Licence Informatique
L3
2024

Rapport final
Compte Rendu du Stage

Noémie GIREAUD 22205610

Rendre la programmation en Scratch accessibles aux enfants malvoyants

Tuteur : Philippe TRUILLET
Affiliation : IRIT

Sommaire

1 - Introduction..... 2

2 - Présentation de TaBGO..... 2

3 - Organisation..... 4

4 - Présentation personnelle..... 6

a) Attentes..... 6

b) Choix du Feedback..... 6

c) Description d'un SVG..... 6

d) Description d'un .JSON de Scratch..... 8

e) Réalisation du convertisseur..... 9

f) Adaptation ivy Bus..... 10

g) Différents types de rendu..... 11

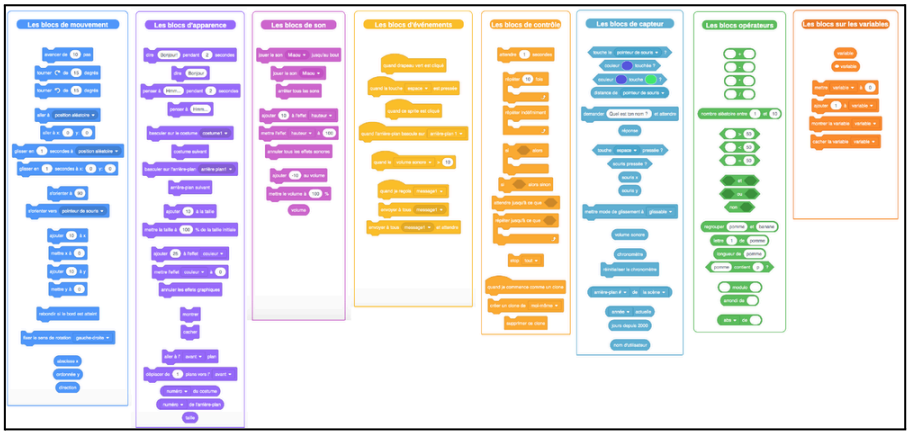
5 - Bilan personnel..... 11

Parties 1, 2 et 3 écrites en commun par Ninon AUTEFAGE-PINIES, Loan VIGOUROUX, Adrian MORELLATO et Noémie GIREAUD.

1) Introduction

Scratch est un environnement de programmation conçu pour aider les enfants et les débutants à faire leurs premiers pas dans le domaine de la programmation informatique. A l'aide de blocs (*Figure 1*) représentant des éléments algorithmiques, il est possible de faire des calculs, tracer des formes géométriques et déplacer un petit chat virtuel dans un espace en deux dimensions. Cet environnement permet l'utilisation des variables, des boucles, des conditions et des entrées utilisateur de manière ludique et visuelle. Il est notamment utilisé au collège depuis la réforme de 2016 dans le cadre de l'introduction à la programmation. Cependant, en tant qu'interface visuelle sur un ordinateur, on peut s'interroger sur son accessibilité, en particulier pour les personnes malvoyantes. C'est pourquoi le projet TaBGO a vu le jour en 2017. A l'aide de blocs tangibles, il a pour objectif de rendre plus autonomes les enfants malvoyants quant à la programmation sur Scratch au collège. En tant que groupe de quatre étudiants, Loan VIGOUROUX, Ninon AUTEFAGE-PINIES, Adrian MORELLATO et Noémie GIREAUD, nous devons poursuivre le projet existant et y ajouter nos propres propriétés et extensions pour l'adapter aux blocs tangibles nouvellement créés.

Fig. 1 - Blocs Scratch



2) Présentation de TaBGO

Afin de rendre Scratch accessible aux enfants malvoyants, la première innovation a été de créer des blocs tangibles, qui, combinés, permettent de former des algorithmes. De nombreux modèles ont été réalisés et ont évolué au fil du temps, passant de premiers blocs en bois à de nouveaux blocs magnétiques et flexibles (*Figure 2*). C'est TaBGO qui permet de passer de blocs tangibles à un programme Scratch en utilisant la reconnaissance d'image. Cependant, comment chaque bloc de l'image est-il reconnu et associé à un bloc Scratch existant ? Ceci est possible grâce au TopCode (*Figure 3*) présent sur chaque bloc tangible. Un

TopCode est analogue à un QRCode et est associé à un opcode indiquant la nature du bloc. Aussi, certains blocs possèdent des variables internes qui doivent être initialisées par l'utilisateur. Par exemple, le bloc “avancer de … pas” doit recevoir un nombre de pas à effectuer. Pour permettre l’initialisation de ces variables, la version actuelle de TaBGO utilise les cubarithmes (*Figure 4*). Un cubarithme est un petit cube dont les faces comportent différentes combinaisons de points en reliefs. Grâce à son mécanisme semblable à celui d’un rubik's-cube, il est possible de modifier l’arrangement des points pour représenter une lettre ou un chiffre en braille. TaBGO peut reconnaître le braille présent sur un cubarithme, par conséquent, le placement d’un cubarithme à côté d’un bloc comportant une variable interne permettra l’initialisation de cette variable. Ainsi, le code, réalisé en Processing et accessible sur GitHub, scanne et convertit les différents TopCodes des blocs, et les cubarithmes présents et génère directement un fichier sb3 qui pourra être utilisé dans Scratch.

