Rapport de projet
P42 : Automatisation de l'assemblage de LEGO

Eloi Zalczer et Justine Senellart

Polytech Lille, IMA S8

Remerciements

Dans un premier temps nous aimerions remercier l'équipe pédagogique de la spécialité Informatique, Microélectronique et Automatique pour les enseignements qui nous ont permis de réaliser ce projet.

Ensuite, nous voudrions remercier M. Redon, M. Boé et M. Vantroys pour le temps qu'ils nous ont accordé, pour leurs conseils et pour le matériel qui nous a été prété. Plus particulièrement, nous voulons remercier M. Redon pour avoir imprimé toutes nos pièces avec son imprimante 3D.

Nous tenons à remercier Thierry Flamen pour son expertise et son aide lors de la réparation des contrôleur moteur et de la fixation de la tête de préhension.

Nous remercions aussi les Fab. Manager pour le prêt de l'imprimante ainsi que pour le temps qu'ils nous ont accordé et l'aide apportée lors de l'utilisation de la découpeuse laser.

Sommaire

Introduction

Dans le cadre de notre deuxième année en cycle ingénieur en Informatique, Micro-électronique et Automatique, nous avons l'occasion de réaliser un projet permettant de mettre en oeuvre les connaissances vues dans l'année et d'en découvrir de nouvelles.

Notre projet à pour but d'aider les gens à gagner du temps lors du montage de leur Lego. En effet, nous nous sommes rendu compte qu'ils ont de moins en moins de temps à consacrer aux petites briques en plastique. Notre ambition est de permettre aux gens de pouvoir consacrer plus de temps à leur passion en leur proposant une machine pour les assister dans leurs constructions. Cette idée est née de la passion pour les Lego de Justine et nous l'avons adoptée car nous avons jugé que ce projet abordait de nombreuses problématiques liées à IMA. En effet, nous avons du établir une interface entre un système électromécanique tel qu'une imprimante 3D et un système informatique.

Nous nous sommes inspirés d'une construction réalisée à l'aide des robots de Lego : les Mindstorm. Nous avons cherché un moyen d'améliorer ce projet et de le construire dans des matériaux plus solides et durables que les Lego. Le projet dont on s'est inspiré est le *Bricasso* [1] qui est un scanner à Lego : il est capable de lire un dessin en pixel art pour le reproduire en Lego.

Chapitre 1

Présentation du projet

1.1 Description

L'idée de notre projet est simple. À l'aide d'une base d'imprimante 3D récupérée au Fabricarium, communiquant avec une application hébergée sur Raspberry Pi, nous avons voulu créer un système capable de monter des objets en LEGO. Pour ce faire, nous avons modifié la tête d'impression de façon à ce qu'elle puisse se saisir de pièces LEGO et les placer sur une plaque LEGO classique. Du côté de l'application, notre but était de fournir à l'utilisateur un moyen facile pour designer et imprimer un modèle simple en LEGO.

1.2 Objectifs

L'objectif global du projet est de faire en sorte que l'imprimante monte des Lego sur une couche tout en respectant un modèle entré par un utilisateur sur une application Web. Pour cela on peut décomposer le projet en plusieurs objectifs :

- Créer une "tête d'impression" pour agripper les Lego.
- Réaliser un réservoir à pièce pour stocker les pièces Lego.
- Modifier le plateau de manière à pouvoir y placer une plaque Lego et le réservoir.
- Réaliser une application Web pour la gestion du modèle et du réservoir.
- Adapter le firmware de l'imprimante et le G-code pour notre utilisation.

Chapitre 2

Préparation

2.1 Analyse du projet

Avant de nous lancer dans le travail, nous avons réservé une période pour nous consacrer à l'analyse du projet. Nous avons donc répondu aux problématiques qui nous ont été posées par nos tuteurs.

2.1.1 Analyse des concurrents

Dans un premier temps, nous nous sommes donc intéressés aux concurrents potentiels de notre projet, ce qui n'a pas été facile car l'idée est assez originale. Nous avons fini par en trouver deux pouvant s'en rapprocher.

— **Bricasso**, que nous avons brièvement présenté plus haut car il s'agit d'une de nos sources d'inspiration, est un projet réalisé entièrement en LEGO Mindstorm. L'imprimante permet de scanner un dessin fait à la main et de le reproduire en créant une mosaïque de pièces LEGO. A priori, ce projet présente de nombreuses limitations : impression uniquement sur une couche, un seul type de pièces etc... De plus, l'imprimante n'est pas connectée à une application et peut seulement être contrôlée via le module de commande Mindstorm. Notre première analyse était donc que nous pouvions améliorer ce projet dans chacune de ces directions et offrir plus de possibilités à un utilisateur.

— **LEGO Digital Designer** est un logiciel développé par LEGO permettant de modéliser et créer des notices pour le montage de LEGO. Il ne s'agit pas à proprement parler d'un concurrent de notre projet puisqu'il n'est pas question d'impression 3D, cependant nous nous y sommes intéressés pour trouver de l'inspiration et tester la possibilité d'une éventuelle compatibilité. Nos recherches ont donné assez peu de résultats, puisque dans chacun des cas nous n'avons rien obtenu d'utile. L'application de modélisation est entièrement en 3D et bien trop lourde pour ce que nous voulions et pouvions faire. Les formats d'export des modèles, que nous voulions éventuellement utiliser pour une interface, sont très mal documentés et visent des logiciels de modélisation 3D. Il n'y a aucun moyen d'exporter une liste de pièces dans un format XML, JSON ou autre. Dans un dernier temps, nous avons regardé s'il était possible de récupérer la liste de pièces du logiciel mais celle-ci était de toute façon bien trop importante pour nous, et nous l'avons récupérée plus tard via une API publique.

2.1.2 Questions difficiles

Trois questions difficiles nous ont été posées lors de la première présentation de notre projet. Ces questions visaient à nous pousser à envisager des problème auxquels nous n'avions pas forcément pensé, et nous les avons donc prises très au sérieux.

Comment démarrer l'imprimante à distance ?

