



Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

**Filière Informatique
Projet de semestre 5
2017-18**

LEGO-2a

Owen McGill

Mandants : iCoSys

iCoSys
Institute of Complex Systems

SeSi

SeSi
Sustainable Engineering
Systems Institute

Superviseurs : Pierre Kuonen
Nicolas Rouvé
Beat Wolf
Jonathan Stoppani

Fribourg, Janvier 2018

Hes·SO
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz

Table des matières

1	Introduction	3
1.1	But du projet	3
1.2	Structure du projet	3
2	État initial du programme	5
2.1	Workflow initial du programme	5
2.2	Programme de conversion	5
2.2.1	Interface graphique	6
2.3	Limitations initiales	7
2.4	Conclusion	8
3	Centrage des modèles	9
3.1	Analyse	9
3.1.1	Calcul du barycentre	9
3.1.2	Modèle avec plaque	9
3.1.3	Comparaison	10
3.2	Implémentation	10
3.2.1	Fichier LXF avec la plaque	10
3.2.2	Code source	11
3.3	Conclusion	12
4	Amélioration du tri	13
4.1	Analyse	13
4.2	Implémentation	14
4.3	Conclusion	16
5	Interface graphique	18
5.1	Analyse	18
5.2	Implémentation	19
5.2.1	Bouton : Nouveau modèle	20
5.2.2	Bouton : Charger un modèle et ouvrir LEGO Digital Designer (LDD)	20
5.2.3	Bouton : Charger un modèle	20
5.2.4	Bouton : Changer la méthode de placement	20
5.2.5	Bouton : Convertir	20
5.2.6	Menu : Model	20
5.2.7	Menu : Configuration	21
5.3	Conclusion	22
6	Vérification de faisabilité	23
6.1	Analyse	23
6.1.1	Hauteur du modèle	23
6.1.2	Dépassement de la plaque de construction	23
6.1.3	Pièces disponible et utilisable	25
6.1.4	Possibilité de poser une pièce	26

6.2	Implémentation	29
6.3	Conclusion	30
7	Conclusion	32
7.1	Limitations résolues	32
7.2	Différence avec le cahier des charges	32
7.3	Idées d'amélioration	33
7.3.1	Serveur de conversion	33
7.3.2	Utilisation de QR codes	33
7.3.3	Modélisation en VR/AR	33
7.4	Avis personnel	33
8	Déclaration d'honneur	34
A	Annexes	35
	Table des figures	35
	Références	35
	Glossaire	36
A.1	Code source	36
A.2	Versions des logiciels/librairie	37
A.3	Guide d'installation	37
A.3.1	Installation du projet pour le développement	37
A.4	Guide utilisateur	37
A.4.1	Création d'un nouveau model	38
A.4.2	Conversion	39
A.5	Ressources pour la base de donnée LDD	40
A.6	Liste des exceptions	41
A.7	Structure clé USB	42

1 Introduction

Le projet LEGO-II est une continuation du projet LEGO-I [1] réalisé durant l'été 2017 à la Haute école d'ingénierie et d'architecture de Fribourg (HEIA-FR).

L'institut Sustainable engineering Systems institute (SeSi) s'est récemment doté d'une ligne de montage robotisée SmartFactory de l'entreprise Festo permettant d'assembler des pièces. Une telle chaîne pourrait, par exemple, être utilisée pour assembler les éléments d'un Smartphone.

Nous désirons utiliser cette ligne d'assemblage dans le cadre d'expositions, par exemple les journées "porte ouverte", pour d'une part présenter les compétences de l'école dans le domaine de l'Industrie 4.0 et d'autre part, pour attirer des étudiants dans notre école.

La réalisation de ce dernier objectif suppose que nos démonstrations attirent les jeunes gens en y intégrant un aspect ludique.

Nous proposons donc d'utiliser cette chaîne d'assemblage pour monter des structures en LEGO, qui auront au préalable été définies par l'utilisateur.

Extrait du rapport du projet LEGO-I [1]

1.1 But du projet

Le but de ce projet était d'améliorer le programme créé durant le projet LEGO-I et la suite qui s'est déroulée durant l'été 2017.

Les limitations du projet LEGO-I au début de ce projet sont discutées dans la section[Sec. 2.3].

À terme, on veut diminuer cette liste de limitations à fin que le programme puisse être utilisé sans avoir besoin de connaissances informatiques préalables.

1.2 Structure du projet

Ce rapport de projet est structuré de la manière suivante :

- Etat initial du programme
 - Workflow initial du programme
 - Programme de conversion
 - Limitations initiales
- Centrage des modèles
 - Analyse : On analyse les différents moyens de centrer les modèles et laquelle nous allons implémenter.
 - Implémentation : On discute de l'implémentation de la méthode choisi pour le centrage des modèles.
 - Conclusion : On discute des résultats et des problèmes possibles de l'implémentation choisie.
- Amélioration du tri :
 - Analyse : On analyse les différents moyens d'améliorer le tri et la(les) quelle(s) nous allons implémenter.

- Implémentation : On discute de l'implémentation de(s) méthode(s) de tri.
- Conclusion : On discute des résultats de(s) méthode(s) et des tests pour choisir la meilleure méthode.
- Interface graphique
 - Analyse : On analyse l'ancienne interface graphique en vue de trouver les défauts
 - Implémentation : On discute les améliorations apportées à l'interface
 - Conclusion : On discute des avantages de la nouvelle interface
- Vérification de la faisabilité des modèles :
 - Analyse : On analyse les différentes parties d'un modèle qu'il faut vérifier
 - Implémentation : On discute des parties qui ont été implémentées
 - Conclusion : On discute des parties implémentées et des parties qu'il faudrait encore implémentées

2 État initial du programme

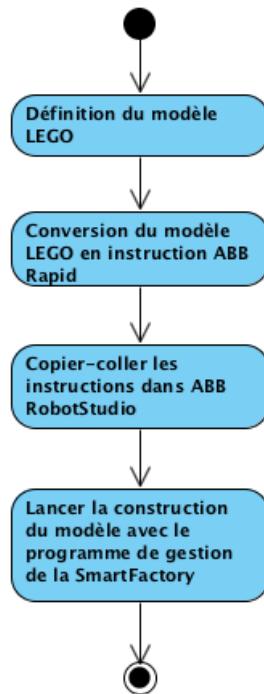
L'état initial du programme est composé du workflow ainsi que des limitations et ceux-ci définissent les éléments qui doivent être améliorés durant ce projet.

2.1 Workflow initial du programme

Le workflow initial[Fig. 1] du système se comporte de 4 étapes séparées, ces étapes sont dans l'ordre suivant :

1. Définition du modèle LEGO en utilisant le programme LEGO Digital Designer
2. Conversion du modèle LEGO dans le format reconnu par le robot ABB, en utilisant le programme de conversion
3. Copier-Coller le contenu du fichier créé par le programme de conversion dans le programme ABB RobotStudio
4. Lancer la construction du modèle à travers le programme de gestion de la SmartFactory fourni par Festo

FIGURE 1 – Workflow initial du programme

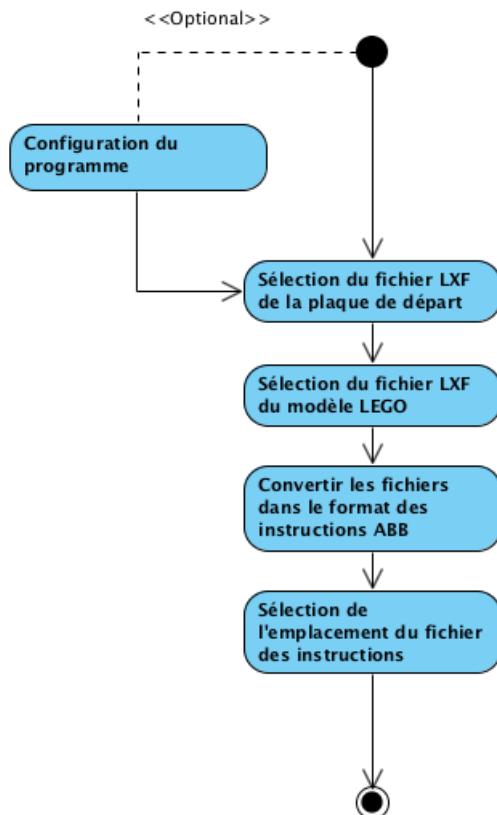


2.2 Programme de conversion

Le programme de conversion comporte en soit quelques étapes nécessaires au bon fonctionnement de celui-ci, visible dans la figure ci-dessous[Fig. 2], ces étapes étant :

1. Configuration du programme avec les fichiers contenant les configurations générales, les valeurs de rotations et le dictionnaire des pièces
2. Sélection du fichier LEGO Exchange Format (LXF) contenant la platine de base
3. Sélection du fichier LXF contenant le modèle LEGO
4. Sélection de l'emplacement du fichier de sortie
5. Convertir les fichiers dans le format du robot

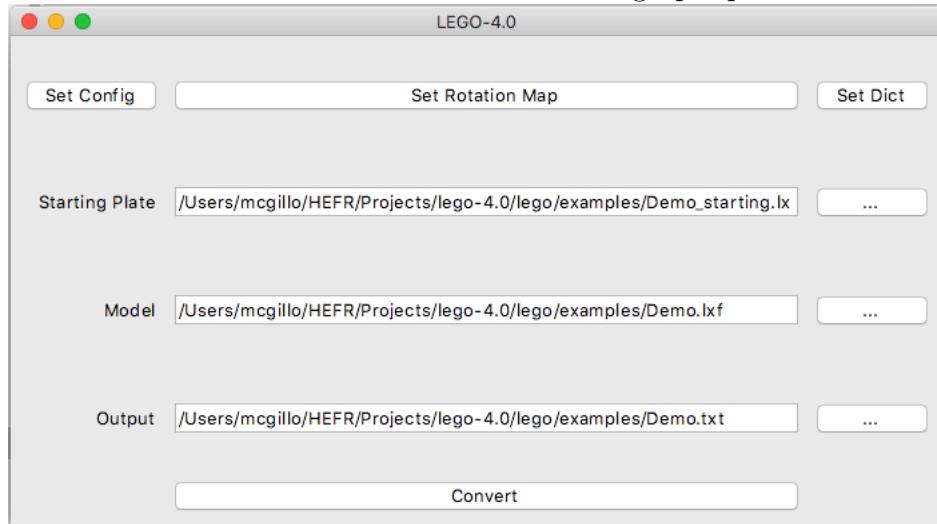
FIGURE 2 – Déroulement initial du programme



2.2.1 Interface graphique

L'interface graphique[Fig. 11] créée durant la suite du projet LEGO-I[1] a été développée pour des utilisateurs ayant des connaissances informatiques plus approfondies que le public ciblé par les projets LEGO.

