Licence informatique
2024

Documentation technique de développement

Noémie GIREAUD

Sommaire

1) Documentation TaBGO

Pour accéder au GitHub du projet TaBGO, suivre ce lien : TaBGO (Tangible Blocks Go Online) (github.com)

L'utilisation du projet TaBGO est décrite dans le fichier "README" du Github.

2) Documentation Convertisseur-SB3

Pour accéder au GitHub du convertisseur-sb3, suivre ce lien : NoemGir/SB3-SVG-Convertor (github.com)

a) Utilisation du convertisseur

Pour utiliser le convertisseur, il y a deux méthodes d'entrée :

- utiliser le convertisseur classique
- passer par un bus ivy.

Pour la présentation du résultat sur le fichier result.svg, deux présentations sont possibles :

- présentation classique avec lignes simples
- présentation avec lignes épaisses et bords colorés adaptée au parcours du robot Ozobot.

i) Utilisation classique

Pour utiliser le convertisseur de manière classique, exécutez le fichier "convertor.py" avec au minimum un premier argument indiquant l'emplacement du fichier sb3 à convertir : *python ./convertor <emplacement_sb3>.*

Un raccourci a été créé dans le cadre de l'exécution du convertisseur en partenariat avec le projet tabgo. Il suffit d'écrire python *./convertor tabgo* pour que le convertisseur recherche le fichier dans "*../tabgo/data/sb3/Programme_scratch.sb3".*

Pour l'apparition du résultat, il y a deux options possibles :

Option 1 : Ne rien ajouter après l'indication de l'emplacement assure que le résultat obtenu soit une ligne simple.

Option 2 : Possibilité de rajouter en argument dans cet ordre : le scale (int), la translation x (int), la translation y (int).

- Le **scale** permet d'augmenter ou de réduire la taille de la figure (mettre 1 si on ne souhaite pas modifier la taille).
- La **translation x** indique le nombre de pixels vers lequel la figure doit bouger (horizontalement).
- La **translation y** indique le nombre de pixels vers lequel la figure doit bouger (verticalement).

Les transformations x et y sont utiles si la figure dépasse des cadres du SVG après l'augmentation de la taille (scale > 1).

Si le scale est donné avec l'Option 2 (si plus de 1 argument est donné), alors le rendu est forcément adapté au robot Ozobot.

ii) Utilisation ivy bus

L'ouverture Ivy du convertisseur permet de le bloquer dans une boucle en communication avec le bus Ivy. Ainsi, il est ouvert à 4 types de messages :

- **"tabgo : *"** -> exécute le programme en cherchant le sb3 généré par TaBGO avec rendu classique.
- **"print tabgo: .*"** -> exécute le programme en cherchant le sb3 généré par TaBGO avec rendu Ozobot.
- **"convert:(.*?)location=.*"** -> exécute le programme avec sb3 donné et rendu classique.

- **"print convert:(.*?)location=.*"** -> exécute le programme avec sb3 donné et rendu Ozobot.

Lorsqu'un autre programme envoie ainsi un message suivant un type indiqué, l'action reliée sera immédiatement exécutée. Possibilité de rajouter *"scale=.*"*, "*x=.**" et "*y=.**" pour le cas print.

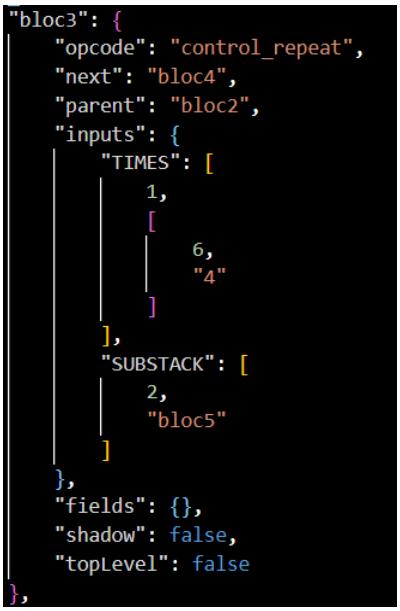
c) Fonctionnement du code

i) Fonctionnement json

Un fichier Scratch est représenté par un fichier .sb3. En réalité, un .sb3 est un dossier compressé, qui, une fois décompressé, contient un fichier .json qui répertorie, entre autres, l'ensemble des blocs présents dans l'algorithme Scratch. Un bloc possède, en plus de son nom, plusieurs attributs. Les attributs utiles pour notre convertisseur sont :