Pour que l'expérience utilisateur soit celle que l'on souhaite, il est nécessaire que l'impression puisse être déclenchée depuis l'application Web si l'imprimante est allumée et connectée. Ainsi, nous avons détaillé le processus d'impression. On suppose que l'installation est complète, c'est à dire que le serveur de l'application et le serveur d'impression sont tous les deux en ligne et accessibles. Pour lancer l'impression de son modèle, l'utilisateur clique sur le bouton *Imprimer* de l'application. L'application envoie alors une requête au serveur d'impression. Si le serveur d'impression est bien joignable, ce dernier tente alors d'ouvrir le port série vers l'imprimante et renvoie un code 500 en cas d'échec. Une fois connecté à l'imprimante, le serveur envoie les instructions en G-Code au fur et à mesure pour ne pas surchager le buffer de l'imprimante.

Un problème évident avec notre système, que nous ne pouvons malheureusement pas vraiment pallier, est le manque de contrôles. En effet, l'impression peut virtuellement se lancer sans la plaque d'impression, avec la tête mal placée ou autres. Pour éviter ces problèmes, il faudrait utiliser une flopée de capteurs et/ou usiner des pièces de façon à limiter le jeu des endstops notamment, ce que nous ne pouvons pas nous permettre de faire.

Comment sera calibrée la plaque d'impression ?

La plaque d'impression utilisée sera fournie avec l'imprimante. Il s'agira d'une plaque LEGO classique, ses dimensions seront donc parfaitement connues. Si l'utilisateur décide de créer un projet de dimensions inférieures dans l'application web, le coin de la plaque sera utilisé.

Étant donné que la plaque sur laquelle sera "imprimé" l'objet fait partie de l'objet il nous faut trouver un moyen pour qu'elle soit toujours à la bonne position. Nous avons pensé à réaliser un coin contre lequel un des coin de la plaque sera bloqué (comme sur un massicot pour bloquer les feuilles de papiers). L'utilisateur devra s'assurer que la plaque est dans le bon sens et qu'elle repose bien contre le coin. Encore une fois, aucune vérification ne pourra être effectuée malheureusement.

Comment la plaque va se fixer sur le sol ?

Il nous fallait trouver un moyen pour que l'utilisateur puisse enlever et remettre la plaque d'impression sur l'imprimante tout en faisant en sorte que cette plaque soit toujours au même endroit pour éviter de devoir recalibrer à chaque fois. Nous avons donc décidé de réaliser des sortes de cales en bois qui sont fixées sur la base de l'imprimante. Il y a très peu de jeu entre ces cales et la plaque de manière à ce que la plaque reste en place pendant les mouvements de l'imprimante. Par la même occasion nous avons décidé de reprendre le même système pour "fixer" le réservoir comme ça il peut facilement être enlevé pour le remplissage.

2.2 Cahier des charges et choix technologiques

2.2.1 Partie matérielle

Pour la base du projet, nous avons décider de modifier une imprimante 3D, car pour placer les Lego nous avons besoin de bouger avec précision sur les axes x, y et z ce qui est possible avec les NEMA d'une imprimante 3D.

- Pour le réservoir à pièces nous avons choisi de faire un "toboggan" qui permettra de stocker une quinzaine de pièces de cinq couleurs différentes. Ce réservoir sera réalisé en impression 3D.
- Pour la "tête d'impression" nous utiliserons une pièce en 3D avec une articulation et une pièce Lego 1x1 à son extrémité de manière à pouvoir facilement agripper et relâcher les pièces.
- Il faudra aussi refaire le plateau en bois de l'imprimante de manière à ce qu'il puisse accueillir la plaque Lego et le réservoir à pièces.

2.2.2 Partie logicielle

Nous avons choisi d'adopter une architecture Cloud pour notre application, car un des points importants de notre projet est que les gens puissent designer leurs modèles depuis n'importe où dans le monde. Une fois créés, les projets peuvent être envoyés en local vers l'imprimante. Notre architecture se divise donc en trois parties : l'application backend, l'application frontend et le code côté imprimante permettant de lancer l'impression. Le cahier des charges final de notre projet est le suivant :

- **Partie Frontend**
  - Proposer une interface paramétrable de création du modèle, en trois dimensions. Cette interface doit permettre de placer les pièces selon des règles de placement adaptées, de les faire tourner et les supprimer.
  - Gérer les différentes couleurs de pièces possibles.
  - Permettre d'envoyer le modèle créé vers l'imprimante et de gérer le réservoir à pièces.
  - Permettre de créer, sauvegarder, ouvrir et supprimer (CRUD) un projet.
- **Partie Backend**
  - Mettre en place une base de données permettant de gérer les projets.
  - Proposer une API facilement utilisable par l'application, en tirant parti des différentes méthodes HTTP et en utilisant les bons codes de retour.
- **Partie Imprimante**
  - Mettre en place une API minimaliste permettant de générer le G-Code d'un projet et de lancer l'impression.

Les choix technologiques de notre projet ont été guidés par la simplicité, la compatibilité et la facilité d'interfaçage des différents éléments du projet. En choisissant d'utiliser JavaScript pour toutes les parties de l'application, nous nous sommes ouverts à une utilisation facile du format JSON, et avons évité un temps important de montée en compétence sur un éventuel autre langage. En détail, nous avons effectué les choix suivants :

- **Backend**
  - Node.js : API RESTful
  - PostgreSQL : Base de données projets et utilisateurs

- **Frontend**
  - HTML5/CSS3/JS natif
  - Utilisation de la balise canvas pour la conception du schéma.
  - Envoi des données en JSON
- **Imprimante**
  - Serveur Node.js sur Raspberry Pi
  - Utilisation du firmware Marlin

2.3 Gestion de projet

Nous avons eu tendance à beaucoup travailler séparément durant le projet, il était donc absolument nécessaire d'avoir une bonne méthodologie et une bonne confiance dans l'autre binôme pour que le projet avance correctement. Nous avons adopté une sorte de méthode Scrum, avec de courtes réunions informelles régulières pour rester informés de l'avancée de l'autre. Le wiki de projet nous a servi de journal de bord et nous nous sommes efforcés de le remplir régulièrement pour garder une trace écrite de notre avancée.

Nous nous sommes aperçus assez rapidement que nous allions devoir révoir à la baisse notre cahier des charges, ce qui nous a forcés à prendre des décisions et à communiquer efficacement. Toutes ces décisions ont été prises assez facilement et sans désaccord. Elles ont été causées principalement par de mauvaises évaluations du temps nécessaire pour certains travaux et des limites techniques notamment pour la diversité des pièces.