FIGURE 3 – Ancienne interface graphique



2.3 Limitations initiales

Les limitations initiales du programme sont :

1. Rotations ne sont pas calculées, mais sont remplacées en fonction de la matrice de rotation créer par LDD avec des valeurs mesurées empiriquement sur la SmartFactory, ceci étant une limitation assé contraignante si nous voulons pouvoir aussi faire des rotations non régulières, par exemple pour du LEGO Technic où les angles ne sont pas nécessairement connus en avance.
2. Après une rotation, la pièce est décalée de son origine, ce qui pour le moment est corrigé avec un fichier contenant les décalages associés avec chaque rotation actuellement possible.
3. L'ordre de construction n'est pas stable comme visible avec le modèle du vaisseau spatial.
4. La platine de départ doit encore être créée manuellement par une personne et qui peut encore contenir des erreurs humaines.
5. Le programme ne prend pas encore en compte les pièces lissent qui nécessiteront un changement de pince/support/gripper
6. La taille maximale d'un modèle est limitée par l'enceinte du bras robotique ABB
7. Ne peut pas poser une pièce par dessous, exemple suspendre une brique à un pont
8. Les modèles ne peuvent pas contenir des pièces autres que ceux définis dans le dictionnaire, pour le moment celui-ci contient 13 pièces différentes
9. Construction linéaire, le système construit une pièce à la fois, de bas en haut
10. Les modèles doivent être construits correctement dans LDD, centré sur l'origine du plan virtuel

11. Le système n'est pas automatisé donc besoin d'effectuer chaque étape manuellement, le système n'est pas capable d'interagir avec le robot sans l'intervention manuelle
12. Seulement capable de construire un modèle avec une seule platine de départ donc limité par la quantité de pièces pouvant être sur une platine

2.4 Conclusion

Les limitations et autres aspects du programme initial qu'on a abordé dans ce projet comme déjà mentionné dans l'introduction étaient :

- Centrages des modèles, limitation numéro 10
- L'amélioration du tri, limitation numéro 3
- La vérification de faisabilité des modèles, limitations numéros 5, 6, 7, 8
- Amélioration de l'interface graphique

3 Centrage des modèles

Le centrage des modèles est nécessaire, car si l'utilisateur construit son modèle dans LEGO Digital Designer et ne fais pas attention où il le place dans le monde virtuel de LEGO Digital Designer, le modèle pourrait ne pas être construit correctement, soit il manquera une partie ou le tout, car le bras robotique va tenter de poser les pièces en dehors de la plaque de construction.

On avait deux possibilités pour effectuer ce centrage, ceux-ci sont analysés dans la section suivante et on discute ensuite l'implémentation de l'option choisie.

3.1 Analyse

On a analysé les deux possibilités qu'on a envisagées et les ont ensuite comparé pour décider laquelle nous allons implémenter.

Les deux possibilités pour effectuer le centrage des modèles sont :

1. Calcul du barycentre du modèle suivi d'une translation de centrage
2. Un modèle de départ contenant la plaque de centrage au centre

3.1.1 Calcul du barycentre

Le calcul du barycentre permettrait d'effectuer une translation du modèle vers le centre du monde virtuel LEGO.

Pour calculer le barycentre, on devrait trouver les pièces se trouvant aux extrêmes du modèle. Les pièces trouvées seront ensuite utilisées pour calculer le barycentre en calculant les moyennes des positions dans les axes X et Y . Une fois les moyennes calculées on a notre barycentre, on soustrairait les positions des pièces par la position du barycentre pour centrer les pièces dans le monde virtuel LDD.

3.1.2 Modèle avec plaque

Le centrage avec un modèle possédant une plaque au centre du monde virtuel LEGO Digital Designer permettrait de montrer à l'utilisateur où construire son modèle et la taille possible de celui-ci.

Le programme devrait ignorer la plaque de centrage durant la lecture du fichier du modèle. Il devrait aussi changer la hauteur des pièces du modèle puisque celui-ci aura une hauteur influencée par la plaque.

On devrait soustraire l'épaisseur de la plaque à la coordonnée Z des pièces pour corriger la différence de hauteur qui arrive avec la suppression de la plaque.

3.1.3 Comparaison

Chacune des deux possibilités possédait des avantages et des désavantages qui ont influencé la décision de laquelle a été implémenté.

Les avantages du barycentre étaient :

- Compatible pour des modèles de toute taille
- Ne nécessite pas un fichier LXF spécifique

Les désavantages étaient :

- Les limites de taille du modèle ne sont pas visibles à l'utilisateur

Les avantages du modèle avec plaque étaient :

- Le modèle sera toujours centré
- L'utilisateur connaît la limite de taille

Les désavantages étaient :

- L'utilisateur est obligé d'utiliser un fichier LXF spécifique

On a décidé d'implémenter le choix du modèle avec une plaque indiquant la taille et position de construction. Ceci, car on a jugé que le centrage avec plaque sera plus facile pour l'utilisateur à utiliser plutôt que de lui dire que son modèle est trop grand quand il essaie de générer les instructions pour le bras robotique.

3.2 Implémentation

On a implémenté la possibilité de créer un modèle dans un fichier prédéfini comportant juste une plaque LEGO 8x16, indiquant la position et les limites de construction.

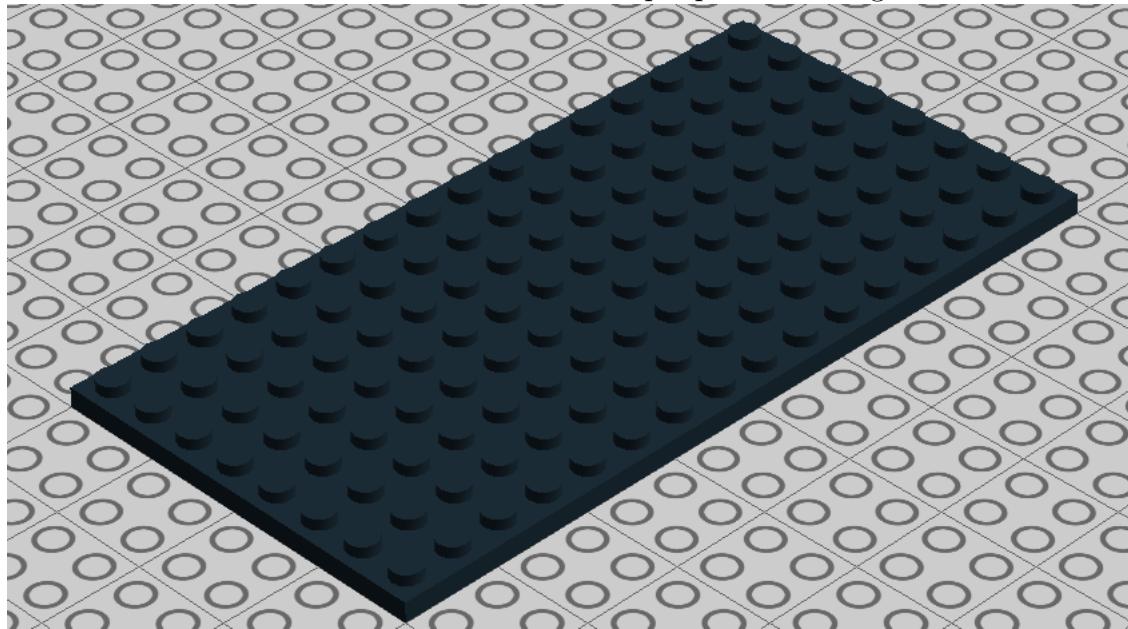
Les étapes de l'implémentation sont :

1. Création du fichier LXF avec la plaque
2. Ignorer plaque de centrage durant la lecture du fichier du modèle
3. Soustraction de l'épaisseur de la plaque à toutes les pièces du modèle

3.2.1 Fichier LXF avec la plaque

Le fichier LXF avec la plaque de centrage, visible dans la figure ci-dessous[Fig. 4] a été créé en utilisant LDD en mettant une plaque LEGO de la même taille que la plaque de construction, 16×8 têtons, au centre du monde virtuel LEGO.

FIGURE 4 – Fichier LXF avec plaque de centrage



3.2.2 Code source

La logique nécessaire pour cette implémentation, visible ci-dessous [Fig. 5], a d'abord été créée dans la version du programme émanant de LEGO-I[1]

FIGURE 5 – Implémentation originale du centrage avec plaque

```
# File lego.py / Lego.convert() / ligne 95 / commit 2396860cf6
if self.is_model:
    new_z = round(new_z - self.cfg['offset']['z'], 2)

# File lego.py / Lego.parse() / ligne 114 / commit 2396860cf6
if int(part_id) != self.cfg['center']['id']:
    print(part_id)
    print(self.cfg['center']['id'])
    matrix = self.convert(
        list(map(float, matrix)),
        self.cfg['lxfml']['translation'],
        part_id,
        material_id
    )
    bricks.append(matrix)

# File lego.py / Lego.construct() / ligne 135 / commit 2396860cf6
self.is_model = True
end = order_map(self.parse(build), False)
```

L'implémentation a changé durant l'intégration du programme du projet LEGO-2b, l'implémentation actuelle est visible dans l'extrait du code source ci-dessous[Fig. 6].

FIGURE 6 – Implémentation actuelle du centrage avec plaque

```
# File model.py / Model.remove_plate() / ligne 27 / commit e5028c4e333
def remove_plate(self):
    pieces = self._model_document.get_pieces()
    plate = None
    for piece in pieces:
        if piece.get_design_id() ==
            cfg.config['ldd']['special_pieces']['centering_plate']['design_id']:
            plate = piece
    if plate is not None:
        pieces.remove(plate)
        offset_x, offset_y, offset_z =
            cfg.config['ldd']['special_pieces']['centering_plate']['offset']
        self._translate(pieces, offset_x, offset_y, offset_z)
```

3.3 Conclusion

L'implémentation du centrage choisi était assez triviale à mettre en place, mais fournit un guide visuel à l'utilisateur. On a aussi essayé de mettre la plaque sous le plan du monde virtuel, mais sans succès.

Les autres problèmes possibles avec cette implémentation sont le dépassement de la zone défini par cette plaque, ce problème est discuté dans la section vérification de faisabilité[Sec. 6].

4 Amélioration du tri

L'amélioration du tri des pièces est nécessaire, car on a remarqué à la construction du modèle de la navette spatiale à la fin du projet LEGO-I[1] que le tri n'est pas encore assez robuste pour empêcher des problèmes à la construction.

