Projet de spécialité : Gamma 3 en 3D

Jonathan Julou

Encadrants: Alain Guyot, Roland Groz

10 Juin 2020

Table des matières

I. Remerciements

Je tiens à remercier Roland Groz et Alain Guyot pour m'avoir proposé ce sujet et conseillé au long du projet. Je remercie aussi Alain Guyot et Philippe Denoyelle pour leurs retours précieux, ainsi que l'association Aconit pour son support en général.

II. Introduction

Ce projet d'histoire de l'informatique s'inscrit à la suite du projet réalisé par Lucas Trampal et José Maillard et accessibles par les liens ci-dessous :
https://www.aconit.org/histoire/Gamma-3/Articles/rapport%20projet%20ENSIMAG-18.pdf
https://github.com/lutrampal/bullgammator

Ils ont réalisé un émulateur web très complet pour la machine de Bull, le Gamma 3, commercialisé pendant les années 50. L'objectif de ce projet est de réaliser un modèle 3D du gamma 3, et de reprendre une partie de cet émulateur pour l'intégrer à une simulation du panneau de programmation de la machine.

Leur simulateur inclut l'extension tambour, qui permet notamment de compiler du code déca. Mais pour ce projet, je me suis concentré sur le Gamma 3 de base, donc on utilise pas cette fonctionnalité.

III. Outils utilisés

i. Blender

Blender est un logiciel libre principalement orienté vers le modélisme et l'animation, bien que l'on puisse faire beaucoup plus avec.

Dans ce projet, tous les modèles ont étés réalisés sous Blender, puis exportés au format COLLADA.

ii. Godot

Godot est un moteur de jeu vidéo libre et multi-plateformes.

Il dispose d'un langage de script fortement inspiré de Python.

J'ai essayé d'utiliser Three.js à la place, mais j'ai eu l'impression de refaire beaucoup de mécanismes qui étaient déjà dans Godot de base, comme une programmation événementielle simplifiée ou la gestion d'un HUD 2D sur une scène 3D.

iii. Node.js

Le framework qui a été utilisé par Lucas Trampal et José Maillard pour développer leur émulateur est Angular.js. Il permet de développer de belles applications javascript, avec du code proprement découpé.

Le problème, c'est que Angular.js est pensé pour des applications en une seule page Web, et je n'ai pas réussi à interfacer ce code avec du javascript plus simple, ni avec Godot, qui peut intéragir avec du javascript de base.

Une solution est de transformer leur code en API Web, donc une interface qui permet d'utiliser le protocole HTML pour gérer les communications. Pour cela j'ai utilisé Node.js. Ce framework permet énormément de chose, et est plutôt pensé pour les applications aux architectures complexe, mais on l'uilise ici simplement pour bénéficier de sa capacité à faire tourner efficacement une API.

IV. Modèles 3D

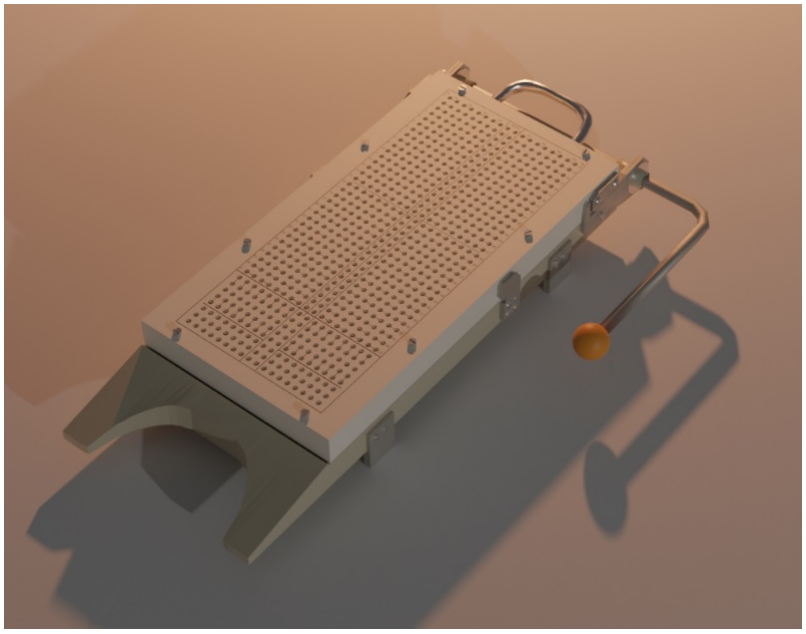
Les composants sur lesquels le plus d'attention a été portée sont le panneau de programmation et les tiroirs contenant la mémoire vive de 12 chiffres décimaux.

