capacities.
- Time-saving animation tools.

Cons :

- Does not support operating systems other than Windows.
- Tends to get unstable when handling high complexity models.

Autodesk Maya 

Maya is an application used to generate 3D assets that are used in films, advertisements, game development and architecture. It was first released in 1998. Its components include a fluid simulator, a dynamic cloth simulation, fur and hair simulation, camera tracking and a camera sequencers.

Pros :

- Handles high-end character-rigs.
- Great with tasks that involve character rigging and animation layering.
- Good motion capture handling.

Cons :

- Powerful computer needed.
- Time-consuming rendering, due to the level of detail attainable.

Houdini 

Houdini is a 3D animation application software released in 1996 which has been used in feature animation production such as Frozen, Zootopia or Rio. Its exclusive attention to procedural generation, that is to say to the automatic creation of numerical content such as models, sounds, drawings, music, dialogs, at a large scale distinguishes it from 3D computer graphics software.

Pros :

- Developed shader system.
- Viscous Fluid tools.

Cons :

- High learning curve.

- Not really user-friendly, so difficult for beginners.

MikuMikuDance 

MikuMikuDance, MMD for short, is a freeware animation program which has first been developed for the well-known, at least in Japan, virtual character Hatsune Miku. This software allows users to create their own 3D animations like those in which Hatsune Miku is staged in. This program is the most representative of what we wish to acheive with this project.

Pros:

- Modules can be added easily.
- Large community dedicated to sharing custom sets, clothing, and even different character models.
- Custom content is easily made with the right tools.

Cons:

- It is not easy nor intuitive to use.
- There is a long learning curve.

4 Parts of the project

Part of the project	**12-16 March 2018 (S1)**	**30 April - 4 May 2018 (S2)**	**6-15 June 2018 (S3)**
Mesh + Sound import	100%	100%	100%
Skeleton creation	100%	100%	100%
Graphic User Interface (GUI)	30%	80%	100%
Creation of motion	0%	55%	100%
Background imaging + sound	0%	0%	100%
Particle animation	0%	0%	100%
Website	40%	70%	100%

Part of the project	12-16 March 2018 (S1)	30 April - 4 May 2018 (S2)	6-15 June 2018 (S3)	Substitute
Mesh + Sound import	Julie Brian			Etienne Léane
Skeleton creation	Léane Etienne			Julie Brian
Graphic User Interface (GUI)	Julie	Léane Etienne		Julie Brian
Creation of motion		Julie Brian		Etienne Léane
Background imaging + sound			Etienne Léane	Julie Brian
Particle animation			Etienne Léane	Brian
Website	Brian			Etienne

4.1 Mesh and sound import

All imported models will be from Blender, or of the same format, in order to simplify the import in Unity. With regards to sound, any sound format that is easily usable with Unity will be accepted (we need to look up what these formats are, and if they are any specific complications regarding some formats and not others). The models will be "uploaded" by the user, who will be able to input a path to his model. The path needs to be able to be opened within a window, in which the user can simply navigate to the file he is looking for (or of course simply write the path). Note: This last part, in the long run, should be connected to the GUI, in order to have a coherent design. A copy will be placed in the program's files, within a folder of same name as the file (without the extension). Along with the model itself, each model folder will have a model properties file (.txt file, but with no .txt extension) IMPORTANT: Only exported 3D files will be used. Proprietary 3D application files such as .blend will not be read. All sounds will also be copied, but since they do not need to be modified, they can all be situated in one general sounds folder.

4.2 Skeleton creation

Before actually creating a skeleton, we need to find a way to analyze the model we are given, meaning to know where in the model space there are indeed material points (=meshes). This is done using the methods unity provides for the meshes.

The first step for skeleton creation is being able to place points on the complete model. All points will be based on one central point, located in the centre of the bottom-lost layer of the model. We hence need to calculate where this central point is located relative to the

Unity object point. Once this centre point is found, it needs to be written in the properties file. (If this ends up being too slow, it can also be stored in as a Unity variable for easier and quicker access. Having this point allows for the addition of new points. A point has 5 coordinates: 3 positional coordinates, and 2 rotational coordinates. All coordinates are relative to the centre point coordinates. Once all points are chosen, they also need to be stored in the model properties. A point will also, of course, be a class object. All information stored in the properties file is simply to be able to recover the information after the program is shut down.

Once points are coded, we need to code point-input. This means the user can input all 5 coordinates for the creation of a point, or decide to generate a set number of evenly spaced points on a line (so we calculate the point coordinates based on the information given). In the long run, this will have to be linked up to the GUI, so that the user can use numerical values or sliders to input point coordinates or generation. From the program's perspectives, only straight lines between two points exist, but the user can see a straight line as a curve, in which case he decides the concavity and its direction.

The user can also use a computer AI to generate all points. In this case, the user can set an average number of points to place, with a certain range of variability. The program will then find the n points for which the variation of distance to all surface points surrounding it (if this distance is greater than the ratio of the total size of the model by the number of points requested, to a given coefficient, then the distance is set to the maximum allowed distance in the calculations of the average distance). The user can change the coefficient previously mentioned in advanced settings.