La raison que le tri des pièces doit être perfectionné est que les pièces LEGO nécessitent une très grande précision. Le bras robotique est capable d'une très grande précision, mais puisque la pince n'est pas autant précise dû au fait qu'elle tient les pièces par seulement deux tétons et que la navette avec les pièces nécessaires à la construction n'est pas toujours exactement au même endroit.

La précision n'est pas autant un problème pour la construction LEGO avec les mains puisque les humains sont capables de souplesse en assemblant les pièces de LEGO.

4.1 Analyse

Un des problèmes probables du tri actuel est que le tri ne prend pas en compte la taille des pièces. Un autre problème pourrait être que le tri utilise l'endroit de saisie (ref_knob) de la pièce et pas le coin inférieur gauche qui est le point le plus proche de l'origine de la plaque.

On ordonne aussi les pièces d'abord par Y, suivi par la diagonale définie par l'addition de X et Y et en dernier par Z.

On a commencé par regarder en détail l'ordre des pièces pour le modèle de la navette spatiale

Dans cette vidéo, Vidéo de la navette spatiale, nous pouvons voir après la seconde 5 qu'une pièce est posée entre deux pièces, la probabilité existe donc que le bras robotique ne pourra pas poser la pièce.

On a donc décidé de permettre le tri par le ref_knob, le barycentre, le coin supérieur droite et le coin inférieur gauche de la pièce. On a aussi voulu voir si l'ordre des opérations de tri changé les résultats, donc on peut avoir les ordres :

- **DXZ** : Commencer par la diagonale, suivi par la coordonnée X et terminer avec la coordonnée Z.
- **DYZ** : Commencer par la diagonale, suivi par la coordonnée Y et terminer avec la coordonnée Z.
- **XDZ** : Commencer par la coordonnée X, suivi par la diagonale et terminer avec la coordonnée Z.
- **YDZ** : Commencer par la coordonnée Y, suivi par la diagonale et terminer avec la coordonnée Z.

4.2 Implémentation

L'implémentation a consisté en première partie de la réécriture de la méthode de tri développé durant le projet LEGO-I[1] avec comme but d'être plus modulaire et flexible au changement de l'ordre des opérations de tri ainsi que de la position interne de la pièce.

Pour cela le tri est maintenant composé de 5 fichiers, ceux-ci étant :

- **ordering.py** : Ce fichier comporte une seule fonction qui prend en entrée la liste des pièces et appelle la fonction configurée dans le fichier de configuration, cette fonction est visible ci-dessous [Fig. 7].
- **ref_knob.py** : Ce fichier comporte 5 fonctions, une pour appliquer l'ordre des opérations défini dans le fichier de configuration, et les 4 autres sont les différentes opérations. Le tri s'effectue sur la position ref_knob de la pièce et ce fichier est visible ci-dessous [Fig. 8, 9]
- **upper_right.py** : Ce fichier est quasi identique que le fichier ref_knob.py, à l'exception de la position sur lequel le tri s'effectue, la position étant le coin supérieur droit de la pièce.
- **lower_left.py** : Ce fichier est quasi identique que le fichier ref_knob.py, à l'exception de la position sur lequel le tri s'effectue, la position étant le coin inférieur gauche de la pièce.
- **barycenter.py** : Ce fichier est quasi identique que le fichier ref_knob.py, à l'exception de la position sur lequel le tri s'effectue, la position étant le barycentre de la pièce.

FIGURE 7 – Fonction principale du tri des pièces

```
# File ordering.py
def order_pieces(pieces):
    """
    Order the list of pieces of a model, depending on the configured
    ordering function and ordering
    method
    """
    fn_ordering = cfg.config['algorithm']['ordering']['fn']
    order_method = cfg.config['algorithm']['ordering']['method']
    if order_method == 'ref_knob':
        return ref_knob.order(pieces, fn_ordering)
    elif order_method == 'upper_right':
        return upper_right.order(pieces, fn_ordering)
    elif order_method == 'lower_left':
        return lower_left.order(pieces, fn_ordering)
    elif order_method == 'barycenter':
        return barycenter.order(pieces, fn_ordering)
```

FIGURE 8 – Fichier ref_knob.py partie 1

```
# File ref_knob.py
def __order_z(pieces):
    """
    Order the map according to z on the ref_knob position
    """
    def order_fn(piece):
        return piece.get_zup_coordinates()[2]

    return sorted(pieces, key=order_fn)

def __order_diagonal(pieces):
    """
    Order the map diagonally using the sum of x and y on the ref_knob
    position
    """
    def order_fn(piece):
        coordinates = piece.get_zup_coordinates()
        x = coordinates[0]
        y = coordinates[1]
        return x + y

    return sorted(pieces, key=order_fn)

def __order_y(pieces):
    """
    Order the map according to y on the ref_knob position
    """
    def order_fn(piece):
        return piece.get_zup_coordinates()[1]

    return sorted(pieces, key=order_fn)
```

FIGURE 9 – Fichier ref_knob.py, partie 2

```

# File ref_knob.py
def __order_x(pieces):
    """
    Order the map according to x on the upper_right position
    """
    def order_fn(piece):
        return piece.get_zup_coordinates()[0]

    return sorted(pieces, key=order_fn)

FN_ORDERING = {'ydz': [__order_y, __order_diagonal, __order_z],
               'dxz': [__order_diagonal, __order_x, __order_z],
               'xdz': [__order_x, __order_diagonal, __order_z],
               'dyz': [__order_diagonal, __order_y, __order_z]}

def order(pieces, fn_ordering='ydz'):
    """
    Order pieces using the ref_knob position of a piece

    Positional Arguments:
    pieces -- list of all the pieces to be sorted

    Keyword Arguments:
    fn_ordering -- the order of the ordering functions (default='ydz')
    """
    for order_fn in FN_ORDERING[fn_ordering]:
        pieces = order_fn(pieces)
    return pieces

```

Cette implémentation permet de facilement rajouter des nouvelles positions sur lesquelles on peut effectuer le tri ainsi que de changer l'ordre des opérations.

On a ensuite créé des tests, ils testent les 4 positions ainsi que les 4 ordres de tri sur 29 modèles avec des cas qu'on a jugés pourraient avoir des problèmes.

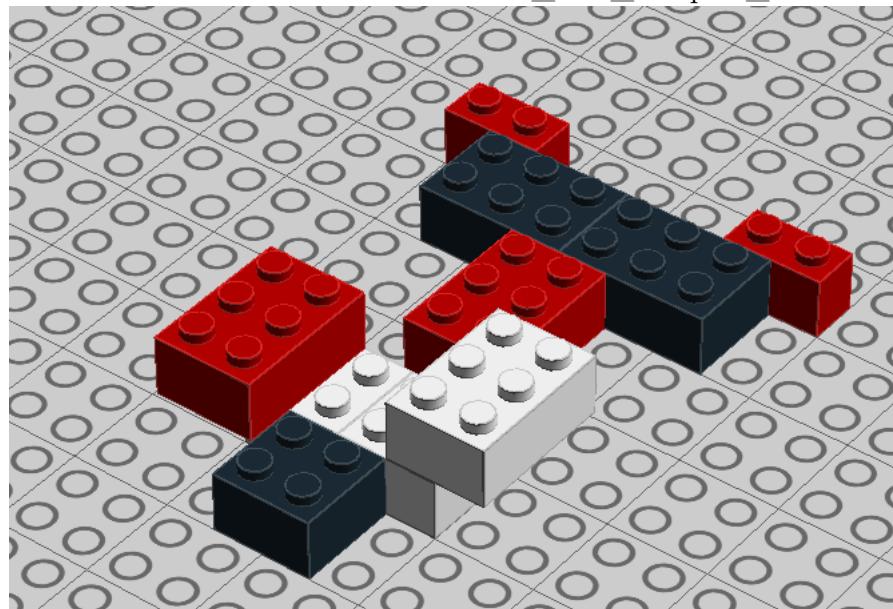
4.3 Conclusion

Les résultats des tests créer dans l'implémentation, aide à décider quelle position et ordre à utiliser de base, mais ceux-là peuvent être changé dans le fichier de configuration globale du programme.

- L'ordre de tri $X D Z$ est l'ordre qui fonctionne le mieux avec 109 tests sur 116 qui passent, les 7 tests qui ne passent pas sont sur les cas les plus complexes, ‘test_case_complex_2’(visible ci-dessous [Fig. 10]), ‘test_case_complex_3’ et ‘test_case_complex_4’.

- La position barycentre est la position qui échoue dans les 3 cas, et la position ref_knob n'échoue jamais.
- L'ordre de tri YDZ réussit 103 tests sur 116, les 13 tests qui ne passent pas sont sur les cas les plus complexes, ‘test_case_complex_2’(visible ci-dessous [Fig. 10]), ‘test_case_complex_3’ et ‘test_case_complex_4’ et aussi sur le modèle de la navette spatiale mais seulement avec la position lower_left.
 - Aucune position n'échoue jamais.
 - L'ordre de tri DXZ réussit 96 tests sur 116, les 20 tests qui ne passent pas sont sur les cas les plus complexes, ‘test_case_complex_2’(visible ci-dessous [Fig. 10]), ‘test_case_complex_3’ et ‘test_case_complex_5’, ainsi que d'autre.
 - Aucune position n'échoue jamais.
 - L'ordre de tri DYZ réussit 65 tests sur 116, les 51 tests qui ne passent pas sont répartis sur une majorité des cas des tests.
 - Aucune position n'échoue jamais.

FIGURE 10 – Cas de test ‘test_case_complex_2’



On peut donc voir que l'ordre qui est adapté pour le plus de cas est l'ordre XDZ ainsi que la position ref_knob mais que dans certaines situations, un autre ordre ou position est plus efficace.

Un moyen d'améliorer encore plus le tri des pièces serait de faire un tri qui analyse le modèle pour trouver l'ordre le plus propice plutôt qu'un tri encore assez simple en soi.

5 Interface graphique

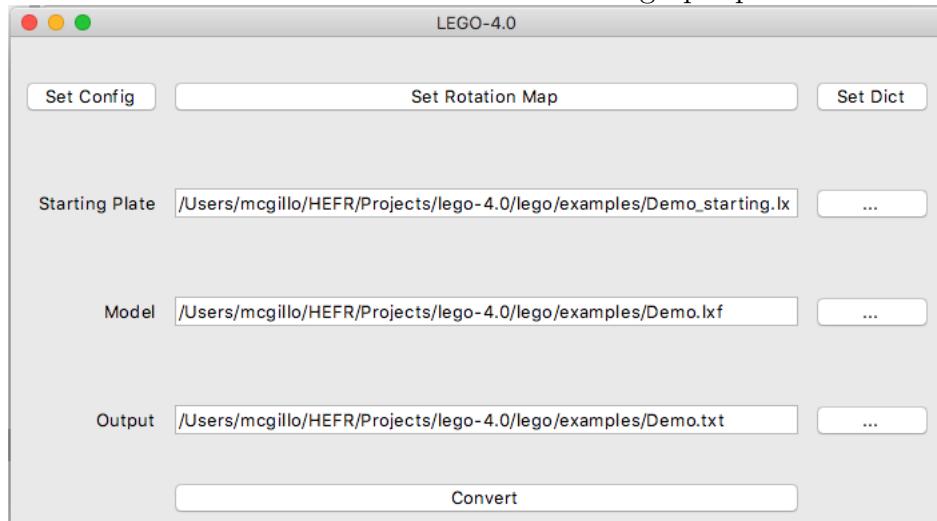
L’interface graphique étant une partie essentielle de n’importe quel programme destiné à être utilisé par des utilisateurs n’ayant pas de connaissances informatiques très avancées, celui-ci doit être le plus simple possible.