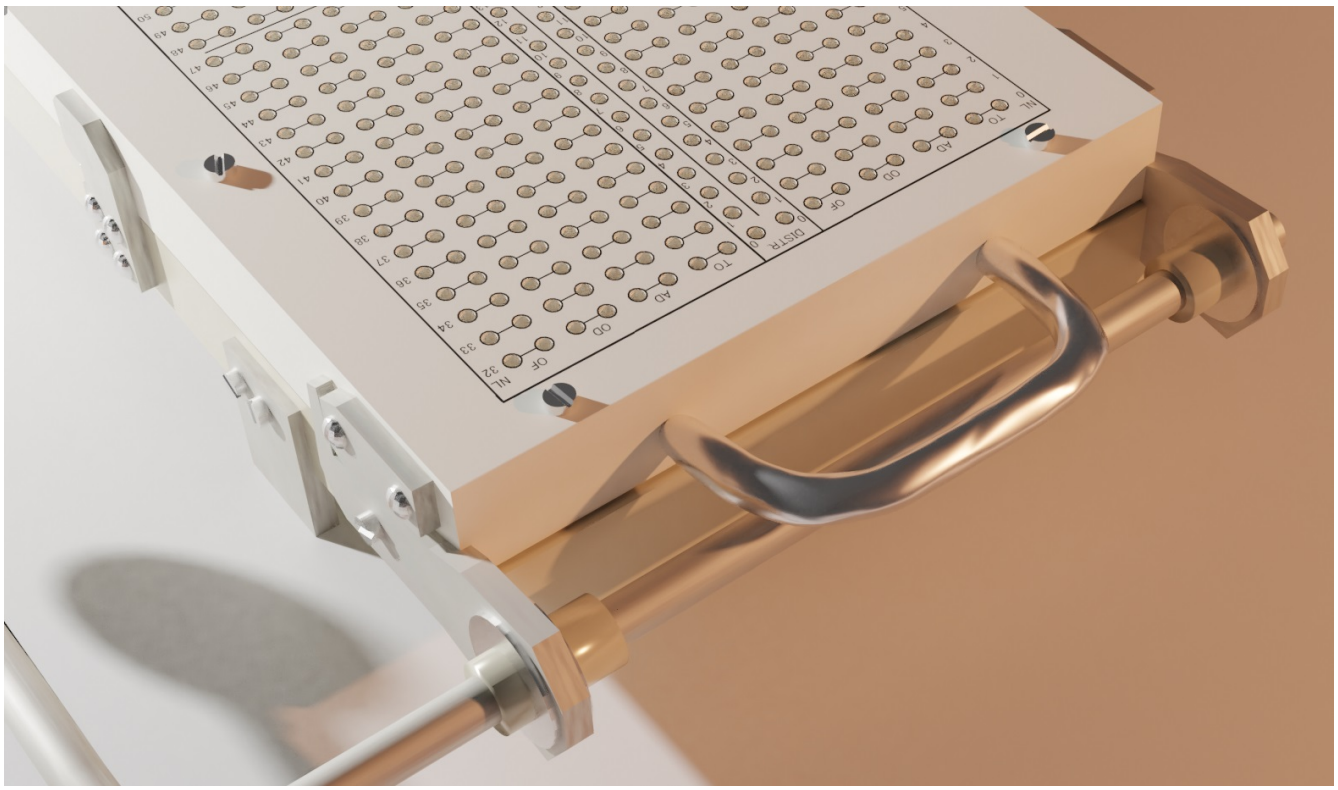
Malgré un usage de textures simples, une attention particulière à se rapprocher de la vraie structure des composants a été portée. Au final certaines libertés "artistiques" ont été prises, surtout pour unifier les variations entre les différentes photos (notamment certains tubes à vides manquant sur des tiroirs, ainsi que l'arrangement des diodes).

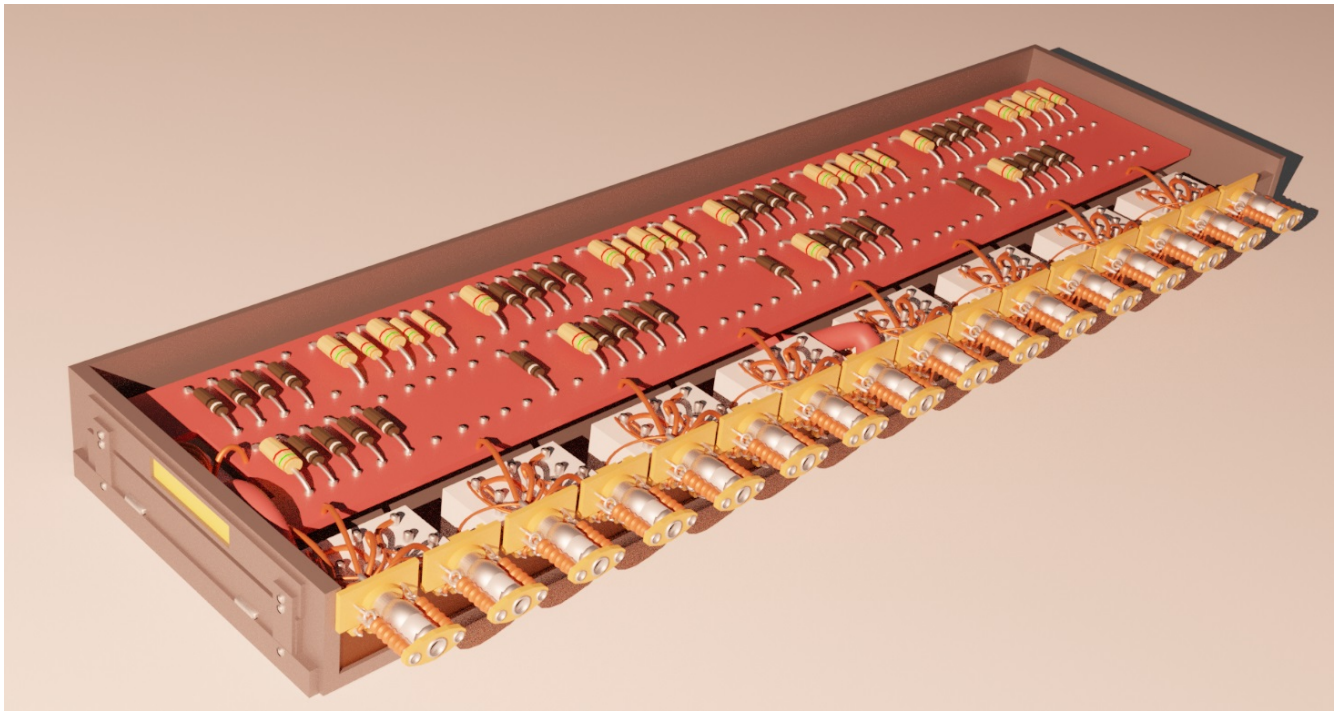
Outre l'attrait pour l'histoire de l'informatique, un de mes objectifs pour ce projet était de mieux comprendre comment utiliser Blender. Voici donc, juste pour la beauté de la chose, quelques rendus des pièces modélisées avec le moteur de ray-tracing de Blender.