- "*opcode*" : l'identificateur du bloc, qui décrit quel type d'action le bloc réalise.
- "*next*" : le nom du bloc suivant.
- "*parent*" : le nom du bloc précédent ou englobant.
- "*inputs*" : les valeurs données avec le bloc.
- "*fields*" : les valeurs choisies avec le bloc.
- "*proccode*" dans l'attribut "*mutation*" : décrit le nom du bloc personnalisé associé.

Exemple de description d'un bloc dans un json :



ii) Fonctionnement svg

Les lignes dans un SVG sont décrites dans l'espace "*path*", grâce à l'attribut *"d"*. Plusieurs lettres sont utilisées pour la création du convertisseur :

- M sert à indiquer au curseur de se rendre aux coordonnées indiquées (sans tracer de trait).
- m indique au curseur d'ajouter les valeurs données à la position courante (sans tracer de trait).
- L indique au curseur de se déplacer aux coordonnées indiquées en traçant un trait.
- l indique au curseur d'ajouter les valeurs données à la position courante en traçant un trait.
- V déplace le curseur verticalement en traçant une ligne et se dirige vers la position indiquée.
- v ajoute la valeur donnée au curseur sur l'axe des ordonnées, et trace la ligne du chemin parcouru.
- H déplace le curseur horizontalement en traçant une ligne et se dirige vers la position indiquée.
- h ajoute la valeur donnée au curseur sur l'axe des abscisses, et trace la ligne du chemin parcouru.

Les attributs "*stroke*" et "*stroke-width*" permettent de définir respectivement la couleur et l'épaisseur du trait.
L'attribut "*translate*" permet de modifier l'emplacement de la figure.
L'attribut "*viewBox*" donne la taille de la fenêtre du SVG.
L'attribut "*fill*" indique si la figure doit être remplie ou non.

Exemple de svg généré et son résultat :

```
<?xml version="1.0"?>

<svg viewBox="0 0 550 425" xmlns="http://www.w3.org/2000/svg" xmlns:svg="http://www.w3.org/2000/svg">
  <g transform="translate(0,0) rotate(0)" fill="none"  stroke-width="14">
    <path stroke = "black" d="M 275.0,212.5 "/>
    <path stroke = "red" d= "M 275.0,212.5 l 14.0,-0.0 "/>
    <path stroke = "blue" d= "M 289.0,212.5 l 14.0,-0.0 "/>
    <path stroke = "black" d= "M 303.0,212.5 l 72.0,-0.0 "/>
    <path stroke = "red" d= "M 375.0,212.5 l -14.0,-0.0 "/>
    <path stroke = "blue" d= "M 361.0,212.5 l -14.0,-0.0 "/>
    <path stroke = "black" d= "M 375.0,212.5 "/>
  </g>
</svg>
```

iii) Fonctionnement convertisseur

Pour assurer la création du nouveau SVG qui se nommera "*result.svg*", le convertisseur se sert d'un SVG vide de référence ("*blank.svg*"), qu'il remplit avec les nouvelles lignes ajoutées par le parcours des blocs.

Tout d'abord, le convertisseur décompresse le fichier SB3 donné et analyse le fichier JSON résultant. Dans le module "*patternMatching*", la fonction "*createDictionary*" est chargée de parcourir l'ensemble des blocs décrits dans le JSON, et pour chacun d'entre eux, de créer un objet de type Bloc regroupant toutes les caractéristiques utiles du bloc, décrites précédemment dans la section i). Par la suite, l'ensemble des objets Blocs créés est ajouté dans un dictionnaire qui, à chaque nom de bloc, associe son objet décrivant ses caractéristiques. Cette étape permet de faciliter le parcours de l'algorithme.