Pour toute la gestion de notre projet, nous avons utilisé la plateforme Gitlab de Polytech Lille. Cela nous a permis notamment de transférer facilement notre code entre la Raspberry Pi du projet, nos PC personnels et les Zabeth en salles E304/306. Notre utilisation de la plateforme est restée minimaliste, car nous n'avons pas travaillé sur les mêmes parties du projet et avons principalement utilisé une seule branche. Nous avons néanmoins fini par créer une branche dev pour la version du code non définitive. Nous nous sommes attachés à nommer correctement nos commits pour que l'arborescence reste compréhensible.

Chapitre 3

Réalisation du projet

3.1 Partie Informatique

La partie informatique de ce projet a été constituée en grande majorité par le développement de l'application Web. En effet, cette dernière avait à l'origine un cahier des charges très vaste, qui a été revu au fur et à mesure. Nous avons commencé cette partie dès le choix du sujet, c'est à dire au mois de Novembre. Cela nous a permis de présenter une première version de l'application au début du projet.

3.1.1 Application : arborescence, framework et gestion des plugins

Une des réflexions préliminaires de notre projet a été le choix des technologies utilisées, et en particulier des frameworks et plugins côté serveur. En Node.js, les plugins sont gérés par un programme appelé npm. Ce dernier permet également de générer les fichiers essentiels à l'arborescence d'un logiciel. En particulier, il génère un fichier appelé package.json et permet de le peupler facilement. Ce dernier sert à renseigner un certain nombre d'informations sur le projet, dans le but de permettre sa publication. Il peut notamment contenir le nom du projet, le dépôt Git correspondant, l'adresse de la documentation, différentes options de configuration, la liste des dépendances etc...

L'arborescence de notre application est la suivante :

```
server/
├── server.js
├── database.sql
├── package.json
├── api
│   ├── controllers
│   │   └── legoControllers
│   └── routes
│       └── legoRoutes
├── public
│   ├── js
│   │   ├── get_list_pieces.js
│   │   ├── init_project.js
│   │   ├── lego_main.js
│   │   ├── print.js
│   │   ├── save.js
│   │   ├── utils.js
│   │   ├── delete.js
│   │   └── abort.js
│   └── style
│       ├── dropdown.png
│       └── style.css
├── views
│   └── lego_main.html
├── lib
│   └── postgres.js
└── node_modules
```

Nous avons apporté un soin particulier à respecter une arborescence logique et compréhensible. La partie API et la partie frontend ont été séparées dans deux dossiers indépendants de façon à clarifier la distinction. Tous les fichiers concernant le frontend sont regroupés dans les dossiers *public* et *views*. En outre, l'accès à la base de données a été fait sous forme de librairie afin d'alléger le code de *server.js*.

Pour la gestion des routes de notre API, nous avons utilisé le framework Express qui a l'avantage d'être bien documenté et rapide à mettre en place. Il nous permet de gérer simplement les différentes routes, les méthodes HTTP, les codes de réponse et bien d'autres éléments essentiels du projet. En terme de plugins, nous avons tenté de rester minimalistes car les plugins npm ont tendance à prendre beaucoup de place et à causer des problèmes de compatibilité. Les plugins que nous avons utilisés sont les suivants :

- **ejs 2.5.7 :** pour le rendu des pages HTML.
- **express 4.16.2 :** le framwork Express.
- **pg 7.4.1 :** plugin de gestion de base de données Postgres.
- **body-parser 1.18.2 :** parsing des requêtes HTTP.

Pour la partie imprimante, nous avons également utilisé le plugin **serialport** qui permet, comme son nom l'indique, d'accéder au port série de la Raspberry Pi et de déclencher l'impression. Nous avons du nous contraindre à utiliser une version de Node

assez ancienne (8.10.0 contre 10.1.0 actuellement) car le plugin n'est pas compatible avec les dernières versions de Node sur Raspberry Pi.

3.1.2 Application : le backend

La partie backend de l'application est basée, comme indiqué plus haut, sur une API RESTful en Node.js. Cette API est assez minimaliste, car les services à fournir sont peu nombreux et afin de ne pas surcharger la Raspberry Pi. Elle fait la liaison avec une base de données Postgres qui permet de sauvegarder les projets de l'utilisateur, ainsi que les différents types de pièces disponibles. Ainsi, une des premières choses que nous avons définies est le schéma UML de notre base de données.

Celui-ci est constitué de quatre tables. La première table permet de sauvegarder les projets. Une seconde contient la liste des pièces disponibles (1x1, 1x2, 2x2). Cette dernière n'est plus utilisée dans la version actuelle du projet car seules les pièces de dimensions 1x1 sont supportées, cependant nous avons décidé de laisser ouvert le point d'entrée API pour une éventuelle future version. La troisième table fait la relation entre les projets et les pièces et enregistre le contenu de chaque projet. Enfin, la quatrième table est à part et contient la liste des couleurs de pièces possibles. La première version de cette table contenait la totalité des couleurs LEGO existantes, mais nous avons décidé de réduire cette liste à 16 valeurs pour faciliter l'expérience utilisateur.

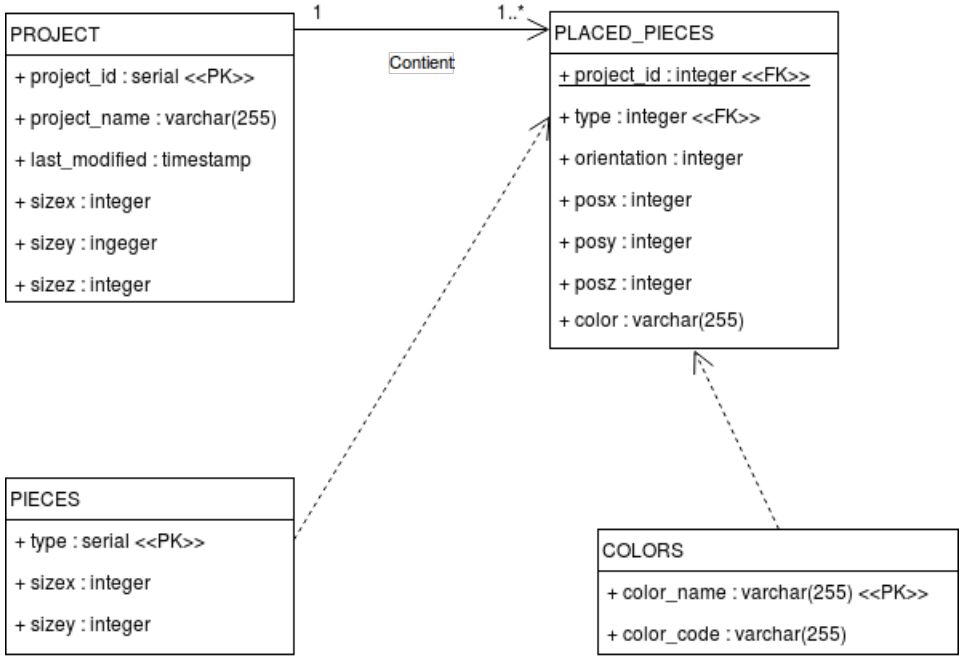

FIGURE 3.1 – MCD du projet

Les points d'entrée de l'API sont les suivants. Tous les résultats sont renvoyés en JSON à l'exception de la page HTML :