On peut regarder individuellement les pièces modélisées sur
https://www.aconit.org/histoire/Gamma-3/Simulateur3D/pieces/panneau/
https://www.aconit.org/histoire/Gamma-3/Simulateur3D/pieces/rack/

Les fichiers Blender/COLLADA sont accessibles ici : [A VOIR SI CA PASSE SUR GITLAB]



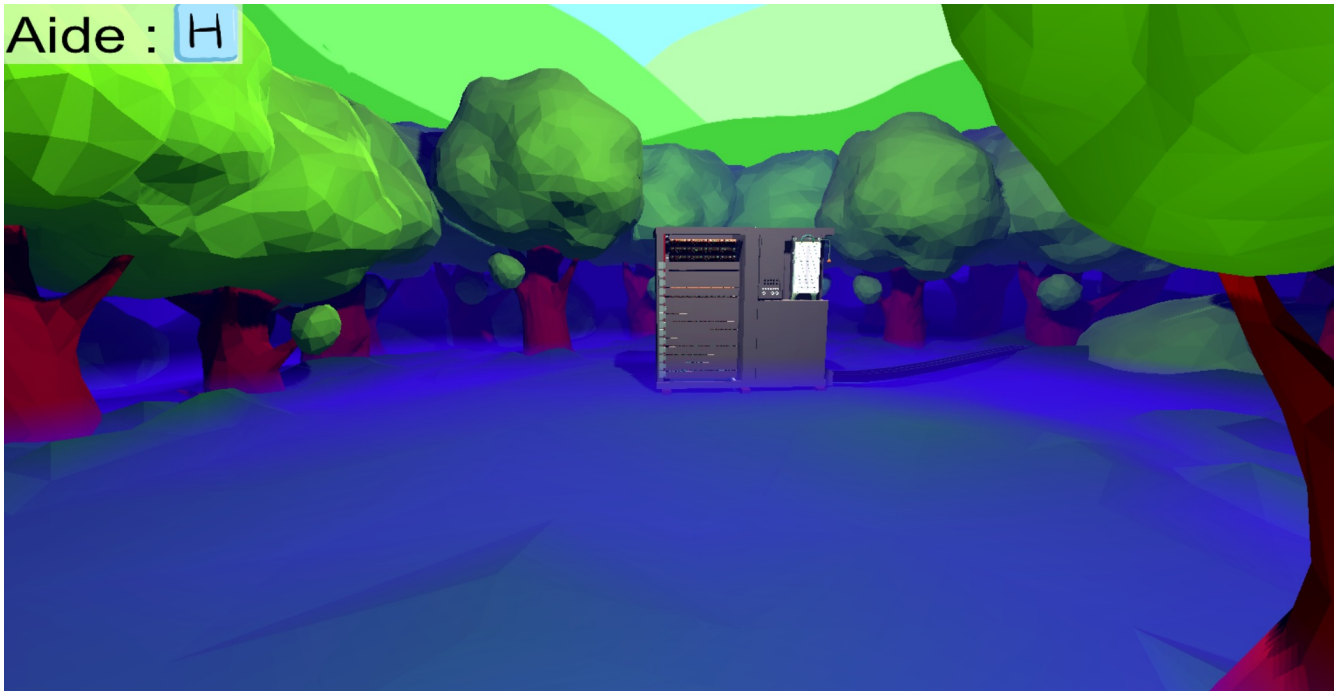




V. Simulateur Web

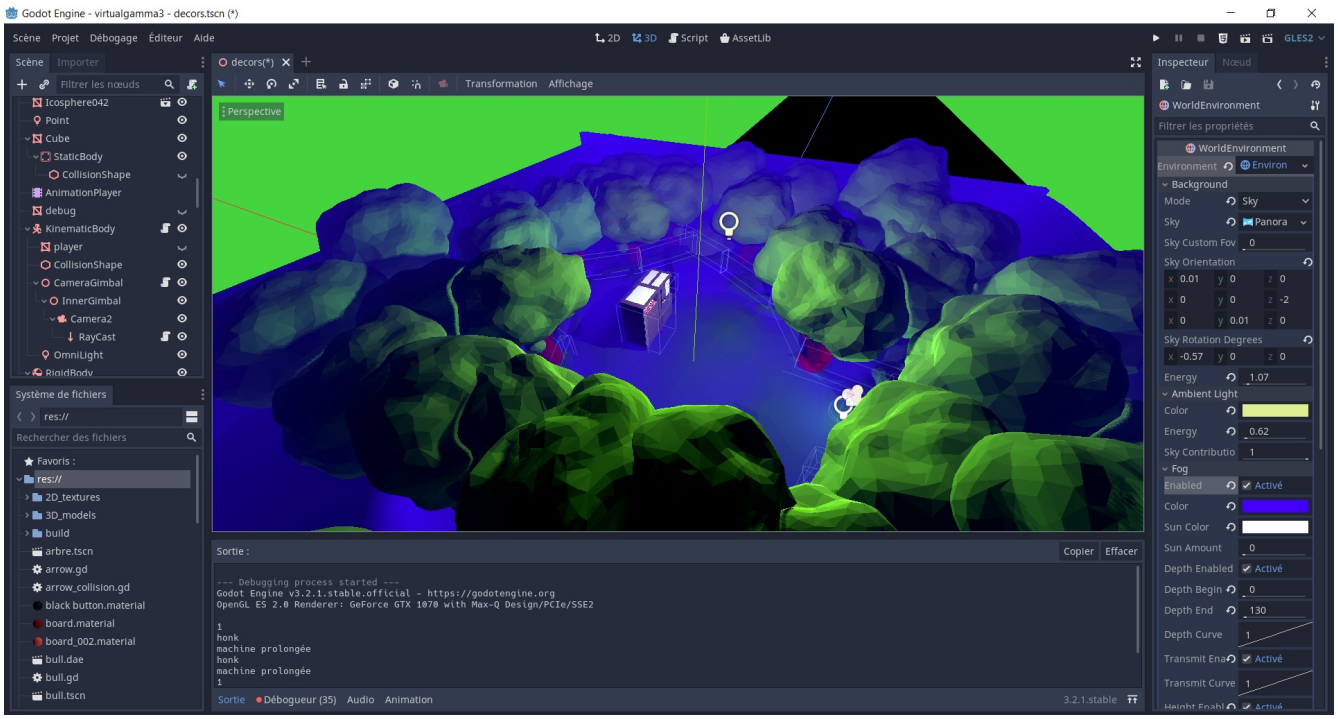
i. présentation

Voici ce que l'on voit en arrivant sur la page du simulateur :
https://www.aconit.org/histoire/Gamma-3/Simulateur3D/