Pour cette raison, on a décidé d’améliorer cette interface, l’interface doit être capable de définir le fichier du modèle et les fichiers de configurations d’une manière intuitive et simple.

5.1 Analyse

On va commencer par analyser l’ancienne interface graphique, vu dans la section de l’état initial du programme[Sec. 2], pour identifier les problèmes avec celle-ci.

FIGURE 11 – Ancienne interface graphique



Dans l’ancienne interface graphique plusieurs éléments portaient à confusion et devaient être enlevés ou changés, ces éléments étaient :

- Les boutons[Fig. 12] utilisés pour définir les fichiers de configuration en haut de l’interface.
- Les champs textuels[Fig. 13] affichant les chemins des fichiers ne sont pas nécessaires pour un utilisateur sans connaissances informatiques.
- L’utilisateur ne sait pas quels fichiers il doit fournir et lesquelles seront fournies par le programme.
- Les boutons à côté des champs textuels n’indiquent pas à quoi ils servent.
- Le bouton ‘Convert’ n’a pas de feed-back, on ne sait donc pas quand la conversion est finie.

FIGURE 12 – Boutons de configuration dans l’ancienne interface

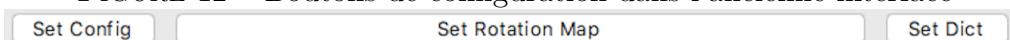


FIGURE 13 – Champs textuels avec chemin des fichiers



```
/Users/mcgillo/HEFR/Projects/lego-4.0/lego/examples/Demo_starting.lx
```

```
/Users/mcgillo/HEFR/Projects/lego-4.0/lego/examples/Demo.lxf
```

```
/Users/mcgillo/HEFR/Projects/lego-4.0/lego/examples/Demo.txt
```

Il existe aussi des éléments qu'on a dû ajouter pour faciliter l'utilisation, ces éléments sont :

- Créer un nouveau modèle avec le fichier LXF avec la plaque de construction [Sec. 3] et le charger en préparation pour la conversion.
- Charger un modèle déjà créé préalablement et ouvrir LDD avec celui-ci.
- Charger un modèle déjà créé préalablement sans ouvrir LDD.
- Convertir le modèle et indiquer à l'utilisateur que la conversion a été achevée.
- Changer la méthode de placement des pièces sur la plaque de départ.
- Indiquer à l'utilisateur des erreurs s'il en arrive.

On a donc simplifié l'interface en rendant les boutons plus clairs, déplacé les méthodes de configuration pour ne pas confondre l'utilisateur et enlever les champs textuels.

La nouvelle interface comporte donc seulement 5 boutons, un pour chacun des cinq premiers éléments dans la liste précédente et fournir un menu avec les options pour configurer le programme.

L'implémentation de cette nouvelle interface est discutée dans la prochaine section.

5.2 Implémentation

On affiche les boutons sur deux lignes, la première avec les boutons pour un nouveau modèle, charger un modèle et ouvrir LDD, charger un modèle, et changer la méthode de placement. La deuxième ligne consiste seulement du bouton pour convertir le modèle.

Ces actions sont aussi disponible dans un menu nommé ‘Model’, dans un menu nommé ‘Configure’ on a mis à disposition les actions pour configurer le programme.

5.2.1 Bouton : Nouveau modèle

Le bouton pour un nouveau modèle sera nommé ‘New Model’, ouvre LDD avec le fichier LXF de base avec la plaque de construction et définit celui-ci comme le modèle à convertir.

Pour s’assurer que le fichier de base soit toujours disponible, on crée une copie de ce fichier avec le nom "Model_<jour-mois-année_heure:minute>.lxf" dans le répertoire personnel de l’utilisateur courant et définit celui-ci comme modèle à convertir.

On ouvre ensuite ce fichier avec l’application par défaut, qui normalement devrait être LDD.

5.2.2 Bouton : Charger un modèle et ouvrir LDD

Le bouton pour charger un modèle et ouvrir avec LDD est nommé ‘Load Model and open LDD’, fournit un dialogue pour choisir le modèle, l’ouvre avec LDD et le définit celui-ci comme le modèle à convertir.

Le dialogue commence dans le répertoire personnel de l’utilisateur courant, et permet l’utilisateur de choisir soit un fichier LXF ou LXFML et celui-ci est ensuite défini comme fichier à ouvrir avec LDD, selon la même méthode que le bouton nouveau modèle et comme modèle à convertir.

5.2.3 Bouton : Charger un modèle

Le bouton pour charger un modèle est nommé ‘Load Model’ et procède de la même manière que le bouton charger un modèle et ouvrir LDD mais sans la partie pour ouvrir LDD.

5.2.4 Bouton : Changer la méthode de placement

Le bouton pour changer la méthode de placement est un bouton avec texte variable, le texte affiché indique la méthode choisie.

Pour changer la méthode, l’utilisateur peut cliquer sur le bouton pour traverser les différentes méthodes, la méthode choisie est ensuite passée en paramètre à la méthode de conversion.

5.2.5 Bouton : Convertir

Le bouton pour convertir est nommé ‘Convert’, et initialise le contrôleur avec les fichiers de configuration et lance la méthode de conversion.

Dès que la conversion est achevée, on ouvre les nouveaux fichiers générés, ceux-ci étant le premier fichier des plaques de départ et le premier fichier avec les instructions d’assemblage.

5.2.6 Menu : Model

Le menu ‘Model’ contient des éléments qui effectuent les mêmes actions que les boutons décrivent ci-dessus, sauf pour le changement de méthode de placement.

5.2.7 Menu : Configuration

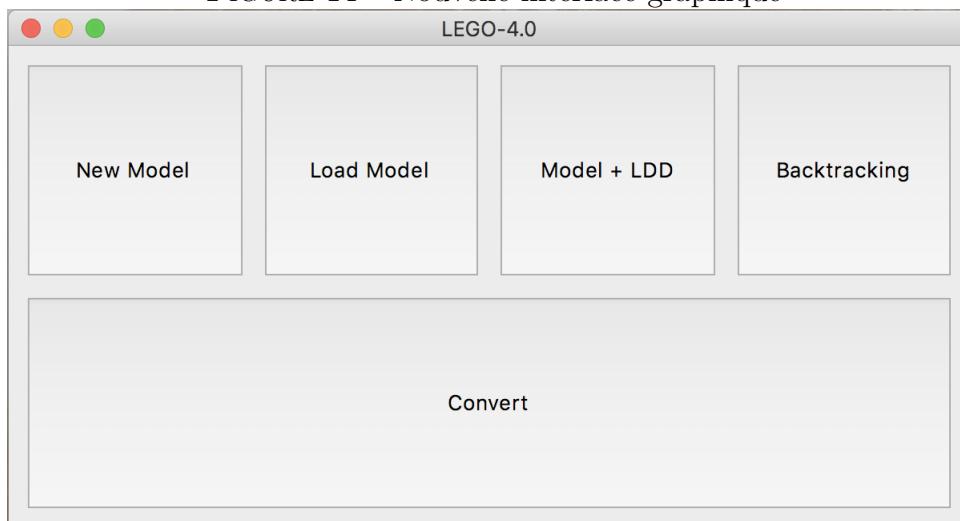
Le menu de configuration est nommé ‘Configure’ et contient des éléments associés à des actions pour définir les différents fichiers de configuration nécessaire au bon fonctionnement du programme de conversion.

Les éléments qui sont représentés dans ce menu :

- ‘**Set Config file**’ : Définis le fichier de configuration globale du programme en utilisant un dialogue avec comme répertoire initial le même répertoire que le fichier de configuration fourni avec l’application.
- ‘**Set Dict file**’ : Définis le fichier de dictionnaire des pièces du programme en utilisant un dialogue avec comme répertoire initial le même répertoire que le fichier du dictionnaire fourni avec l’application.
- ‘**Set Rotation file**’ : Définis le fichier avec les valeurs des quaternions du programme en utilisant un dialogue avec comme répertoire initial le même répertoire que le fichier des quaternions fourni avec l’application.
- ‘**Set Template file**’ : Définis le fichier du chablon pour les plaques de départ du programme en utilisant un dialogue avec comme répertoire initial le même répertoire que le fichier chablon fourni avec l’application.

La nouvelle interface ressemble finalement à la figure[Fig. 14] visible ci-dessous.

FIGURE 14 – Nouvelle interface graphique



Le menu ‘Model’[Fig. 15] et ‘Configure’[Fig. 16] sont visibles ci-dessous.

FIGURE 15 – Menu Model de la nouvelle interface

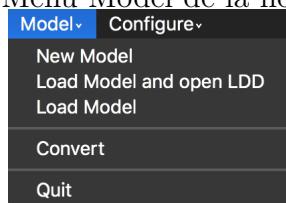
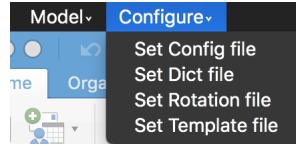


FIGURE 16 – Menu Configure de la nouvelle interface



5.3 Conclusion

La nouvelle interface simplifie grandement l'utilisation du programme de conversion des modèles LDD dans des instructions ABB Rapid pour contrôler le bras robotique.

L'interface abstrait aussi l'utilisation de la plaque de centrage qui pourrait confondre l'utilisateur.

L'interface permet aussi d'afficher des exceptions qui sont levées durant l'exécution du programme de conversion, ce qui permet à l'utilisateur de savoir qu'une exception a été levé, la liste détaillée des exceptions est listée dans l'annexe [A.6] ainsi que dans le répertoire Git du projet.

6 Vérification de faisabilité

La vérification de faisabilité est un aspect important pour ce projet, car sans cette vérification, c'est possible de convertir un modèle LDD que la SmartFactory n'est pas capable de construire d'une manière automatisée.