Par la suite, dans le module "*converter*", deux fonctions sont primordiales :
- "*sequenceLoop*" permet de réaliser un parcours de l'ensemble des blocs de l'algorithme. Pour ce faire, on lui donne en argument le premier bloc, et il va continuer à passer de bloc en bloc jusqu'à ce que le prochain bloc n'ait plus de suivant (next). Chaque bloc parcouru est analysé, permettant de retracer les actions Scratch effectuées et ainsi d'obtenir les lignes à ajouter dans le SVG.
- "*blockAnalysis*" est chargée de l'analyse d'un bloc. En fonction de son opcode, l'action résultante sera effectuée.

De nombreux blocs possèdent des effets spéciaux lors de leur analyse. Tout d'abord, il y a les blocs de mouvement, qui définissent une coordonnée à laquelle se déplacer ou ajoutent des valeurs aux coordonnées existantes. Ces blocs ajoutent des lignes dans le SVG. D'autres blocs servent à modifier l'état actuel du programme, par exemple les blocs "*pen_penDown*", "*pen_penUp*", ou alors "*motion_setDirection*", "*motion_turnright*"... Cela indique au programme si les prochains mouvements tracent des traits ou changent la direction vers laquelle le curseur doit pointer avant d'avancer. Comme les SVG n'acceptent que des coordonnées comme valeurs, il est alors souvent

nécessaire de calculer les coordonnées en fonction de la distance à parcourir et de l'orientation actuelle. Pour finir, certains blocs possèdent une section de blocs propre à eux, comme les blocs de contrôle (répéter, si…alors, etc.) ou les blocs de procédure (blocs personnalisés). Lorsque l'on entre dans cette section, celle-ci est traitée grâce à la fonction *sequenceLoop* mentionnée précédemment.

Pour continuer, certains blocs possèdent des valeurs, comme par exemple le bloc "*motion_movestep*" qui prend comme valeur le nombre de pas à avancer. Pour l'ensemble de ces blocs, un autre bloc opérateur peut être mis comme input, et la valeur sera donc le résultat de l'opération. Ainsi, il est nécessaire d'étudier le cas des blocs opérateurs pour chaque valeur donnée dans un bloc. Cela s'effectue dans le module "*computeValues*", la fonction "*getValue*". Elle repère si la valeur dans le bloc représente le nom d'un autre bloc présent dans le dictionnaire, ou s'il s'agit seulement d'une valeur simple. S'il s'agit d'un bloc présent dans le dictionnaire, elle analyse le bloc avec la fonction "*operatorCompute*", qui, en fonction de l'opération du bloc, la réalise sur les deux valeurs données en argument du bloc opérateur (qui peuvent elles-mêmes être des blocs opérateurs).

Un détail auquel il faut faire attention dans le convertisseur : celui-ci utilise une orientation trigonométrique. C'est-à-dire que le point de droite représente 0 degrés, celui du haut 90 degrés, gauche 180 degrés et bas 270 degrés. Cependant, le système d'orientation sur Scratch est différent : il place le point du haut à 0 degrés, droite à 90 degrés, gauche à -90 degrés et bas à 180 degrés. Il est donc nécessaire d'établir une conversion entre les deux valeurs avant de les utiliser.

Dans le cas d'un résultat adapté avec le robot Ozobot, la modification de la taille du trait, de la translation et du changement de couleur se fait grâce au module "*fileModification*". Les manipulations nécessaires afin de générer les différentes couleurs sont quant à elles réalisées dans le module "*convertor*". Dès qu'un bloc "*pen_penDown*" est détecté, alors la variable booléenne "*isNewMovement*" est mise à True. Ainsi, lors du prochain mouvement réalisé, la couleur se matérialisera. Dans le cas d'une fin de ligne, cela peut se détecter avec un bloc "*pen_penUp*" ou lors de la fin du parcours de l'algorithme. Cependant, il faut s'être assuré qu'un trait a bien été tracé précédemment, d'où l'utilisation de la variable "*hasMoved*". Enfin, il faut également se souvenir de l'orientation du dernier trait réalisé, pour que la couleur soit bien placée dans la bonne orientation. Cela nous impose de calculer l'orientation présente pour chaque mouvement effectué. Finalement, un dernier travail à effectuer est de réduire la taille du prochain trait après l'ajout de la couleur en début de ligne. Cela empêche le trait d'aller plus loin que la distance indiquée à cause de l'ajout des couleurs.