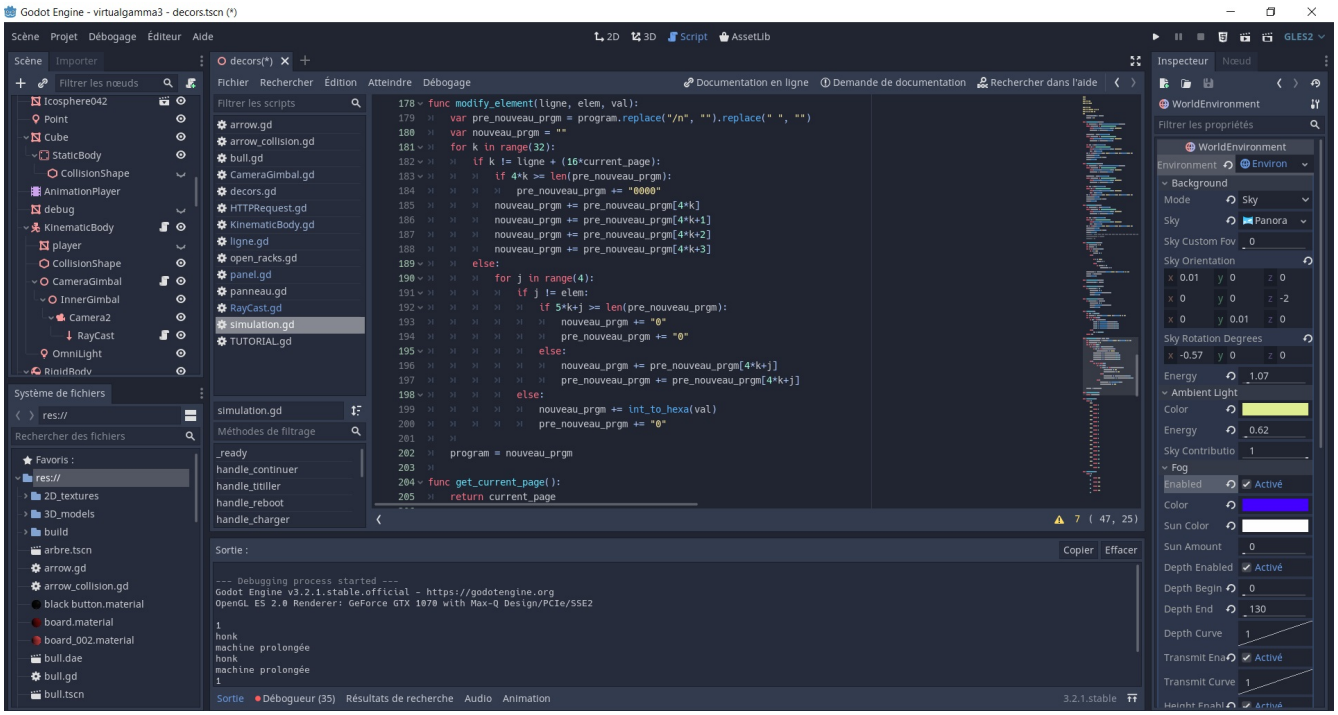


Le simulateur utilise le moteur Godot, avec le moteur de rendu 3D OpenGL ES 2. Ce dernier est dépassé, car il y a déjà OpenGL 3 et 4. Cependant ceux-ci sont instables et très lents sur beaucoup de navigateurs Web.

L'environnement de développement ressemble à ceci :



Il y a également une interface permettant de faire aisément de la programmation évenementielle en attachant des scripts à des objets :



Au final, le simulateur ressemble un peu à un jeu vidéo, où on peut se déplacer autours du Gamma 3, et ouvrir ses tiroirs de mémoire. Cependant cette partie est assez limitée. Le coeur du simulateur est une interface bi-dimentionelle de programmation, qui se veut à mi-chemin entre ce que l'on trouverait sur la vraie machine, et l'interface de l'émulateur de Trampal et Maillard.

ii. mise en place

Une fois le projet exporté, on récupère des fichiers Web Assembly et un fichier index.html. C'est ce dernier qui sert de point d'entrée au simulateur.

Le simulateur a besoin de l'API qui se trouve dans l'émulateur, il faut donc la configurer avec l'ip du serveur et un port, puis lancer avec "node api.js" pour que le simulateur fonctionne. Il faut spécifier l'ip et le port dans le projet Godot aussi.

iii. fonctionnement de l'émulation

Le programme fournissant l'API maintient une liste de machines. Ce sont des objets représentant chacun un Gamma 3, avec son panneau, sa mémoire, etc...

Quand quelqu'un accède à la page Web du simulateur, ce dernier envoie une requête à l'API pour lui créer une machine. On lui affecte donc un numéro et une machine est ajoutée à la liste.

Toutes les 42 secondes, l'API regarde si le client est encore actif. Si ce test échoue 5 fois, la machine de ce client est supprimée. Heureusement pour lui, le simulateur signale toutes les 10 secondes qu'il est encore vivant, tant que la page Web n'a pas été quittée.