6.1 Analyse

La vérification est divisée en plusieurs parties :

- Hauteur du modèle
- Dépassement de la plaque de construction
- Pièces disponible et utilisable
- Possibilité de poser une pièce

6.1.1 Hauteur du modèle

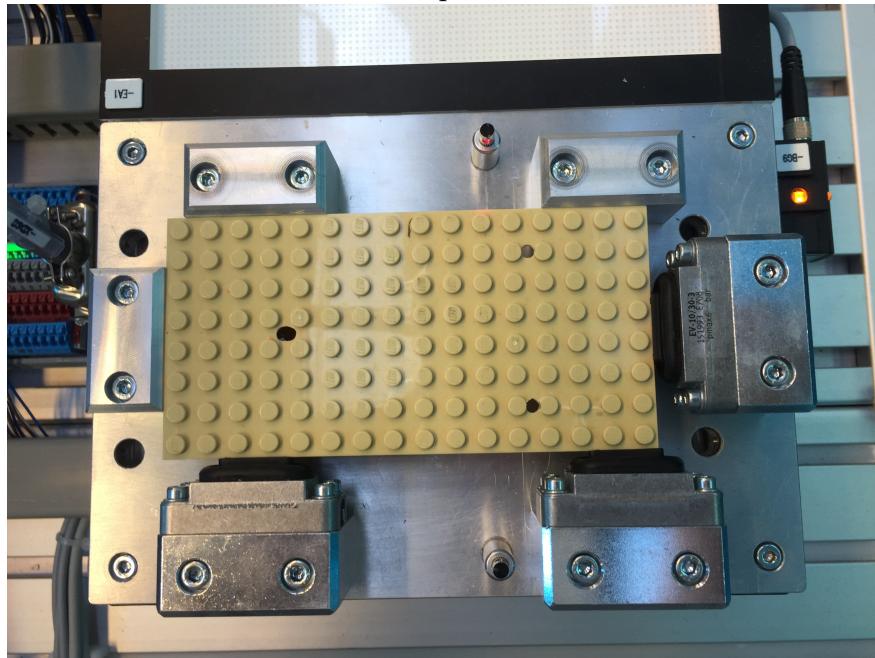
La hauteur du modèle doit être vérifié, car si le modèle est trop haut, le modèle ne pourra pas sortir de l'enceinte du bras robotique sur les tapis roulants de la SmartFactory. Le fait de ne pas pouvoir sortir de l'enceinte limiterait l'aspect d'automatisation du processus d'assemblage, car on devrait ouvrir l'enceinte du robot pour sortir le modèle, ce qui interromprait la chaîne d'assemblage.

L'enceinte permet des modèles avec 9 niveaux de briques (9.6mm) de haut de sortir.

6.1.2 Dépassement de la plaque de construction

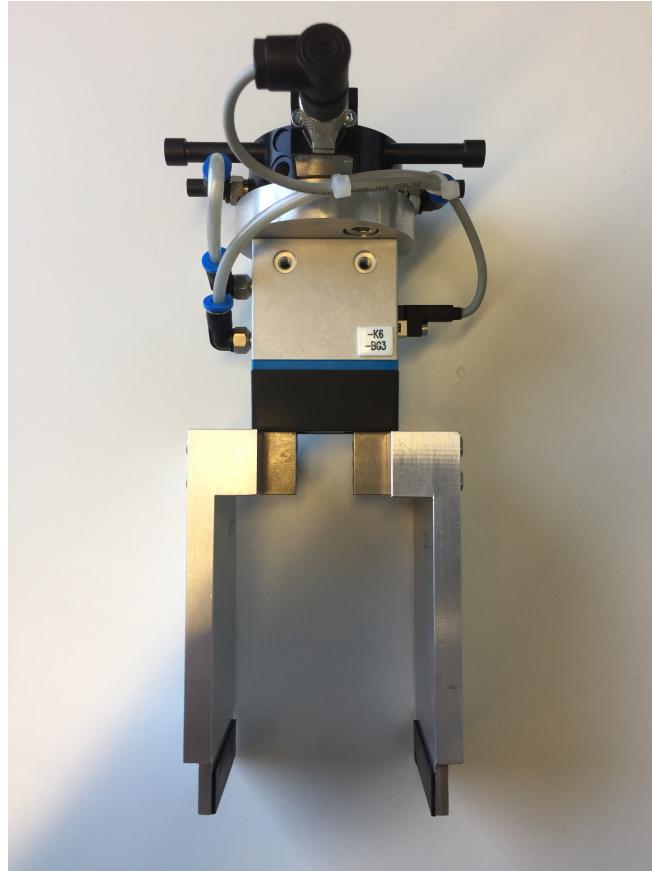
Le contrôle du dépassement de la plaque de construction[Fig. 17] est nécessaire pour être sûr que le bras robotique ne devra jamais sortir de la zone délimitée par la plaque de construction, aussi pour assurer l'équilibre du modèle durant le transport sur les tapis roulants et aussi pour que le bras robotique puisse prendre la plaque depuis la zone d'assemblage et la remettre sur la navette avec la pince pour plaques[Fig. 18].

FIGURE 17 – Plaque de construction



Solution naïve contrôler qu'aucune position ne dépasse $(16 * 8) \times (8 * 8) = 128 \times 64$, en combinant cela avec la vérification de la hauteur du modèle, les limites à contrôler sont $128 \times 64 \times (9 * 9.6) = 128 \times 64 \times 86.4$, les axes étant $X \times Y \times Z$.

FIGURE 18 – Pince pour plaques



6.1.3 Pièces disponible et utilisable

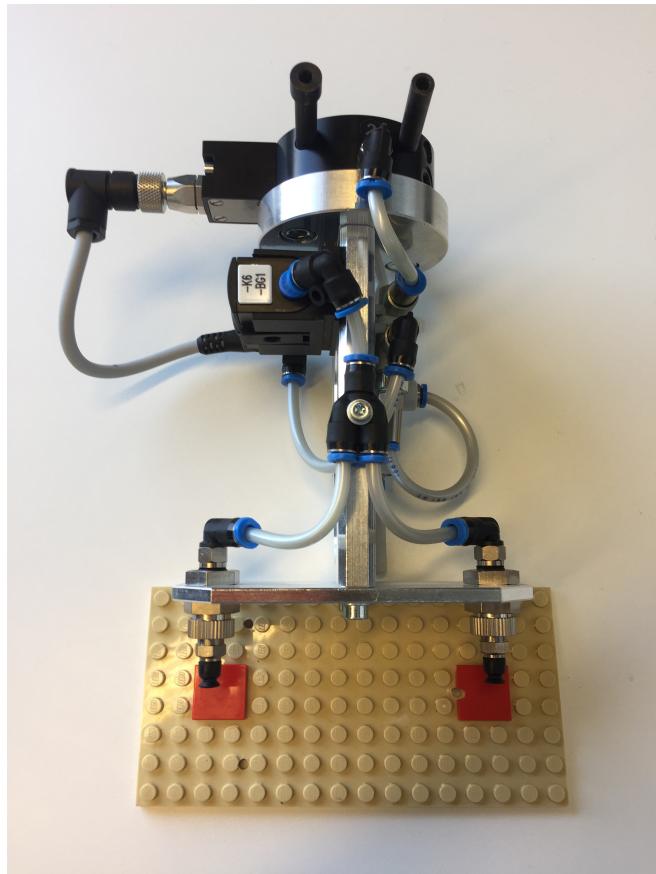
La vérification des pièces disponible et ceux utilisables par le bras robotique est probablement l'aspect le plus important de la vérification de la faisabilité puisque nous utilisons un dictionnaire des pièces avec des valeurs définissant la forme, l'endroit de saisie et le point de pivot, montré ci-dessous [19]. Le modèle ne peut donc pas utiliser une pièce qui n'est pas définie, car le programme ne saura pas comment le manipuler correctement.

FIGURE 19 – Extrait du dictionnaire

```
# Brick 2x4
3001:
    ref_knob: !!python/tuple [0,1,90]
    ldd_pivot: !!python/tuple [1,0]
    shape:
        [
            [3,3,3,3],
            [3,3,3,3]
        ]
```

Le bras robotique possède une pince avec ventouse[Fig. 20] qui est utilisé momentanément pour le transfert de plaque de construction depuis le ‘quai’ de chargement, mais peut être envisagée pour pouvoir placer des pièces lisses.

FIGURE 20 – Pince avec ventouse



Une solution envisagée est un programme permettant de modifier la base de données LDD contenant les pièces à disposition. La base de données pourra être modifiée avec un programme graphique utilisant les fichiers OBJ générés avec le script de J.Stoppani[A.5] et permettant de définir le nombre à disposition.

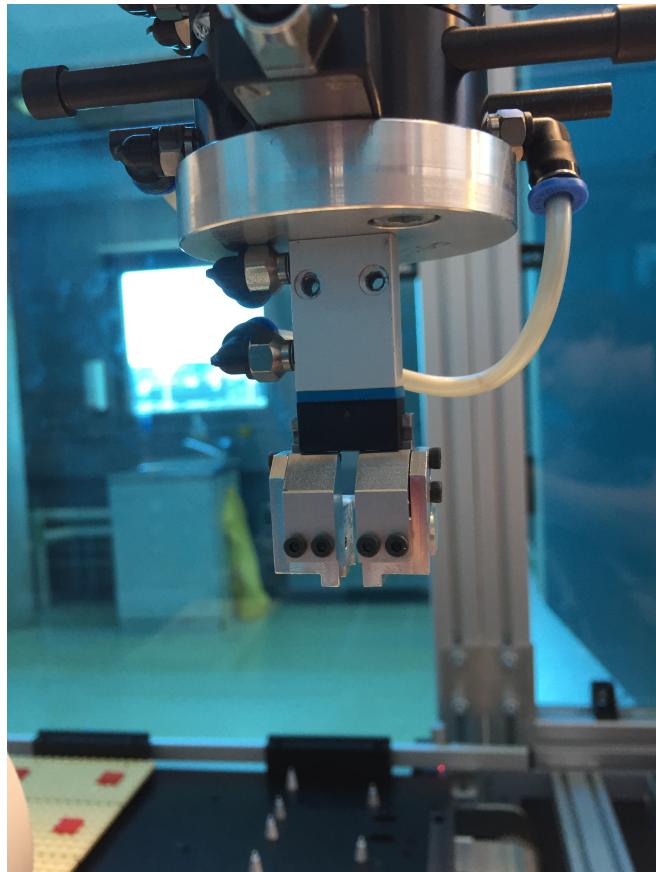
Le nombre à disposition peut ensuite être utilisé par le programme de conversion pour contrôler que l’utilisateur possède le nombre de pièces nécessaires pour le modèle en conversion.