c) Description du module “blocStructure”

```
class Bloc(NamedTuple):
    """A structure capable of stocking all the informations related to a bloc"""
    opcode: str
    parent : str
    next: str
    inputs: array
    fields: array
```

Le module "*blocStructure*" contient seulement la définition d'un bloc, avec les attributs suivants :

- *opcode* : le opcode du bloc.
- *parent* : le nom du bloc parent.
- *next* : le nom du bloc suivant.
- *inputs* : correspond aux valeurs données avec le bloc.
- *fields* : correspond aux valeurs choisies dans les champs du bloc.

d) Description du module “patternMatching”

Cette classe possède toutes les fonctions réalisant du pattern Matching.

La fonction *createDictionary(fileReading: str)* parcourt l'ensemble des blocs présents dans la chaîne de caractères donnée en entrée. Grâce au pattern matching, elle identifie d'abord les blocs, puis extrait les différentes valeurs (opcode, parent, next, inputs, fields) associées à chaque bloc. Elle retourne un dictionnaire qui associe à chaque nom de bloc son objet Bloc contenant les informations correspondantes.

La fonction *getInputs(inputs: str, pattern: regex)* recherche dans la chaîne de caractères "inputs" toutes les valeurs séparées par des virgules qui correspondent au modèle donné par le pattern regex. Cette fonction est utilisée pour extraire les valeurs à l'intérieur des champs "inputs" et "fields".

La fonction *advanceBlock(fileReading: str)* permet de faire avancer la lecture de la chaîne de caractères "*fileReading*" jusqu'à la seconde occurrence du mot "block", c'est-à-dire juste avant la description de l'ensemble des blocs Scratch présents.

La fonction *get_first_blocks(fileReading: str)* recherche spécifiquement l'ensemble des blocs qui sont des blocs de départ (commençant par "*event_...*").

Les fonctions *recognize_X_size(fileReading: str)* et *recognize_Y_size(fileReading: str)* recherchent respectivement la largeur et la hauteur du SVG créé.

Les fonctions *get_location(text: str)*, *get_x(text: str)*, *get_y(text: str)* et *get_scale(text: str)* sont utilisées dans le cas de la transmission d'informations depuis le bus Ivy. Elles lisent les messages du bus et en extraient respectivement la localisation, la translation en X, la translation en Y et l'agrandissement (scale).

e) Description du module “fileManipulator”

Ce module représente l’ensemble des fonctions qui permettent de lire/écrire des fichiers.

La fonction *extractSB3(sb3_name: str)* décompresse le fichier SB3 donné en argument et récupère le fichier .json à l'intérieur pour le placer dans le répertoire courant.

La fonction *insertLinesSVG(newLines: list str)* insère les mouvements créés dans le fichier result.svg et indique le nombre de lignes ajoutées.

La fonction *getInitialCoordinate()* recherche dans le fichier SVG référentiel (blank.svg) la taille du SVG et retourne les coordonnées du milieu.

La fonction *modifySize(scale: int)* modifie la taille du SVG généré en la multipliant par le scale donné.

La fonction *transform(x, y)* initialise les valeurs à l'intérieur de l'attribut "translate" pour modifier le positionnement de la figure.

La fonction *addBigStroke()* augmente l'épaisseur du trait.

La fonction *JSONreader()* lit le project.json dans le répertoire courant et renvoie le dictionnaire ainsi que la liste des premiers blocs décrits à l'intérieur du JSON.