Il est possible d'effectuer certaines opérations sur la machine à distance, comme modifier le panneau de connexions, récuperer l'affichage de la console, continuer l'execution, titiller... On peut aussi récuperer les programmes d'exemples fournis avec l'émulateur.

iv. manuel utilisateur

iv.1 au tout début

Il faut probablement cliquer sur la page avec la souris pour pouvoir utiliser les contrôles.

Ensuite, les commandes sont affichées en appuyant sur la touche 'H', comme indiqué en haut à droite de l'écran.

Il faudrait rapidement appuyer sur 'C', pour "capturer la souris". Cela permettra de contrôler la caméra avec. La molette permet de zoomer. Il est possible de régler la sensibilité avec les touches "P" et "M".

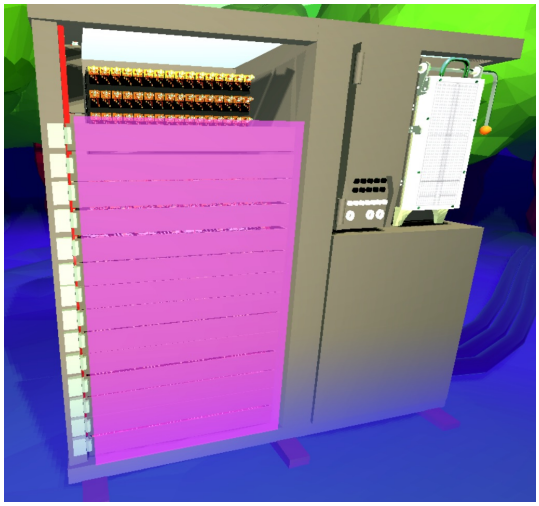
Pour récupérer sa souris, il faut appuyer sur "F", ou bien la touche "echap" peut marcher selon les navigateurs.

On peut se déplacer avec les touches ZQSD, ou bien le pavé directionnel.

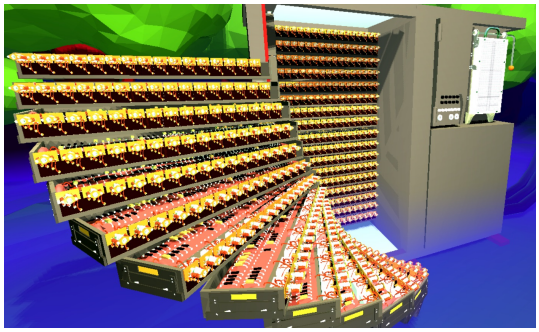
Il est possible d'intéragir avec certaines parties du Gamma 3 en utilisant le clic droit. Ce mécanisme est décrit dans la section suivante.

iv.2 intéragir avec les tiroirs

En s'approchant de ceux-ci, une surbrillance violette devrait apparaitre.



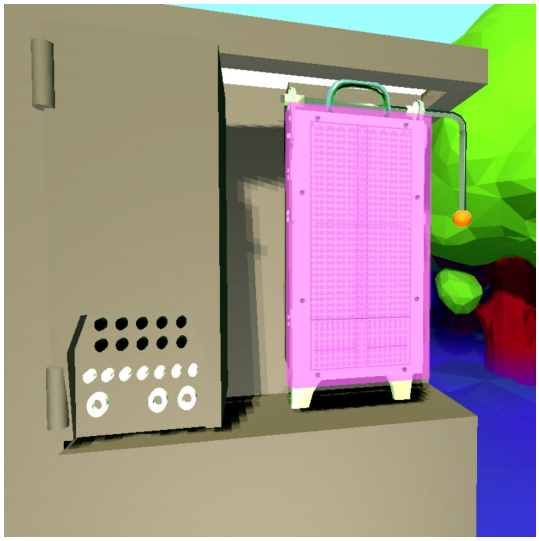
On peut alors faire un clic droit pour activer l'animation qui va les ouvrir.



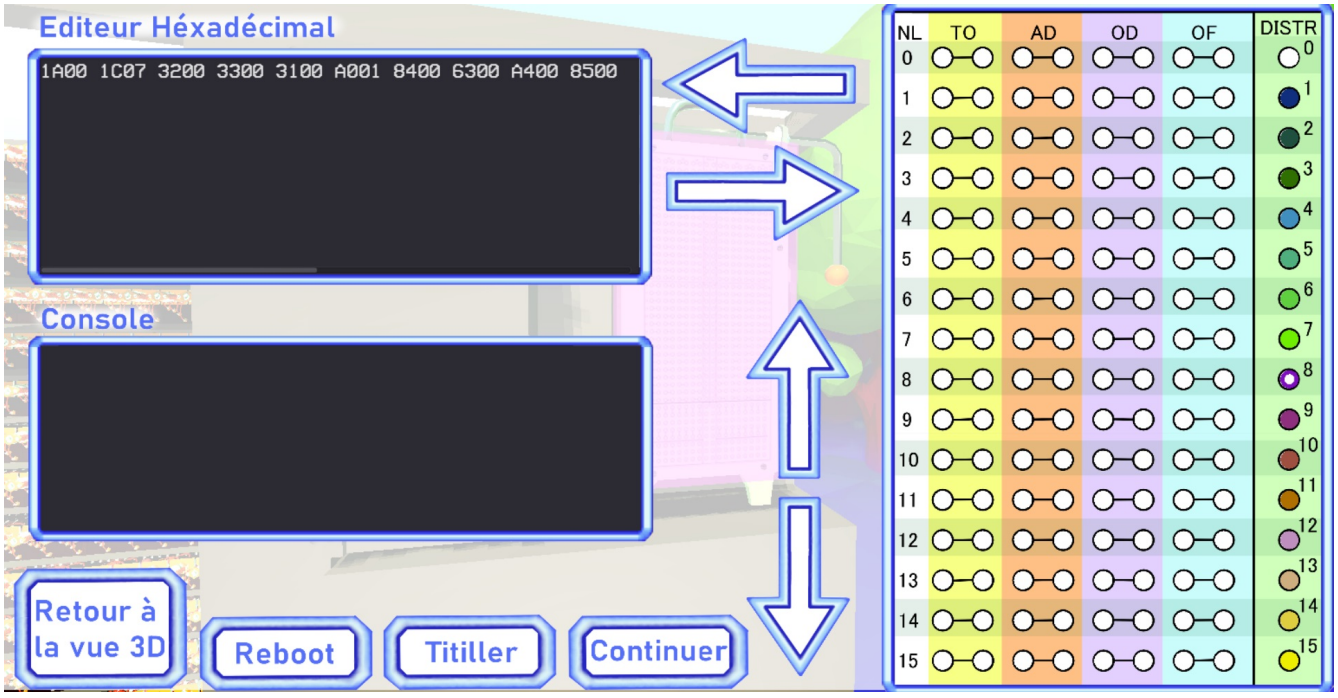
De même, on peut les refermer.