Once all points are placed, all possible links are created. A link is a pointer to another point, which is created if and only if the line between the two points does not pass through the object surface (so stays inside the object). Links can be added or deleted at the user's discretion.

Once all points are connected, the point hierarchy will be implemented. The user can hence place, point by point or by selecting a group of points by 'click and drag', each point in a point tree that hierarchically ranks certain points (previously placed by the user). If a point is not placed in a tree, it is considered independent. In order to implement these trees, the point class will have a variable for its parent, and an array for its children. If the point is the root of the tree or is a leaf, the respective variable will be set to null. The hierarchy is put in place for more flexibility when creating motion.

Finally, any point can be set as a fixed point.

4.3 Creation of motion

A point can be selected and given a specific motion. These can either be basic, predefined motions, such as a straight line, a circle, an ellipse, a rectangle, or can be designed by the user. A motion has a start time (which will be modifiable with a numerical value, or through the timeline GUI by dragging and dropping), and a duration, which allows for the calculation of the speed of the point. If a point is the parent of other points, those points will move without any changes to their relative positions to the point in motion. If a

point is linked to the point in motion and is not a parent of the latter, then this point will move according to the movement of its connected point to avoid increasing or decreasing the distance between the two points. If the point cannot follow the motion without changing the length of a link, then the object will be deformed. The deformation will be greatest for links closest to the point in motion. If the links lead to a fixed point, then there will be no deformations of links that are past that fixed point.

For any movement of the rotation of a point, the closest mesh elements will be modified to fit the new shape. For each point, the user will be able to change the field of influence it has over the mesh.

By the second oral presentation, we will have created the generic structures of the motion creation, involving all the basic motion mechanics. Any preset movements or more minute properties will be put in place for the final presentation.

4.4 Background imaging and Sound

Background imaging simply uses Unity functionalities to generate plain terrain and basic background objects. This is done through Unity script, and is associated with GUI windows. Sound can be either associated with a specific motion in the timeline, which means the sound file is run at the same time (with or without a positive or negative delay) as that animation. If not, it can be run independently at any time. In both cases, this is done in the GUI timelines.

4.5 Particle animations

Particles are considered as a type of objects and are useful to represent smoke, fluids or other projections. Each particle is an independent image or object emitted in a great number to create the illusion of a moving amorphous entity. Unity's system of particles will be used and should allow the user to modify some parameters.

The first group of parameters affects the general behavior of the particles. Each particle has a lifetime, from when it is emitted to the moment it disappears. During this lifetime, the particle can undergo several changes. The system emits particles in random directions in a sphere-shaped area. Another parameter is the emission rate which is the number of particles produced per seconds - the exact time of emission is however slightly randomized. Other parameters are specific to each particle and affect their state over time. The velocity vector determines the distance and direction of the motion of a particle. It can be affected by gravity or other forces such as wind zones. The color, size and rotation of a particle can also be modified over time.

Fluids are a specific application of the particles system. A waterfall, for instance, can be synthesized with a low emission rate and gravity affecting the velocity vector.

4.6 GUI

Main window

The main window is generated almost blank. It has a menu bar, and a tools bar. The menu bar has the following tabs: File, Edit, View, Insert, Tools, Options.

The File tab has the following options: New project, Open project, Save, Save as, Render, Render as, New project element.

The View tab has all the different possible windows listed. If they are checked, they are displayed, if they are not checked, they are not displayed. It also has an option for auto-arranging windows, and the option to show or not basic grid lines on the preview/main visualization window (these include center lines and golden ratio lines).

The Insert tab has the following options: objects (basic shapes), background shapes, background images, music. The Tools tab has all the basic tools that are found in the various windows, such as inserting points, or a tool to link various elements in the timelines together.

The Options tab will have any extra options we are able to implement as we go through the project.