- **GET** / : renvoie la page principale du projet, rendue grâce au framework Express.
- **GET** /**project** : renvoie la liste des projets.
- **PUT** /**project** : crée un nouveau projet. La requête doit alors contenir le nom du projet à créer, ainsi que les dimensions du canvas.
- **GET** /**project**/([0-9]+) : renvoie le projet correspondant à l'identifiant donné.
- **PUT** /**project**/([0-9]+) : sauvegarde le projet correspondant à l'identifiant donné.
- **DELETE** /**project**/([0-9]+) : supprime le projet correspondant à l'identifiant donné.
- **GET** /**pieces** : renvoie la liste des pièces disponibles. Ce point d'accès n'est plus utilisé depuis que nous avons décidé de gérer uniquement des pièces de taille 1x1.
- **GET** /**colors** : renvoie la liste des couleurs possibles pour les pièces.

La structure de notre API, comme nous l'avons précisé plus haut, est divisée entre les fichiers legoRoutes.js et legoControllers.js. Nous pouvons analyser une petite partie du code pour comprendre son fonctionnement.

```
1  'use strict';
2  module.exports = function(app) {
3      var lego = require('../controllers/legoControllers');
4
5      // todoList Routes
6      app.route('/').get(lego.mainpage);
7
8      app.route('/project')
9      .get(lego.list_projects)
10     .put(lego.create_project);
11
12     app.route('/project/:project_id')
13     .get(lego.open_project)
14     .put(lego.save_project)
15     .delete(lego.delete_project);
16
17     app.route('/pieces')
18     .get(lego.list_pieces);
19
20     app.route('/colors')
21     .get(lego.list_colors);
22
23 };
```

Les routes et les protocoles sont déclarés de façon très claire et en très peu de code grâce au *method chaining* de JavaScript qui permet d'appeler plusieurs méthodes sur le même objet en une même ligne.

Pour les controllers, nous étudierons seulement le point d'accès open_project qui permet comme son nom l'indique de charger les données d'un projet. Cette fonction a l'avantage d'être assez courte (elle n'utilise qu'une seule requête SQL) et de bien démontrer le fonctionnement d'Express.

```
1  'use strict';
2
3  var pg = require('../../lib/postgres');
4
5  exports.open_project = function(req, res) {
6      var sql = 'SELECT * FROM PLACED_PIECES, PIECES WHERE project_id = $1
       AND PLACED_PIECES.type=PIECES.type';
7      pg.client.query(sql, [ req.params.project_id ], function(err,
      results){
8    if(err){
9        console.error(err);
10       res.statusCode = 500;
11       return res.json({errors: ['Could not open project'] });
12   }
13   res.statusCode = 200;
14   console.log(results.rows);
15   return res.json(results.rows);
16     });
17 };
```

Les paramètres de l'URL sont récupérés via le mot-clé $1 qui, comme en bash, permet d'accéder au premier paramètre. Le contenu de la requête GET est lui récupéré depuis l'objet req.params. La requête SQL est générée et envoyée grâce à la méthode pg.client.query, et la réponse HTTP renvoyée prend en compte le résultat de la requête et renvoie un message facilement affichable à l'utilisateur.

3.1.3 Application : partie fonctionnelle

La partie fonctionnelle de l'application a été la plus longue à développer, notamment car nous avons décidé d'utiliser uniquement du Javascript natif pour maximiser la compatibilité. L'objectif était de proposer à l'utilisateur une interface intuitive pour concevoir son modèle. La solution que nous avons adoptée directement est d'utiliser des balises canvas HTML5 pour permettre à l'utilisateur de "dessiner" dessus. L'idée est simple : plusieurs canvas sont empilés au même emplacement, avec un z-index différent de façon à maîtriser leur ordre. Un premier canvas appelé *background* permet d'afficher la couleur de fond. Au-dessus de ce dernier se situent les canvas sur lesquels l'utilisateur dessine, et qui sont en nombre égal à la dimension du projet en z. Encore au-dessus se situent deux autres canvas, pour afficher la grille de dessin et le placement de pièce correspondant à la position actuelle de la souris.

La première partie du développement a été de gérer la partie graphique, c'est à dire de faire en sorte que les canvas réagissent correctement aux mouvements de la souris et aux clics de l'utilisateur. Cette partie a globalement été assez rapide car les coordonnées des pièces peuvent facilement être calculées en fonction des dimensions du canvas et du nombre de points de la grille. Nous avons néanmoins développé une fonctionnalité de rotation des pièces au clic droit, qui s'avère désormais inutile mais qui fut plus longue à implémenter. Nous avons également implémenté un mode *Supprimer* qui permet comme son nom l'indique à l'utilisateur de retirer une pièce déjà placée. Enfin, la gestion des couleurs est venue plus tard, autour de la semaine 5 du projet. Une fois que l'utilisateur était en mesure de créer son modèle, nous avons commencé à nous pencher sur les autres

parties de l'application. La première chose à mettre en place était bien entendu l'interface entre l'application et l'API, et particulièrement déterminer la façon dont les données seraient envoyées de l'un vers l'autre. Nous avons eu quelques problèmes qui nous ont amenés à revoir le MCD une fois ou deux, et nous avons utilisé du JSON pour la simplicité d'interprétation, le serveur et le client étant tous les deux en Javascript.