iv.3 Panneau de programmation

De même qu'avec les tiroirs, on peut s'en approcher et cliquer droit quand la surbrillance violette apparait. Un bug a été remonté qui rend l'interface inaccessible, il faut alors cliquer sur le bouton [A RAJOUTER].



On arrive dans l'interface suivante :



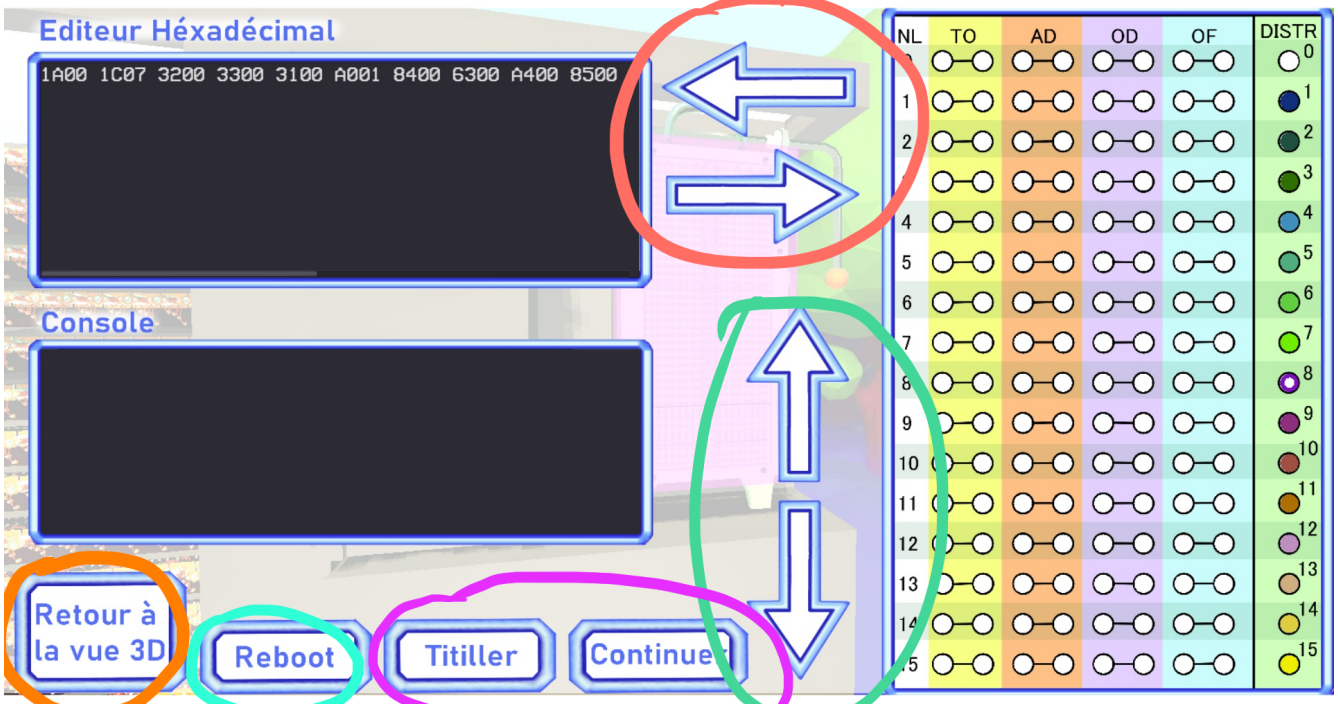
Ci dessous, les fonctions effectuées par chaque bouton :

Chargement du programme:

Panneau -> Editeur

et

Editeur -> Panneau



Quitte l'interface de programmation

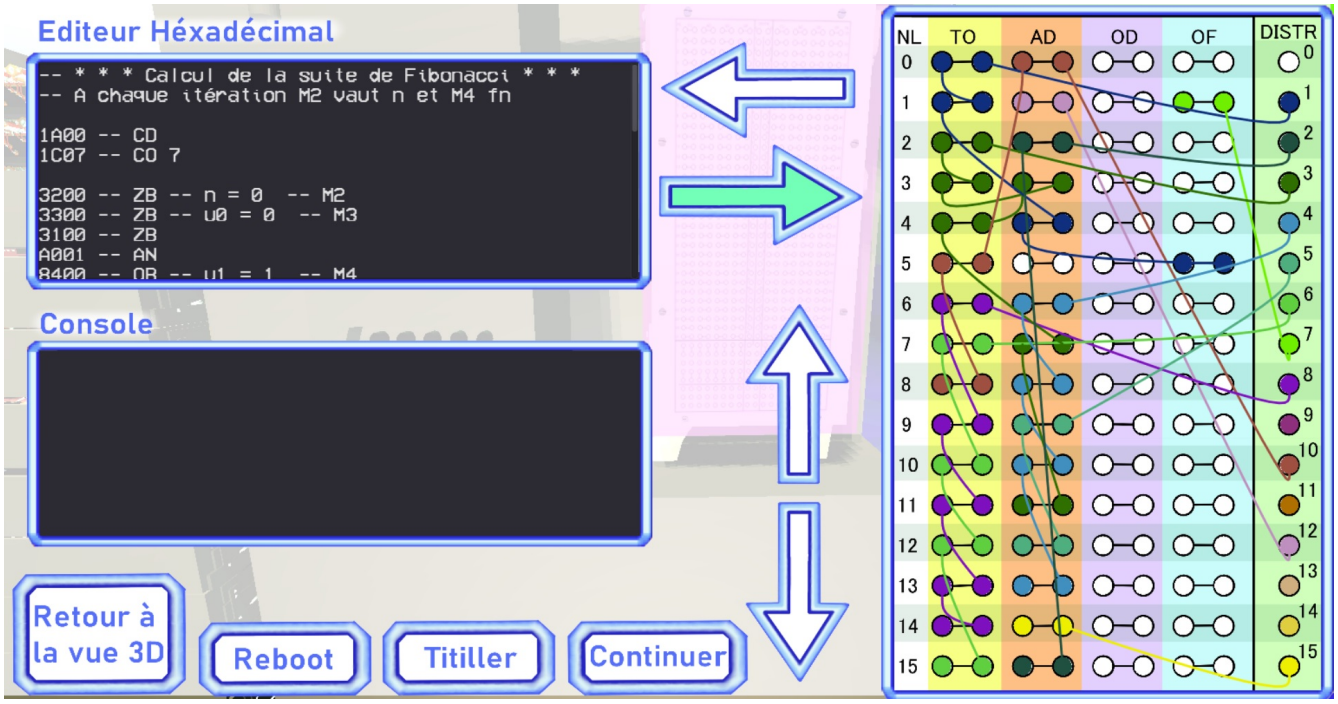
Réinitialise la machine

Fonctions similaires a celles de l'émulateur de Trampal/Maillard.

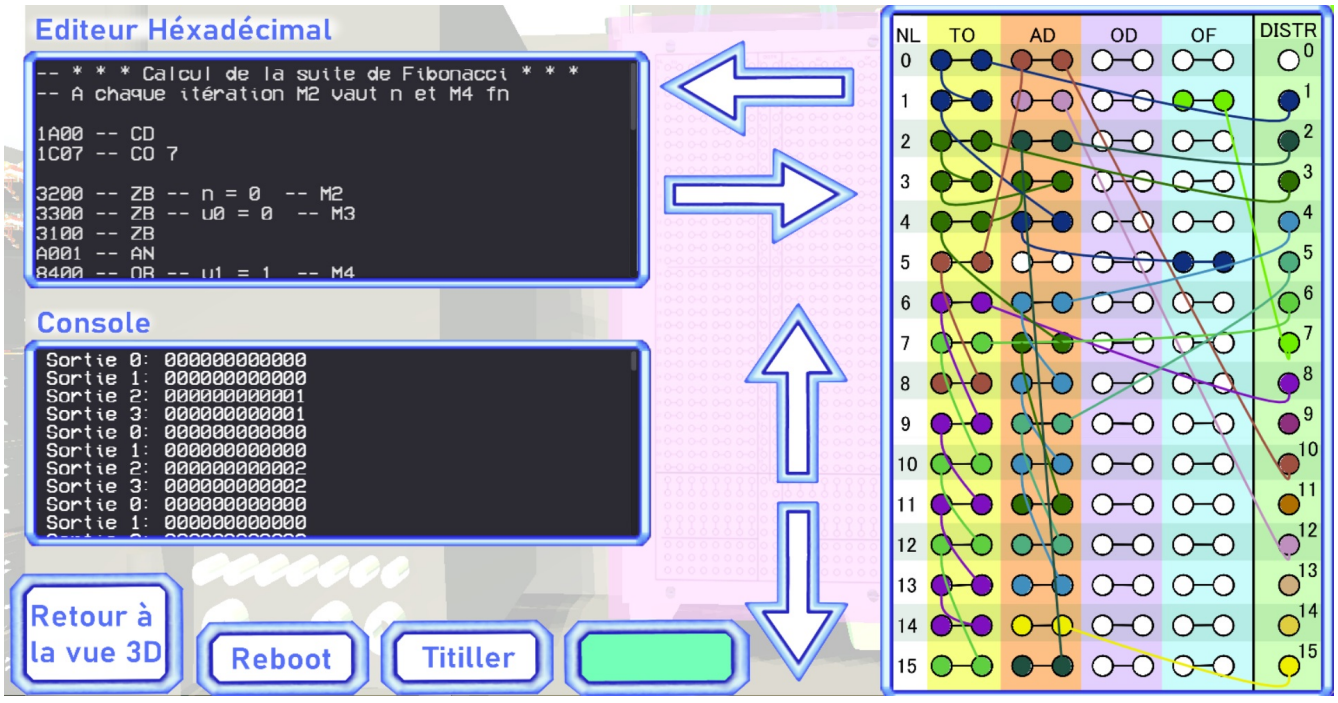
Executer ligne par ligne ou Tout éxecuter

Défile entre les "pages" du tableau de programmation:
0-15
16-31
32-47
48-63

Par exemple, on peut charger le code du calcul de la suite de Fibonacci, qui est de base dans l'éditeur :



Et le lancer sur l'émulateur :



On peut aussi programmer directement sur le panneau. Pour cela, il faut d'abord selectionner une valeur en cliquant sur le trou d'une des constantes. Ensuite, un disque blanc apparait dans ce trou. Cela signifie que la constante est sélectionnée.