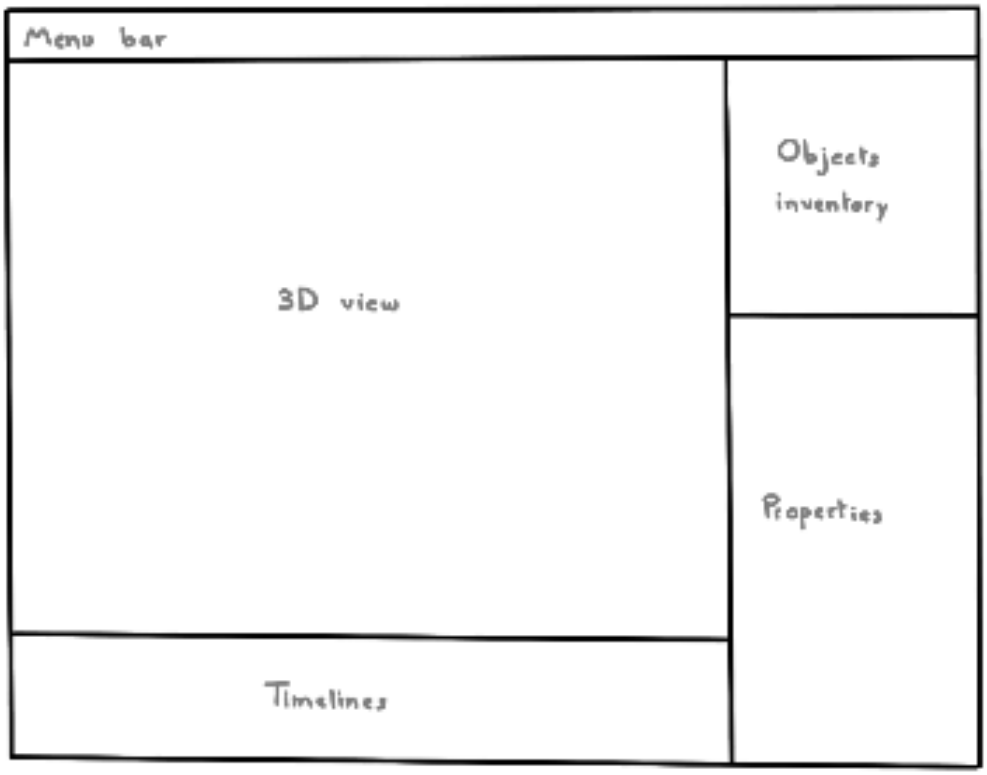
Internal window class

Anything not mentioned above is a window. This means that it can be resized, or even popped out and moved as the user pleases. Each window will have the same basic structure: title of the window at the top left, an options bar at the bottom, options icons centered, and the main part in between, with minimal margins on each side. At the far right of the options bar will be an option to 'pop out' the windows, to freely move it around the screen.

It is important to note that all purely graphic specifications, most notably button arrangements, may change if we deem a different layout to be be more optimal, once we have a graphics prototype, and can hence see all the objects mixed together.

The essential variables that are general to all windows of the class are the size (both height and width), a boolean determining if it is inlayed or popped out.

Furthermore, a generic organization will be put in place, with all the basic items of the class present. This will be modifiable by the user at any time. This general structure is as follows:



The first section of the GUI structure will be partially implemented for the first presentation. The first two sections will be completely implemented for the second one. All windows, which will be described in the next part, will be implemented as their corresponding functionalities are developed.

List of internal windows

All elements of the GUI will be organized into windows. These windows can either be organized in a plane or can be separated and overlaid.

The **3D view** window is the most important window. It is the window in which the user is able to see and navigate around his scene. He can also move objects through a click-and-drag system, and open an object's properties by clicking on it. Depending on the object type, the properties window will be replaced by a new one, that is specific to that object's type. For example, clicking on a point will open the Points window, and select that specific point. Objects can be selected, moved and modified directly from this window. Shortcuts will allow the user to access the left, right, up and down views as well as the view from the camera. Another way to navigate around the scene is by using Unity's commands.

The **Object** inventory draws up a list of the objects present in the scene. By default, the object inventory contains a lamp and a camera. The user can add objects through this window, which will be placed at a default position until moved by the user.

The **Files** is the window in which the user can access all the models or sounds that he has given the program access to. He can also add new models or sounds to the project inventory through this window (opening a new window to access the file through the options

tab).

The **Timeline bar** displays all the timelines used during the animation. There are three types of timelines.

The Visual timeline manages the visual effects, which include movements, objects appearing and disappearing, particles.

Then, there is the Sound timeline which can be divided into two categories: one for background sounds that can be used for music, voiceover, noises and the other one for sound effects. For example, the sound of the collision between two objects or the voice of a character will be featured in the second timeline. These noises can be linked to movements which means that if an element from the visual timeline is moved, the sound effect moves at the same time.

Then, we have a Camera timeline which is responsible for all camera movements (rotation and position), including overlapping ones and which camera is active.

Finally, the Scenes timeline (divided in parts corresponding to each scene of the animation) is more general timeline which encompasses all the others. The user can add or delete any of these four types of timelines (does not include scenes timeline).

The last window is the **Properties** window. As explained earlier, this window displays the properties of all the different types of objects in the scene. These will all have position, but will also have the all the specific properties of the studied object.

All other windows, including but not limited to point hierarchy, background management and particle management, will be openable through the View tab mentioned earlier.

4.7 Website

The main structure of the website will be done by the first presentation. This will be done using a template, which will help us quickly apply all the wanted graphics, as well as the general structure of the page. It will be filled out as we go with all the necessary information about the project. This will go on until the project is finalized.

Conclusion

To conclude, we did not want to make a game, so we made an ambitious choice and decided to do a 3D animation software. Clarifying our ideas helps a lot in understanding how we will proceed to achieve our goal. We know obstacles will be on our path, but this will be rewarding, as we are going to learn a lot from this experience.