Une fois que nous avions déterminé définitivement le modèle de données utilisé, nous avons commencé à implémenter les interfaces graphiques permettant à l'utilisateur d'accéder aux différents points API. Nous avons choisi d'utiliser un système simple à base de fenêtres pop-up. Au lancement du projet, l'utilisateur choisit sur une première fenêtre s'il souhaite créer un nouveau projet ou en ouvrir un déjà existant. Le reste de la page est alors grisé et non éditable. Une fois que l'utilisateur a choisi une option valide, la fenêtre se ferme et laisse la place à l'édition. Une autre fenêtre s'ouvre lorsque l'utilisateur souhaite lancer une impression, que nous avons implémentée bien plus tard. Cette dernière a deux objectifs : demander à l'utilisateur de rentrer l'adresse IP de l'imprimante, et de renseigner la position des pièces de différentes couleurs dans le réservoir de l'imprimante.

Bien entendu, l'application a subi de nombreuses modifications, améliorations et optimisations tout au long du projet. Une des parties qui a pris le plus de temps à implémenter a été les règles de placement des pièces. En effet, les pièces de LEGO ne peuvent pas être placées n'importe comment, et il existe donc deux règles : une pièce ne peut pas être placée au même endroit qu'une autre, et une pièce ne peut pas être placée sur un calque plus haut s'il n'y en a pas une pour la soutenir sur le calque du dessous. Nous avons pris beaucoup de temps à programmer cette fonctionnalité, car il était difficile de trouver une solution marchant avec tous les types de pièces. Nous avons également eu des problèmes de performance, car nous étions obligés de vérifier à chaque nouveau clic la possibilité ou non de placer la pièce, et donc de parcourir l'intégralité des pièces déjà placées. En nous limitant aux pièces de taille 1x1, cette partie du code a été grandement simplifiée et l'implémentation finale est assez simple.

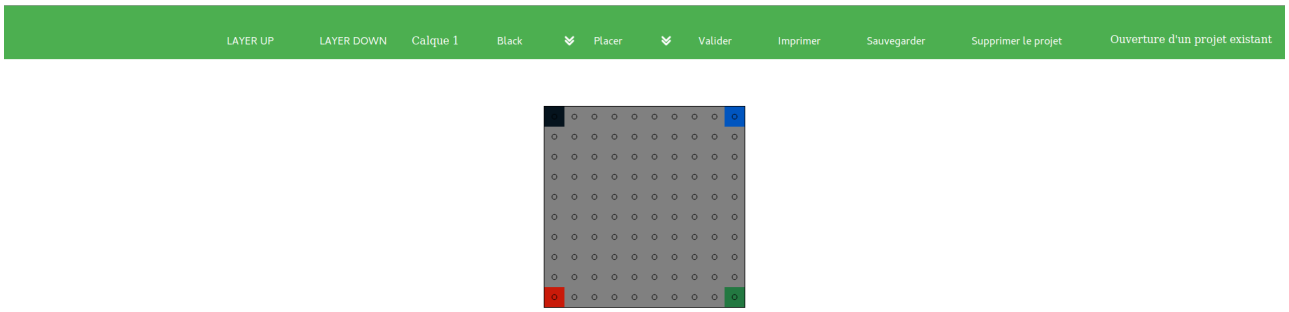

FIGURE 3.2 – Design du projet

Probablement la dernière partie de l'application dont nous nous sommes occupés était le design. Jusqu'à la semaine 11, la page était constituée uniquement de boutons et de texte en plus des canvas, sans vrai design. De plus, des parties de la page étaient devenues inutiles. Nous avons donc créé un design simple rendant l'application plus agréable à utiliser. Le fait de faire ce design nous a également permis de nous rendre compte de certains manques ou problèmes dans l'application, que nous avons corrigés par la suite. Dans la version en développement du code, on peut trouver plusieurs vestiges de fonctionnalités que nous avons envisagé d'implémenter avant de nous raviser, notamment la possibilité d'annuler/refaire avec Ctrl+Z et Ctrl+Y. Néanmoins, la version de production sur la branche **master** a normalement été nettoyée du code mort et correctement commentée.

3.1.4 Imprimante : génération et envoi du G-Code

Le serveur utilisé pour la génération du G-Code et la gestion de l'imprimante est un programme totalement séparé de la partie backend de l'application. Après discussion avec nos tuteurs, nous avons décidé de le développer également en Node.js afin de ne pas perdre de temps sur l'architecture et de pouvoir nous concentrer sur la génération du G-Code. Le serveur est très simple et constitué d'uniquement deux fichiers. Le fichier *server.js* contient simplement les composants d'un serveur minimaliste capable de répondre à des requêtes POST, ainsi que les fonctions d'initialisation et de gestion du port série pour la communication avec l'imprimante. Le fichier *gcode.js*, comme son nom l'indique, contient les fonctions permettant de générer le G-Code ensuite envoyé à l'imprimante.

L'imprimante 3D utilise le framework Marlin [2] pour fonctionner. Ce framework implémente sa propre version de G-code [3]. Globalement, les seules commandes que nous sommes amenés à utiliser pour notre projet sont G0/G1 pour les mouvements linéraires, G28 qui permet de retourner la tête à sa position d'origine, et éventuellement G90/G91/G92 pour paramétrer le système de coordonnées. Nous avons paramétré le framework de façon à ignorer les problématiques de température ou de débit liées à l'impression 3D et inutiles dans notre cas.

La génération du G-Code est en fait un processus assez simple. Ce dernier est généré dans sa totalité avant d'être envoyé à l'imprimante afin d'éviter une impression partielle en cas d'erreur de génération. Nous avions dans un premier temps commencé par travailler en coordonnées relatives, puis nous sommes repartis en coordonnées absolues par simplicité. La première ligne de G-Code envoyée est systématiquement la commande G28 afin de ramener la tête d'impression à sa position initiale avant de commencer l'impression. Puis, le mouvement est décomposé en six parties :

- Montée de la tête d'impression de 15 unités.
- Déplacement vers la position du réservoir.
- Baisse puis remontée de la tête d'impression.
- Déplacement vers la position où déposer la pièce.
- Baisse de la tête d'impression.
- Mouvement de deux centimètres en X pour détacher la pièce.