6.1.4 Possibilité de poser une pièce

La vérification de la possibilité de poser une pièce est composée de trois parties, la première étant due aux restrictions physiques de la pince[Fig. 21] et des blocs[Fig. 22] qui tiennent la plaque de construction immobile. La deuxième partie est l’orientation des pièces, vérifier que les pièces se trouvent dans le plan X/Y avec des rotations de 0°, 90°, 180° et 270°. La troisième partie est la plus compliquée, vérifier si une pièce n’est pas posée avec une partie n’ayant aucun support dessous,

ce qui en fonction d' où la pièce est tenu et quelle partie à un support, la pièce ne tiendra pas en place.

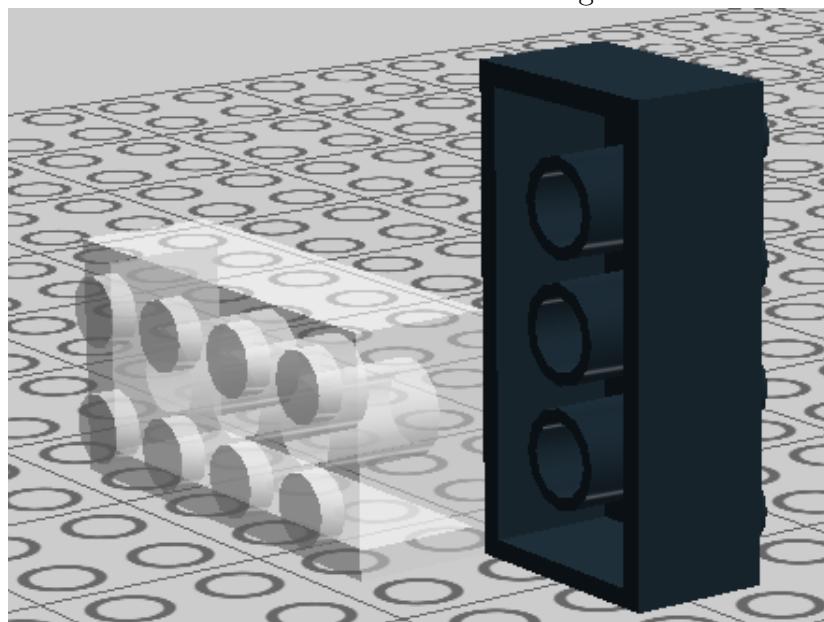
FIGURE 21 – Pince LEGO, vue de face



Pour s'assurer que toutes les pièces puissent être posé, nous devrions contrôler que le modèle ne comporte aucune pièce basse de hauteurs 3.2 dans les bords de la plaque de construction[Fig. 17].

FIGURE 22 – Blocks de retention

Pour s'assurer que toutes les pièces soient dans le plan X/Y, nous devrions contrôler la matrice de transformation de chaque pièce soit dans ce plan, un exemple de rotations non gérées par le programme est visible ci-dessous[Fig. 23].

FIGURE 23 – Rotations non gérées

Pour s'assurer que toutes les pièces ne soient pas placées dans le vide, la pièce rouge dans la figure ci-dessous[Fig. 24], ou avec trop peu de support, la pièce dans la deuxième figure ci-dessous[Fig. 25], nous avons pensé qu'une méthode pourrait être de contrôler l'intersection des pièces proches et qui sont l'un sur l'autre, si le

pourcentage d'intersection est trop faible nous pourrions savoir que cette pièce ne pourra pas être posé avec le bras robotique.

FIGURE 24 – Pièce suspendu

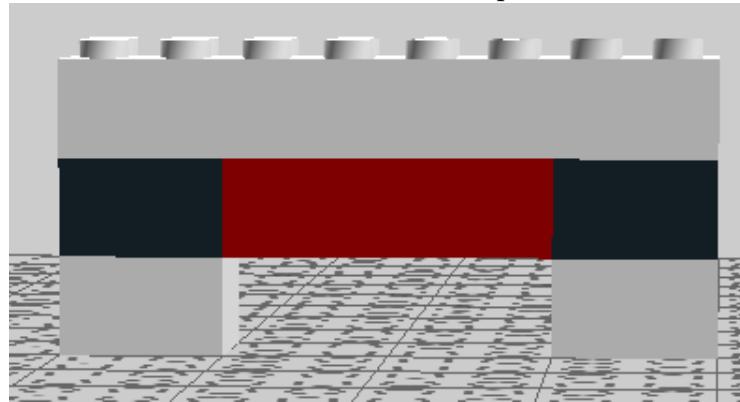
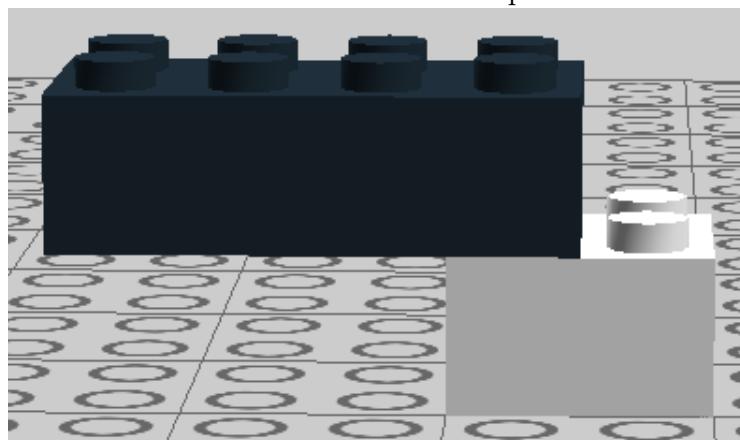


FIGURE 25 – Pièce en suspension



Nous allons implémenter seulement la vérification de la hauteur ainsi que le dépassement de la plaque de construction, qui peuvent être résolu de la même manière. La raison que nous n'implémentons que ces deux est à cause du manque de temps restant pour ce projet. Les autres éléments pourraient servir d'idée d'améliorations dans les futurs projets LEGO.

6.2 Implémentation

L'implémentation de la vérification de la hauteur et du dépassement de la plaque a été réalisée dans la même méthode, visible ci-dessous [Fig. 26]

FIGURE 26 – Implémentation de la vérification de la hauteur et dépassement de la plaque

```
# File model.py
def check_bounding_box(self):
    pieces = self._model_document.get_pieces()

    for piece in pieces:
        x_min, x_max, y_min, y_max, z =
            piece.get_bounding_box_coordinates()
        if x_min < 0 or \
           y_min < 0 or \
           z < 0 or \
           x_max > cfg.config["abb_rapid"]["bounding_box"]["x"] \
               or \
           y_max > cfg.config["abb_rapid"]["bounding_box"]["y"] \
               or \
           z > cfg.config["abb_rapid"]["bounding_box"]["z"]:
            raise InvalidModelError(
                piece.get_ref_id(), "The model does not fit within
                the borders of the construction plate")
```

Cette méthode se sert d'une méthode qui calcule le bounding box d'une pièce, visible ci-dessous [Fig. 27]

FIGURE 27 – Implémentation de la vérification de la hauteur et dépassement de la plaque

```
# File piece.py
def get_bounding_box_coordinates(self):
    x_min, y_min, z = self.get_zup_coordinates_without_ref_knob()
    scale_x, scale_y, scale_z = cfg.config["abb_rapid"]["scale"]
    x_max = x_min + self.get_length() * scale_x
    y_max = y_min + self.get_width() * scale_y
    return (x_min, x_max, y_min, y_max, z)
```

Comme déjà mentionnés, les autres éléments de la vérification de faisabilité n'ont pas été implémentés pour des raisons de manque de temps en fin de projet après les autres éléments de ce projet.

6.3 Conclusion

Cette partie du projet fut la dernière à être abordée et pour cette raison, c'est aussi la partie avec une implémentation la moins fournie.

Les parties de la vérification qu'on a analysées, mais qui n'ont pas été implémentées sont donc des possibilités pour la suite du projet, ce qui avancerait le

projet LEGO dans une bonne direction dans l'aspect de la vérification des modèles.

7 Conclusion

Les projets LEGO sont maintenant capables d'être utilisé par plus de gens qu'avant et nécessite moins de contrôle manuel, ils permettent aussi des modèles plus complexes et intéressants.

Il reste pourtant encore beaucoup de choses à améliorer, surtout des limitations qui empêchent des modèles ressemblant au modèle fourni par LEGO.

7.1 Limitations résolues

On a pu résoudre quelques limitations dans ce projet qu'on avait au début de ce projet, ces limitations étant :

- Le centrage du modèle
- Le tri est maintenant plus robuste qu'avant, mais peut toujours être amélioré
- La vérification que les dimensions d'un modèle soient adaptées à la Smart-Factory

Le projet LEGO-2b résout des autres limitations, tel que :

- Génération de la plaque de départ
- Besoins d'un fichier avec les valeurs des quaternions

Pour de plus amples informations sur les limitations, se référer au projet LEGO-2b

Les limitations qui restent sont :

- Le programme ne prend pas encore en compte les pièces lissent qui nécessiteront un changement de pince/support/gripper
- La taille maximale d'un modèle est limitée par l'enceinte du bras robotique ABB
- Ne peut pas poser une pièce par dessous, exemple suspendre une brique à un pont
- Les modèles ne peuvent pas contenir des pièces autres que ceux définis dans le dictionnaire
- Construction linéaire, le système construit une pièce à la fois, de bas en haut
- Le système n'est pas automatisé donc besoin d'effectuer chaque étape manuellement, le système n'est pas capable d'interagir avec le robot sans l'intervention manuelle

7.2 Différence avec le cahier des charges

On a réussi à accomplir la majorité de nos objectifs définies dans notre cahier des charges, le seul objectif qui n'est pas atteint est le changement des outils et méthode de saisie mais celui-ci a été remplacé par l'amélioration de l'interface graphique de l'application

7.3 Idées d'amélioration

Nous avons aussi pensé à quelques idées qui permettraient d'améliorer le projet. Quelques idées sont expliquées en dessous.

7.3.1 Serveur de conversion

Une idée serait d'avoir un serveur de conversion où des personnes pourront télécharger leurs modèles LEGO créés avec LDD et ensuite recevoir les fichiers nécessaires pour les mettre sur l'usine pour construire leurs modèles.

Une idée plus poussée serait que ce serveur soit directement relié à l'usine et que la construction se lance automatiquement dès que la conversion est achevée et que la personne puisse ensuite par la suite venir là chercher.

7.3.2 Utilisation de QR codes

Cette idée se repose sur l'idée d'un serveur de conversion, après la conversion achevée l'utilisateur reçoit un QR code qui leur permettrait d'ensuite aller à l'usine et faire scanner ce code pour faire construire leur modèle.

7.3.3 Modélisation en VR/AR

La dernière idée que nous avions est de pouvoir créer le modèle en réalité augmenté ou en réalité virtuelle et d'ensuite voir notre modèle construit en vrai dans l'usine.