La fonction *putColor(coordinate, color)* retourne ce qu'il faut ajouter à la ligne pour changer la couleur du trait actuel. L'argument coordinate sert à préserver les coordonnées du curseur actuel.

f) Description du module “convertor”

Le module convertor est le module principal : il se charge d'analyser la suite de blocs de l'algorithme et de réaliser leurs actions. Les variables globales sont les suivantes :

- *draw* : boolean -> indique si le mouvement doit tracer un trait ou non.
- *coordinate* : [int, int] -> les coordonnées actuelles du curseur.
- *initCoordinates* : [int, int] -> les coordonnées du milieu du SVG.
- *orientation* : int -> le degré de l'orientation actuelle (droite = 0 degrés).
- *variables* : dict -> dictionnaire qui associe à chaque nom de variable sa valeur.
- *scale* : int -> l'agrandissement de la figure.
- *color* : boolean -> indique si les couleurs doivent être ajoutées en début/fin de lignes ou non.
- *orientationLastMovement* : int -> sauvegarde l'orientation présente lors du dernier mouvement (cela servira pour les couleurs, pour savoir dans quelle orientation les ajouter).
- *isNewMovement* : boolean -> indique si le mouvement qui suit sera un début de ligne ou non (pour savoir s'il est nécessaire d'ajouter les couleurs).
- *hasMoved* : boolean -> indique si le curseur a bougé en créant une ligne (pour savoir s'il est nécessaire ou non de colorier les extrémités).

La fonction *rightBloc(bloc)* retourne un booléen indiquant si le bloc fait partie des blocs reconnus par le convertisseur. Cette fonction utilise l'opcode du bloc et le compare avec les opcodes des blocs connus.

La fonction *generateLine(lettre, x, y)* retourne une chaîne de caractères représentant un mouvement défini par la lettre, ainsi que les deux coordonnées x et y.

La fonction *moveCase(dico, bloc, orientation)* représente le cas où un bloc de mouvement est exécuté. Le bloc de mouvement correspond à un bloc d'avancement ou de modification de x et y. Plusieurs cas sont présents dans cette fonction : si la ligne doit être coloriée ou s'il faut ou non tracer le trait du mouvement. Cette fonction doit calculer les nouvelles coordonnées du point en fonction de l'orientation et de la distance d'avancée.

La fonction *set_X(dico, bloc)* représente le cas où un bloc définissant x est exécuté. Plusieurs cas sont présents dans cette fonction : si la ligne doit être coloriée ou s'il faut ou non tracer le trait du mouvement. De plus, cette fonction définit l'orientation du dernier mouvement.

La fonction *set_Y(dico, bloc)* représente le cas où un bloc définissant y est exécuté. Plusieurs cas sont présents dans cette fonction : si la ligne doit être coloriée ou s'il faut ou non tracer le trait du mouvement. De plus, cette fonction définit l'orientation du dernier mouvement.

La fonction *goTo_X_Y(dico, bloc)* représente le cas où un bloc définissant x et y est exécuté. Plusieurs cas sont présents dans cette fonction : si la ligne doit être coloriée ou s'il faut ou non tracer le trait du mouvement. De plus, cette fonction définit l'orientation du dernier mouvement.

La fonction *modifyOrientation(dico, bloc, droite)* représente le cas où l'orientation est modifiée. "droite" est un booléen indiquant si le sens de rotation est vers la droite ou vers la gauche.

La fonction *getValue(dico, bloc, id)* récupère la valeur à l'intérieur de l'indice id dans l'input du bloc donné. Cette fonction est utile si la valeur donnée est un bloc d'opération et renverra donc la valeur résultant de l'opération.

La fonction *inEdge()* indique si le curseur actuel touche le bord du SVG.

La fonction *condition(dico, bloc)* recherche si la condition donnée par le bloc est vraie ou non.

La fonction *controlRepeat(dico, bloc)* exécute les blocs à l'intérieur du bloc "control repeat" donné, autant de fois que décrit sur le bloc.

La fonction *controlRepeatUntil(dico, bloc)* exécute les blocs à l'intérieur du bloc "control repeat until" donné, tant que la condition donnée par le bloc est vraie.

La fonction *controlIfElse(dico, bloc)* recherche la condition donnée par le bloc "control if else" et exécute les codes internes si la condition est vraie ou fausse.

La fonction *controlIf(dico, bloc)* recherche la condition donnée par le bloc "control if" et exécute le code interne si la condition est vraie.