Ce code forme une boucle qui peut se répéter jusqu'à arriver à la fin du modèle. Une fois généré, le gcode est retourné par la fonction et peut alors être envoyé vers l'imprimante. A chaque ligne de G-Code envoyée et exécutée avec succès, l'imprimante retourne le message

"ok". Nous pouvons alors envoyer la ligne suivante. L'imprimante dispose d'un buffer permettant d'envoyer plusieurs commandes à la fois, cependant sa taille est assez limitée (16 par défaut) et il est donc plus raisonnable d'envoyer les commandes une par une. A l'origine, nous souhaitions utiliser le code de retour de la requête XHR pour signifier à l'utilisateur si l'impression s'était bien déroulée ou non. Cependant, étant donné la durée d'une impression, cela n'a pas été possible car nous aurions reçu un timeout. Le seul contrôle pouvant s'effectuer depuis l'application est l'ouverture du port série, ce qui permet d'afficher un message à l'utilisateur si l'imprimante est mal connectée.

3.2 Partie mécanique

La partie mécanique du projet concerne toute la partie physique. Nous avons récupéré une imprimante 3D *Prusa i3* [4] pour nous servir de base. Nous avons donc dû dans un premier temps la remettre en état de marche pour pouvoir ensuite l'adapter à l'utilisation que nous voulons en faire.

3.2.1 Réparation de l'imprimante

L'imprimante que nous avons récupéré avait été utilisé dans un ancien projet d'IMA4[5] et avait déjà subit quelques adaptations pour ce projet. De plus, l'imprimante n'était pas en très bon état avec certaines parties manquantes et d'autres qui étaient mal fixées. Lors de la première séance nous nous sommes assurés que les capteurs de fin de course et les moteurs fonctionnaient correctement.

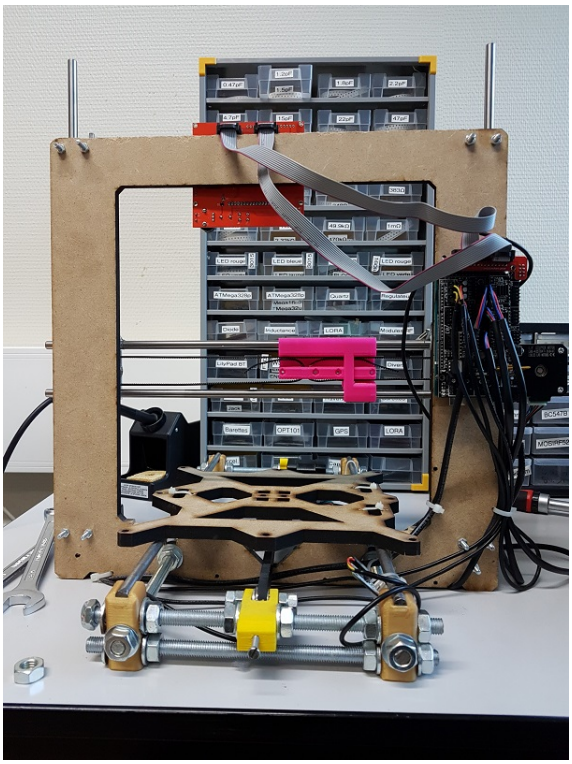

FIGURE 3.3 – état de l'imprimante quand on l'a récupéré

La remise en état de l'imprimante à pris un certain temps car en réparant certaines parties de l'imprimante nous découvrions de nouveaux problèmes. La première étape a

été de revisser toutes les vis et d'en changer quelques unes qui avaient été sciées et ne pouvaient pas être revissées. Une fois que l'imprimante était dans un état un peu plus solide, nous avons refait des tests pour les moteurs et nous nous sommes rendus compte que les capteurs de fin de course qui étaient fixés avec des zip étaient entraînés par les moteurs et n'étaient pas tout le temps activés. Nous avons donc dû réaliser des supports pour les différents capteurs de fin de course [6] qui ont été réalisés en impression 3D.

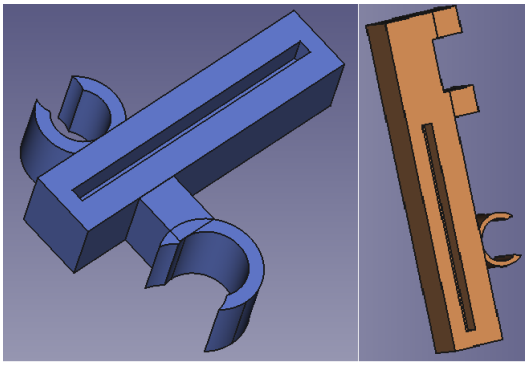

FIGURE 3.4 – Supports capteurs fin de course

Une autre pièce de l'imprimante à dû être réalisée : des clips de maintien de la courroie[7].

3.2.2 Adaptation de l'imprimante

Une fois que l'imprimante était en état de fonctionner correctement il a fallu adapter l'imprimante pour l'utilisation que nous voulions en faire. Dans un premier temps, nous avons modélisé en 3D un genre de toboggan pour distribuer les pièces au fur et à mesure. Ce réservoir est composé de 5 rails, la taille de chacun est adaptée à la taille du type de brique qu'il recevra.
Nous nous sommes ensuite penchés sur le système qui nous permettra d'aller chercher les pièces et de les positionner sur la plaque Lego. Pour cela, il nous faut un système avec une articulation qui permettra de faire en sorte que la pièce reste sur la plaque et non sur la tête. Nous avons donc modélisé deux systèmes d'articulation en 3D : une pièce en un bloc et une pièce composée de deux parties à assembler. La pièce en un bloc s'imprime sans problème notable et présente l'avantage de ne pas nécessiter de raccord.

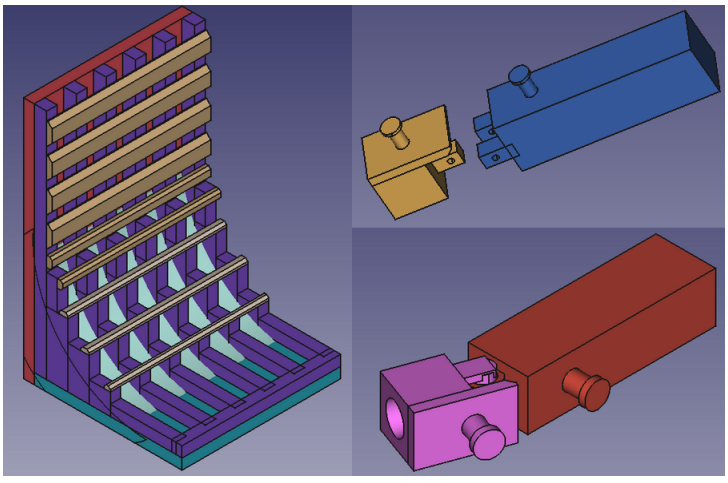

FIGURE 3.5 – Différentes Pièces réalisées

La dernière pièce à réaliser était un support sur lequel on vient caler la plaque Lego ainsi que le réservoir. Il nous a fallu trouver un moyen de fixer corrctement la plaque et le réservoir de manière à ce qu'ils soient toujours au même endroit tout en pouvant les enlever et les remettre assez facilement. Nous avons donc découpé des sortes de rails en bois qui ont été placés avec très peu de jeu pour assurer la stabilité.