Pour la modélisation en réalité augmentée, nous pourrions envisager d'utiliser un casque Microsoft Hololens[2] ou un casque Meta[3], ces deux casque étant les meilleurs disponibles sur le marché en ce moment.

Pour la modélisation en réalité virtuelle, nous pourrions envisager d'utiliser un casque Oculus Rift[4] ou un casque HTC Vive[5], ces deux casques sont déjà très utilisés par le grand public pour jouer à des jeux vidéo.

7.4 Avis personnel

Ce projet fut plus conséquent que le projet LEGO-I[[1]], ceci à cause de la complexité de certains aspects telle que l'amélioration du tri et l'adaptation pour les nouveaux éléments provenant du projet LEGO-2b.

La complexité de l'amélioration du tri provient du fait qu'il existe une infinité de modèles possible et qu'il a fallu trouver des cas qui représentent le plus grand nombre de possibilités de ces modèles. Un autre aspect du tri est que c'est très dur de choisir le meilleur ordre des pièces d'un modèle sans avoir essayé l'assemblage du modèle avec le bras robotique car les quaternions indiquant l'orientation de la pièce ne sont pas les plus simples pour ce faire une représentation de la position finale, c'est encore plus le cas avec des pièces de plus en plus grandes.

Une possibilité pour palier à cette complexité serait de pouvoir simuler les mouvements du bras robotique dans un programme qui nous permettrait de visualiser plus rapidement les modifications et aussi sans soucis d'endommager la SmartFactory.

8 Déclaration d'honneur

Je, soussigné, Owen McGill, déclare sur l'honneur que le travail rendu est le fruit d'un travail personnel. Je certifie ne pas avoir eu recours au plagiat ou à toutes autres formes de fraudes. Toutes les sources d'information utilisées et les citations d'auteur ont été clairement mentionnées.

A Annexes

Table des figures

1	Workflow initial du programme	5
2	Déroulement initial du programme	6
3	Ancienne interface graphique	7
4	Fichier LXF avec plaque de centrage	11
5	Implémentation originale du centrage avec plaque	11
6	Implémentation actuelle du centrage avec plaque	12
7	Fonction principale du tri des pièces	14
8	Fichier ref_knob.py partie 1	15
9	Fichier ref_knob.py, partie 2	16
10	Cas de test ‘test_case_complex_2’	17
11	Ancienne interface graphique	18
12	Boutons de configuration dans l’ancienne interface	18
13	Champs textuels avec chemin des fichiers	19
14	Nouvelle interface graphique	21
15	Menu Model de la nouvelle interface	21
16	Menu Configure de la nouvelle interface	22
17	Plaque de construction	24
18	Pince pour plaques	25
19	Extrait du dictionnaire	25
20	Pince avec ventouse	26
21	Pince LEGO, vue de face	27
22	Blocks de retention	28
23	Rotations non gérées	28
24	Pièce suspendu	29
25	Pièce en suspension	29
26	Implémentation de la vérification de la hauteur et dépassement de la plaque	30
27	Implémentation de la vérification de la hauteur et dépassement de la plaque	30
28	Interface graphique du programme	38
29	Interface graphique du programme	39
30	Plaque de départ	40
31	Cas de figure où une exception "InvalidModelError" est lancée . . .	41
32	Cas de figure où une exception "DictionaryValueError" est lancée .	41
33	Diagramme de classes	41

Références

- [1] Owen MCGILL. *Lego-I*. Rapp. tech. HEIA-FR, 2017.
- [2] MICROSOFT. *Microsoft Hololens*. URL : <https://www.microsoft.com/en-us/hololens> (visité le 26/06/2017).

- [3] META. *Meta 2*. URL : <https://www.metavision.com/> (visité le 26/06/2017).
- [4] OCULUS. *Oculus Rift*. URL : <https://www.oculus.com/rift/> (visité le 26/06/2017).
- [5] HTC. *HTC Vive*. URL : <https://www.vive.com/eu/> (visité le 26/06/2017).

Glossaire

ABB RobotStudio Programme de contrôle du bras robotisé ABB. 5

barycentre Centre géométrique d'un objet, la moitié de chaque axe de l'objet.
9, 10, 13, 14

bras robotique Robot ABB dans la SmartFactory de l'institut SeSi. 13, 33,
36

Haute école d'ingénierie et d'architecture de Fribourg . 3

Industrie 4.0 . 3, 36

LEGO Digital Designer Application mis à disposition par le groupe LEGO pour créer des modèles LEGO. 1, 5, 9, 36

LEGO Exchange Format Fichier comportant une image et un fichier LEGO Exchange Format Markup Language (LFXML) généré par LDD. 6

LEGO Exchange Format Markup Language Fichier XML généré par LDD.
36

modèle Modèle LEGO créé avec . 9, 10, 13, 36

navette spatiale Modèle LEGO en forme de navette spatiale, utilisé comme démo durant la journée . 13, 17

plaquette de centrage La plaque inclut dans le fichier initiale pour un , définit l'endroit de construction. 9, 10, 22

plaquette de construction La plaque où le modèle va être assemblé dessus. 10,
23

ref_knob Position sur une pièce LEGO où le va tenir la pièce. 13, 14, 17

SmartFactory Usine Cyber-Physique construite par Festo. 3, 5, 7, 32, 33

têtons les éléments sur le dessus d'une pièce LEGO, permet l'embriquement des pièces. 10, 13

A.1 Code source

Le code source est hébergé sur le GitLab de la HEIA-FR, LEGO Repository

A.2 Versions des logiciels/librairie

Python : Version 3.6.3
pipenv : Version 8.3.2
pip : Version 9.0.1
PyYAML : 3.12
pytest : 3.3.2
PyInstaller : 3.3.1

A.3 Guide d'installation

Pour utiliser ce programme il faut avoir Python version 3.6 installé sur la machine.

A.3.1 Installation du projet pour le développement

Vérifier si *pip* est installé sur votre système, en exécutant

```
$ pip3 --version
```

Si la commande vous retourne des erreurs, installer *pip* comme indiqué sur le site <https://pip.pypa.io/en/stable/installing/>.

Une fois que vous avez *pip* installé, vous pourrez exécuter

```
$ pip3 install pipenv
```

Après l'installation de pipenv, vous pouvez aller dans le dossier racine du projet et exécuter

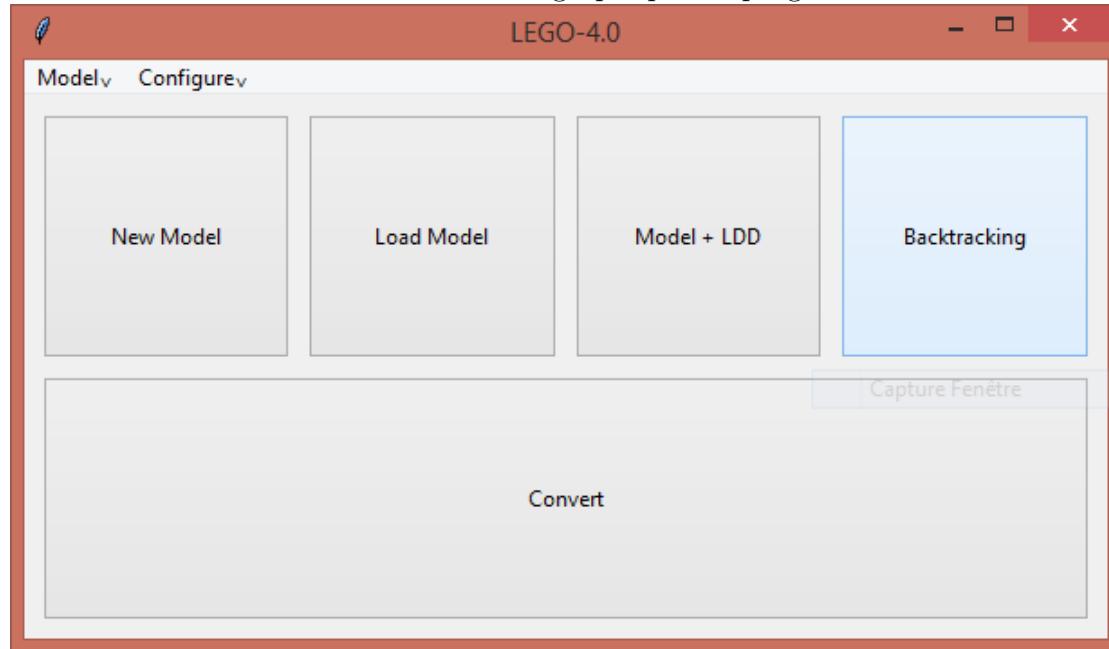
```
pipenv shell
```

Cette commande créera un environnement virtuel et installera les paquets nécessaires pour le développement.

A.4 Guide utilisateur

Cette section explique à l'utilisateur comment utiliser le programme. Pour lancer le programme, il suffit d'exécuter le fichier LEGO-4.exe. Une fois ouvert, l'interface présentée à la figure 28

FIGURE 28 – Interface graphique du programme



L'utilisateur a le choix des options suivantes.

- Création d'un nouveau modèle avec LDD
- Charger un model pour le convertir
- Charger un model et ouvrir LDD pour le visualiser
- Changer la méthode de génération des plaques de départ
- Lancer le processus de conversion