La fonction *changeVariable(dico, bloc)* modifie la valeur à l'intérieur d'une variable de l'environnement. Si la variable n'existait pas, elle est créée et initialisée avec la valeur donnée dans le bloc.

La fonction *drawColors(orient)* ajoute les sections de couleurs rouge et bleue indiquant une fin de ligne. L'argument "orient" indique l'orientation du placement des couleurs.

La fonction *drawColorEndLine()* est appelée lors de la fin du tracé d'une ligne et ajoute les couleurs si nécessaire.

La fonction *drawColorStartLine()* est appelée lors du début du tracé d'une ligne et sert à ajouter les couleurs.

La fonction *blockAnalysis(dico, bloc)* étudie le bloc actuel et exécute l'action liée au bloc. Elle retourne l'addition que ce bloc ajoute à l'SVG.

La fonction *sequenceLoop(dico, bloc)* réalise une boucle depuis le bloc donné jusqu'à ce que celui-ci n'ait plus de bloc suivant. Elle est utilisée pour la lecture de l'algorithme, mais aussi pour l'exécution des sections de code à l'intérieur des blocs de contrôle et des blocs personnalisés. Cette fonction renvoie l'ensemble des nouvelles actions réalisées par les blocs de la section.

La fonction *generateNewLines()* initie l'analyse de l'algorithme et lance l'écriture du résultat dans le SVG.

La fonction *convertor(location)* décompresse le fichier SB3 donné en argument et lance la création du fichier SVG.

La fonction *matchLocation(location)* analyse la localisation donnée et lance le convertisseur avec une localisation adaptée.

La fonction *convert(location, printable, new_scale, x, y)* analyse les arguments donnés pour lancer le convertisseur dans le mode voulu (imprimable ou simple).

La fonction *runConvertor(location, printable, new_scale, x, y)* analyse les arguments donnés dans le système avant de lancer le convertisseur.

g) Description du module “computeValues”

Cette classe possède toutes les fonctions réalisant des calculs complexes.

La fonction *compute_x(distance, orientation)* retourne la coordonnée x (axe des abscisses) obtenue après avoir avancé de la distance donnée dans l'orientation donnée.

La fonction *compute_y(distance, orientation)* retourne la coordonnée y (axe des ordonnées) obtenue après avoir avancé de la distance donnée dans l'orientation donnée.

La fonction *computeOrientation(initialCoord, finalCoord)* calcule l'angle entre une ligne horizontale et la ligne formée par le parcours des coordonnées *initialCoord* aux coordonnées *finalCoord*.

La fonction *convertOrientation(scratchOrient)* convertit l'orientation Scratch donnée en argument en orientation utilisée par le convertisseur.

La fonction *computeDistance(coordinate, destination)* calcule la distance entre les deux points donnés en arguments.

La fonction *getValue(dico, bloc, id, variables, initCoordinates, coordinates, orientation)* récupère la valeur à l'intérieur de l'indice id dans l'input du bloc donné. Cette fonction est utile si la valeur donnée est un bloc d'opération, et renverra donc la valeur résultant de l'opération.

La fonction *operatorCompute(dico, bloc, initCoordinates, coordinates, orientation, variables)* analyse le bloc opérateur donné et retourne la réalisation de l'opération correspondante.

h) Description du module “ivyConvertor”

Cette classe sert à initialiser un bus Ivy et récupère des ordres pour lancer le convertisseur automatiquement.

La fonction *on_tabgo(agent)* lance le convertisseur qui recherchera le fichier SB3 dans l'espace de fichier SB3 de Tabgo. Elle servira à générer des lignes classiques.

La fonction *on_tabgo_printable(agent)* lance le convertisseur qui recherchera le fichier SB3 dans l'espace de fichier SB3 de Tabgo. Elle servira à générer des lignes Ozobot.

La fonction *print_given(agent, text)* lance le convertisseur sur le fichier SB3 donné dans le texte. Elle servira à générer des lignes Ozobot.

La fonction *on_given(agent, text)* lance le convertisseur sur le fichier SB3 donné dans le texte. Elle servira à générer des lignes classiques.