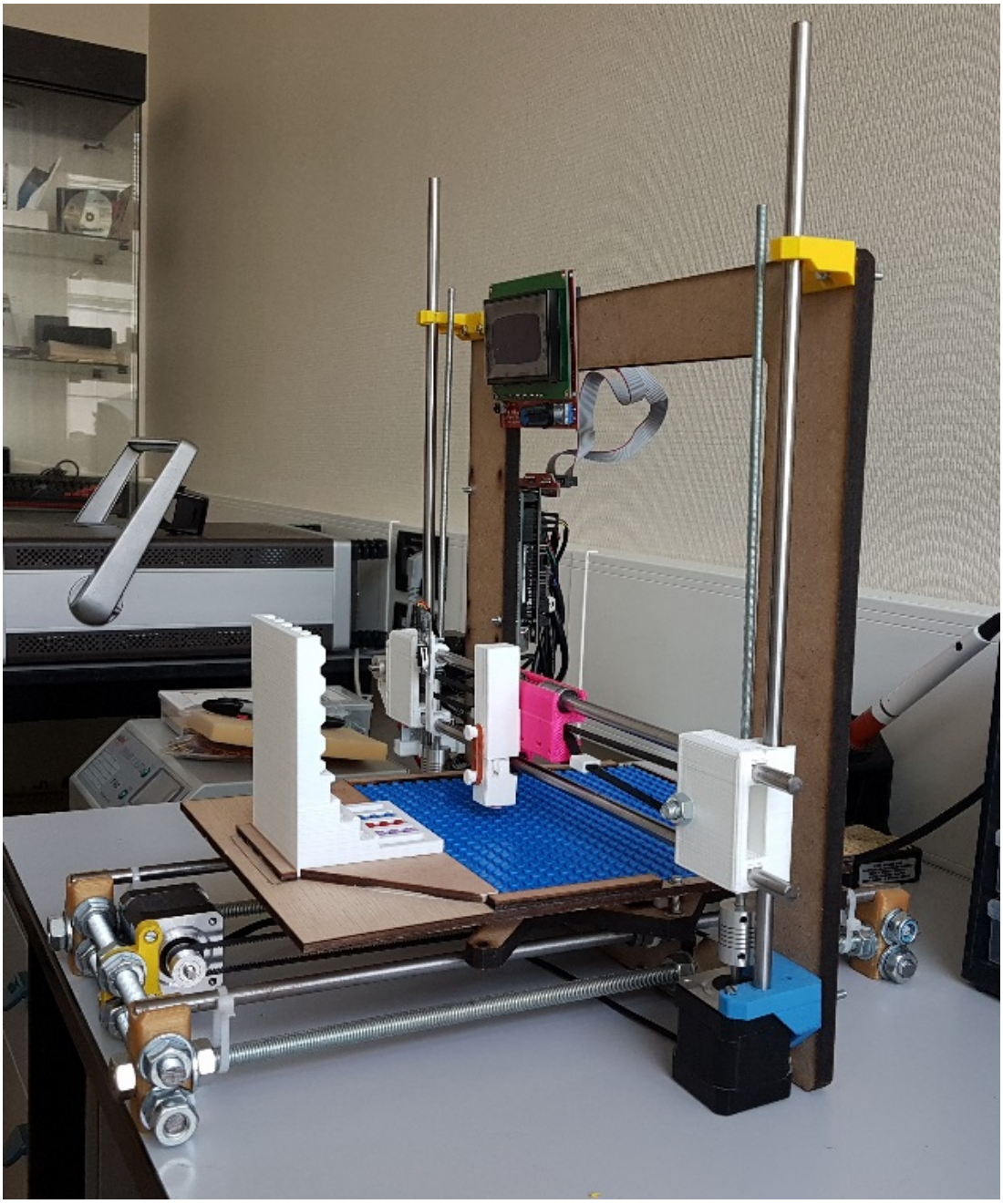

FIGURE 3.6 – Etat final de l'imprimante

3.2.3 Modélisation 3D

La plupart des pièces réalisée pour ce projet ont été faites en impression 3D. Le logiciel utilisé est freeCAD [8] qui est un modeleur 3D permettant de réaliser des objets en prenant des formes de base (cube, cylindre, sphère ...) qui peuvent être modifiés à souhait pour former l'objet désiré. Nous n'avions jamais fait de modélisation 3D auparavant, la prise en

main du logiciel n'a donc pas été des plus évidente et les premières pièces ont été difficiles à réaliser. Nous avons commencé par le réservoir qui reste une pièce assez simple à créer en combinant des carrés et des cylindres pour la pente. La réalisation de cette pièce à tout de même nécessité plusieurs essais pour obtenir le comportement désiré mais elle nous a permis d'acquérir des compétences nous permettant de réaliser plus rapidement les pièces suivantes. Pour réaliser la tête d'impression il a fallu comprendre comment fonctionne l'impression 3D notamment l'utilisation de support pour la réalisation d'une articulation en une pièce. Suite à la réalisation de ces pièces nous sommes maintenant capables de modéliser des pièces simples en 3D.

3.3 Mise en commun

Une fois que les parties mécanique et informatique fonctionnaient correctement individuellement, nous avons fait en sorte de les faire travailler en commun pour donner vie au projet. Il s'est avéré que cette étape était plus difficile que prévue, mais nous avons fini par réussir à faire fonctionner les deux parties comme nous le souhaitions. La première étape que nous avons accomplie était de brancher les éléments ensemble et de tenter d'envoyer une instruction à l'imprimante. Pour ce faire, nous nous sommes intéressés au firmware Marlin et avons décidé de l'adapter pour notre utilisation.