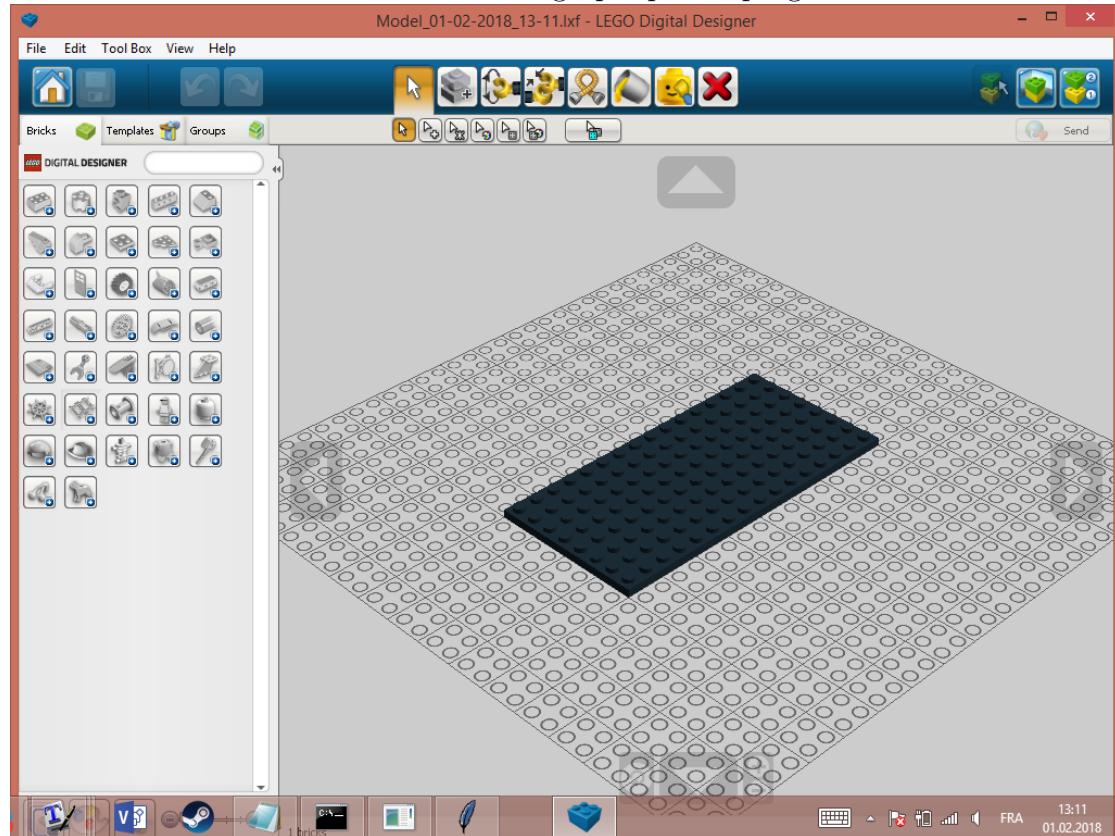
Sur le menu "Configure" de la barre en haut de la fenêtre, il y a la possibilité de modifier les fichiers de configuration pour appliquer des paramètres plus poussés.

- Changer le fichier de configuration
- Charger le fichier du dictionnaire
- Charger le template pour la plaque de départ
- Changer le mapping des rotations

A.4.1 Crédit à la création d'un nouveau model

Lorsque l'utilisateur démarre un nouveau modèle il se retrouve face à un écran similaire à la figure 29. Le programme a créé un fichier dans le dossier "home" de l'utilisateur, avec pour nom la date du moment. Une fois créé le programme a ouvert LDD sur ce fichier.

FIGURE 29 – Interface graphique du programme

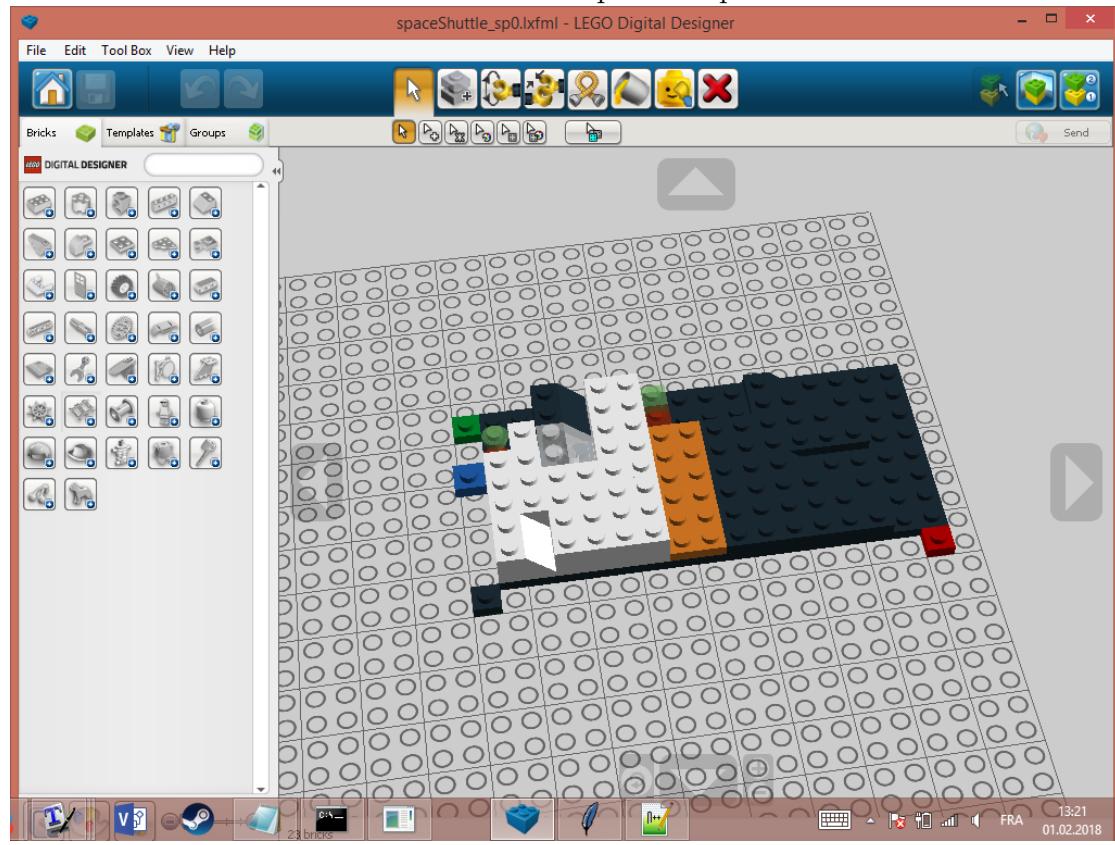


L'utilisateur voit alors une plaque noire, il s'agit de sa zone de construction. Une fois que l'utilisateur a placé ses pièces. Il doit sauvegarder le fichier, puis il peut fermer LDD.

A.4.2 Conversion

L'utilisateur peut maintenant cliquer sur le bouton "convert" pour lancer le processus de conversion. Une fois ce processus terminé, deux fenêtres apparaissent à l'écran, une 1ère avec l'éditeur de texte par défaut de l'utilisateur, affiche les instructions à envoyer à la machine. La seconde fenêtre s'ouvre avec LDD et présente la plaque de départ que l'utilisateur doit reproduire. Un exemple de plaque de départ est sur la figure 30.

FIGURE 30 – Plaque de départ



La petite pièce noire dans le coin de la plaque représente le bord inférieur gauche de la plaque, la pièce rouge est l'axe X, la verte l'axe Y et la bleue l'axe Z.

A.5 Ressources pour la base de donnée LDD

Les ressources pour la modification de la base de donnée sont dans le dossier *ressources* du répertoire du projet.

- LIF-Extractor est utilisé pour extraire les fichiers des pièces depuis la base de donnée LDD
- lego-extractor est le script créé par Jonathan Stoppani pour visualiser les fichiers des pièces dans un format .OBJ

La base de donnée est stockée aux emplacements suivants :

- **Windows :**

```
C:\Users\<user_name>\AppData\Roaming\LEGO Company\LEGO Digital
Designer\db.lif
```

- **MacOS :**

```
/Users/<user_name>/Library/Application Support/LEGO
Company/LEGO Digital Designer/db.lif
```

LEGO Digital Designer lit soit un fichier *db.lif* ou un dossier *db*.

A.6 Liste des exceptions

Deux classes d'exceptions ont été créées pour les besoins du programme. La première a pour rôle d'être lancée lors qu'une erreur dans le dictionnaire est détectée. La seconde concerne les erreurs dans le modèle Lego crée avec LDD. Voici les différents cas de figure où ces exceptions peuvent être lancées 31 et 32.

FIGURE 31 – Cas de figure où une exception "InvalidModelError" est lancée

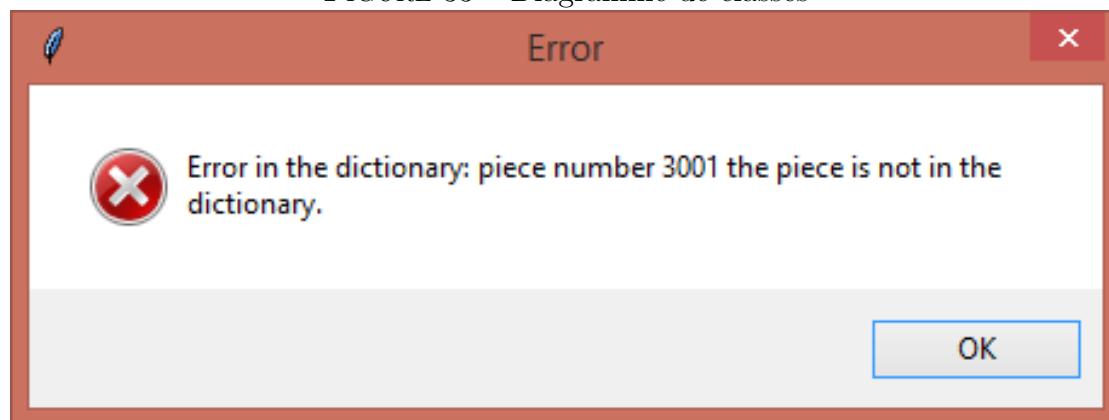
- La structure Lego dépasse les bords de la plaque de construction
- Orientation de la pièce invalide

FIGURE 32 – Cas de figure où une exception "DictionaryValueError" est lancée

- La pièce n'est pas dans le dictionnaire
- L'attribut "shape" n'est pas une matrice rectangulaire
- L'attribut "shape" ne contient pas que des nombres entiers
- L'attribut "shape" ne contient pas que des valeurs positives
- L'attribut "shape" n'existe pas
- L'attribut "ldd_pivot" n'est pas un tuple de 2 nombres entiers
- L'attribut "ldd_pivot" ne représente pas une coordonnée à l'intérieur de la pièce
- L'attribut "ldd_pivot" n'existe pas
- L'attribut "ref_knob" n'est pas un tuple de 3 nombres entiers
- L'attribut "ref_knob" ne représente pas une coordonnée à l'intérieur de la pièce
- Le *ref_knob* secondaire est d'une hauteur supérieur à celle du principal
- L'attribut "ref_knob" n'existe pas

Lors ce que l'un de ces cas surviens, le message d'erreur avec le numéro de la pièce incriminé apparaît à l'écran tel que sur la figure 33.

FIGURE 33 – Diagramme de classes



Toutes les exceptions sont systématiquement attrapée et montrée à l'utilisateur de cette manière.

A.7 Structure clé USB

- lego-4.0 : Dossier contenant le code source du programme
 - dist : Exécutable
 - lego : Code source du projet
 - Pipfile.lock
 - Pipfile
 - compile.sh : Script de génération de l'exécutable
 - LEGO-4.0-dir.spec
- Ordre du jour : (12 elements)
- Planning : (3 elements)
- PV : (13 elements)
- Cahier des Charges.pdf
- LEGO-2a.pdf
- Presentation_intermediaire_LEGO-2.pptx
- README.pdf