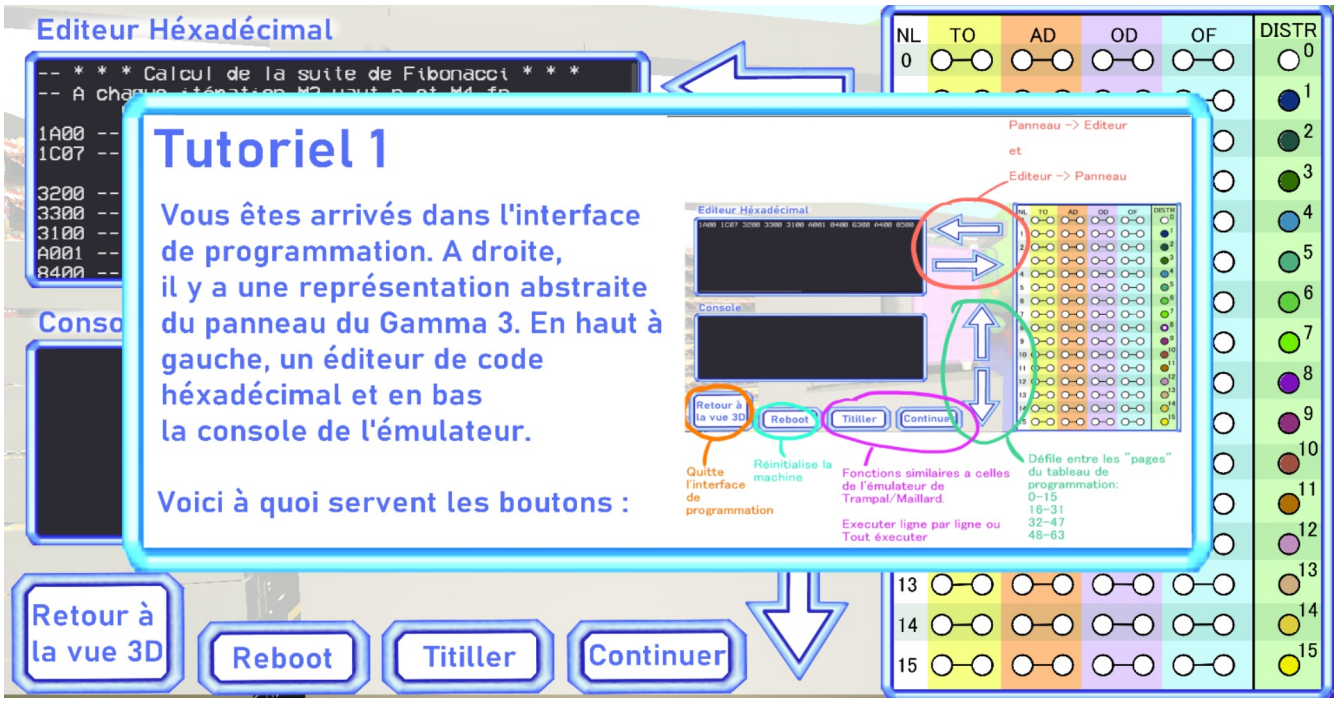
On peut ensuite mettre autant de trous d'instruction que l'on veut a cette valeur en cliquant dessus. Un très léger nettoyage du cablage sera effectué pour avoir une chaine plus réaliste (En vrai, on reprendrait souvent la constante d'une instruction précédente pour la distribuer aux autres champs d'instruction proches, d'où les deux trous), mais l'algorithme n'est pas encore totalement au point.

v. fonctionnalités expérimentales

v.1 tutoriel

Une mécanique permettant de rédiger un tutoriel qui peut continuer ou non selon la sortie de la machine a été mise en place. Les pages du tutoriel sont stockées sous forme d'images, ce qui permet de facilement inclure des images dans l'énoncé, et de les éditer avec n'importe quel logiciel d'édition d'image.

Cela permettrait de faire une introduction rapide aux instructions du Gamma 3. Par exemple, on pourrait d'abord expliquer comment afficher un registre, puis quand c'est fait, on passe à la suite qui demande des manipulations simples sur les registres. Et enfin on arriverait au programme de la suite de Fibonacci.



L'élaboration d'un bon tutoriel serait importante pour permettre aux utilisateurs de s'initier aux bases de la programmation du Gamma3 en autonomie.

[Je vais faire une ou deux pages présentant rapidement ce qui est dit dans la section 'manuel utilisateur']

v.2 export en éxécutable

Dans les retours qui m'ont été faits, Philippe Denoyelle a relevé un important problème de performances sur certaines machines. De plus, la taille des données à télécharger pour accéder au simulateur web est assez conséquente, et le téléchargement peut prendre très longtemps selon la connexion du client. Ce sont des problèmes propres au fait de faire de la 3D sur le Web, et qui peuvent être surmontés avec énormément d'optimisations.

Une autre possiblité offerte par Godot est d'exporter le projet comme un éxécutable. Sous Windows, on obtient alors un fichier .exe, mais on peut aussi générer l'executable pour Linux ou Mac.

Il serait donc possible de proposer un téléchargement de cet executable en plus de la version Web.

Il y a alors deux possibilités concernant l'API de l'émulateur :

La première solution est de quand même se connecter à l'API distante hébergée sur le serveur d'Aconit.

La deuxième solution consiste à lancer un serveur local pour l'API sur la machine où on va utiliser l'executable. Ainsi, on peut utiliser le simulateur sans connexion internet. Cela demande cependant un peu plus de travail car il faut alors installer node.js, se rendre dans le dossier 'emulator', configurer api.js pour heberger sur localhost, et lancer la commande 'node api.js' avant d'éxecuter le simulateur 3D.

VI. Bilan

En conclusion, les bases du simulateur sont fonctionelles, mais il est encore assez loin d'être parfaitement ergonomique. J'aurais aussi aimé modéliser plus de pièces du Gamma 3, pour l'instant il n'y a que le panneau de connexions et les tiroirs, ainsi qu'une esquisse de chassis.

Durant ce projet, j'ai beaucoup appris. D'une part à utiliser Blender et Godot, deux logiciels qui m'ont énormément plu et que je continuerai d'utiliser pour mes projets personnels. Mais aussi les bases du javascript, et quelles sont les contraintes du développement Web.