3.3.1 Le firmware Marlin

Le firmware Marlin est le firmware de base sur les imprimantes Prusa i3 dont nous utilisons la base. Nous avons récupéré cette imprimante d'un projet précédent qui avait déjà compilé sa propre version du firmware. Leur projet consistait à imprimer des objets en chocolat, ce qui implique bien évidemment des problématiques différentes des notres (température etc...). Afin de compiler notre propre version de Marlin, nous avons récupéré leurs sources car elles se sont avérées bien plus légères que les dernières versions disponibles en ligne. Nous avons supprimé les parties spécifiques au chocolat et avons modifié des paramètres pour éviter d'utiliser des capteurs ou des fonctionnalités inutiles pour nous. Une fois les modifications faites, nous avons rencontré un certain nombre de problèmes à la compilation : le firmware redéfinissait le type fpos_t qui est défini dans stdio.h et le compilateur C++ a été modifié depuis la création du firmware ce qui a entrainé des erreurs de syntaxe dans notre programme. Le temps d'avoir une version fonctionnelle, nous avons continué à effectuer nos tests avec l'Arduino du projet précédent. Après un certain nombre d'essais, notamment pour connaître le format précis des messages à envoyer (retour chariot + retour à la ligne à la fin de chaque message), nous avons réussi à faire se déplacer l'imprimante. Dans un premier temps, cette dernière ne pouvait effectuer qu'un retour à sa position de départ. Quand nous essayions de faire un mouvement linéaire (G0 ou G1), l'imprimante répondait systématiquement que des endstops étaient atteints. Nous nous sommes rendus compte plus tard qu'il fallait en fait désactiver certains endstops dans la configuration de Marlin, sans quoi il les considéraient comme toujours actifs car ils n'étaient pas présents. Une fois ce dernier réglage effectué, nous avions une version de Marlin parfaitement fonctionnelle et adaptée à notre projet et nous avons pu commencer à effectuer des tests pour calibrer la plaque d'impression et placer correctement les pièces.

3.3.2 Tests et calibration

La phase de tests, contre toute attente, a été très difficile et fastidieuse pour plusieurs raisons. Dans un premier temps, l'échelle utilisée par l'imprimante qui est censée être des millimètres n'est en réalité pas vraiment des millimètres. Ainsi, un déplacement de 1.4cm nécessite une valeur de déplacement d'environ 26. Dans un second temps, nous nous sommes également aperçus que la prise et la pose des pièces nécessitait une précision extrême, non seulement en X et en Y mais aussi en hauteur de la tête. Pour que l'opération se déroule correctement, il faut en effet que la tête descende suffisamment bas pour agripper la pièce, tout en évitant qu'elle ne s'enfonce trop empêchant alors la pose de la pièce. Le réglage doit donc être précis au demi-millimètre, par rapport au endstop qui lui-même n'est pas parfaitement fixé. Nos premiers tests ont été faits hors ligne, en envoyant simplement les données d'un projet par défaut à l'imprimante. Après une après-midi de tests, nous avons réussi à agripper les pièces dans chacun des réservoirs de façon presque systématique. La prochaine session de tests, nous nous sommes appliqués à utiliser l'application web pour envoyer les données, ce qui nous a permis de confirmer que l'interface fonctionnait bien. Nous avons fait un certain nombre de tests pour déterminer les valeurs de déplacement correspondant à l'écart entre deux tenons. Cependant, même après deux après-midi complets, nous n'avons toujours pas réussi à déposer une pièce correctement car le plateau a tendance à se plier sous la force du moteur et la pièce à ne pas se détacher. La précision nécessaire pour que la pièce rentre sans effort à son emplacement est très grande (0.2mm) et, même avec une telle précision, la tête doit descendre avec suffisamment de force pour faire rentrer la pièce et se détacher correctement.

Conclusion

Le but de notre projet était de pouvoir placer une couche de pièces de Lego sur une plaque à l'aide d'une imprimante 3D recyclée en respectant un modèle dessiné par un utilisateur sur une application web. En l'état actuel, nous sommes capables d'aller récupérer des pièces Lego dans le réservoir à l'aide de la tête de préhension que nous avons réalisée et d'effectuer les mouvements pour placer les pièces aux endroits désignés par l'utilisateur. Cependant, nous ne sommes pas en mesure de déposer les pièces sur la plaque. Globalement, nous avons respecté notre cahier des charges : seule la partie de l'application permettant de gérer le remplissage du réservoir n'a pas du tout était traitée.

Nous avons rencontré plusieurs problèmes lors de la réalisation de ce projet, tant sur la partie mécanique avec des pièces qui ont dû être réalisées plusieurs fois pour obtenir quelque chose d'utilisable que sur la partie informatique, notamment pour les règles de placement et la gestion des requêtes cross-origin. Le problème qu'il reste à régler pour pouvoir placer correctement les pièces serait de modifier la tête de préhension car l'articulation n'est pas solide et la pièce bouge lorsque l'on essaye de la mettre sur la plaque ce qui entraine un décalage par rapport aux tenons.

Ce projet nous a permis d'acquérir des compétences dans des domaines liés à l'informatique ainsi qu'à la mécanique et la gestion de projet. Ils nous a dans un premier temps fallu définir les tâches à réaliser et répartir la quantité de travail. La partie mécanique nous a permis d'acquérir des compétences globales sur les imprimantes 3D et la modélisation 3D et la partie informatique une meilleure compréhension de l'architecture REST.

Pour aller plus loin, si ce projet devait être repris, il serait possible d'améliorer la partie mécanique notamment la tête de préhension ainsi qu'améliorer l'application pour pouvoir gérer le remplissage du réservoir. Il serait également possible de reprendre la partie du cahier des charges indiquant que les pièces pourraient être posées sur plusieurs couches. En revanche, la gestion de différentes formes de pièces demanderait un mécanisme totalement différent et paraît un objectif peu réaliste, même si l'application le permet déjà.

Bibliographie

Notes

[1] Bricasso : http://jkbrickworks.com/lego-mosaic-printer/
[2] Marlin : http://marlinfw.org/
[3] G-code : http://marlinfw.org/meta/gcode/
[4] Prusa i3 : https://www.prusa3d.fr/
[5] Imprimante à chocolat : https://projets-ima.plil.fr/mediawiki/index.php/Imprimante_3D_%C3%A0_chocolat
[6] Support endstop x : https://www.thingiverse.com/thing:784801
[7] Clip courroie : https://www.thingiverse.com/thing:820510/
[8] freeCAD : https://www.freecadweb.org/?lang=fr