Fig. 2 - Blocs tangibles magnétiques

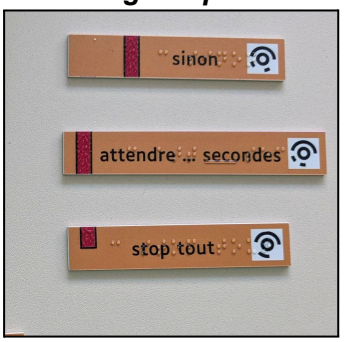
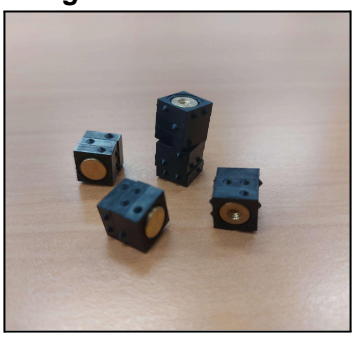


Fig. 3 - TopCode

Fig. 4 - Cubarithmes



En arrivant sur ce projet, ceci était la situation face à laquelle nous nous somme trouvés. De nombreuses questions se sont posées quant à ce que l'on pourrait ajouter au projet, qui semblait au départ parfaitement fonctionner. Cependant, nous nous sommes vite rendu compte que certaines fonctionnalités du projet étaient manquantes ou non fonctionnelles. Tout d’abord, l’usage des cubarithmes évoqués plus haut soulève quelques problématiques. En effet, leur petite taille les rendait difficilement malléables pour les enfants, et difficilement reconnaissables par TaBGO sur une photo. De plus, le code a été adapté à une ancienne version de blocs tangibles capable de contenir des cubarithmes entre les blocs. Cependant, dans la dernière version de blocs tangibles (la version magnétique), tous les cubarithmes doivent être placés à droite des blocs. Cette nouvelle gestion des variables est le premier problème que nous devions résoudre. Plus tard, nous avons également fait part de nos inquiétudes sur le manque de praticité du code, ainsi que sur le manque de retours. En effet, il était initialement difficile de comprendre comment récupérer et exécuter le code créé par TaBGO. En faisant part de notre inquiétude, nous avons appris qu’en pratique les professeurs

n'utilisaient même pas le code créé par TaBGO, mais qu'ils réécrivaient le code réalisé par leur élève directement sur Scratch, à la main. Enfin, même si quelques recherches ont été faites dans ce domaine, aucun système de retour d'expérience n'a été créé et utilisé, de sorte que les enfants ne peuvent même pas avoir une bonne représentation du code qu'ils ont réalisé. Pour résumer, l'utilisabilité de TaBGO, ainsi que le système de variables et la création de feedback étaient nos principales préoccupations et enjeux concernant notre projet.

3) Organisation

Forts de toutes ces préoccupations, notre objectif était alors de permettre aux enfants et aux enseignants de créer le code à partir de blocs tangibles, mais aussi de l'exécuter dans Scratch, et enfin d'accéder à un retour complet, et tout cela en parfaite autonomie. Tout d'abord, nous avons décidé que Ninon AUTEFAGE-PINIES serait chargée de créer une interface à l'application TaBGO ainsi que de faciliter son installation. Cela permettra aux enseignants ainsi qu'aux enfants malvoyants d'exécuter le code TaBGO. De son côté, Loan VIGOUROUX souhaitait implémenter le nouveau système de variables et gérer la transition entre les deux systèmes. Cela permettra aux enfants de créer leur code en utilisant différentes valeurs comme entrées pour les blocs. Adrian MORELLATO a quant à lui travaillé sur un feedback orienté professeurs avec un script qui ouvre directement Scratch en important le .sb3 résultant de l'exécution de TaBGO. Grâce à cela, les enseignants pourront visualiser les blocs réalisés en blocs tangibles sur Scratch. Pour finir, Noémie GIREAUD s'est concentrée sur l'implémentation de feedbacks pour les enfants, afin qu'ils puissent percevoir le code qu'ils créent. Par conséquent, tout le monde à travaillé sur un des objectifs principaux de ce projet, avec l'espoir de rendre le projet TaBGO réellement utilisable par les professeurs et ses élèves, sans l'aide d'un visiteur extérieur.

Afin de s'organiser, nous avons effectué de nombreuses réunions entre membres du groupe mais également avec notre tuteur Philippe Truillet. Cela nous à permis de s'informer sur le projet TaBGO et de faire part de nos avancées personnelles, mais aussi des difficultés rencontrées, et de trouver ensemble les moyens d'y remédier. À distance, Discord fut l'outil le plus utilisé pour communiquer, et GitHub nous servit à transmettre le code en ligne et à la portée de tous. Après avoir assimilés les problématiques liées au projet TaBGO et s'être répartis les rôles de chacun, nous avons ensuite établi une liste de tâches (*Figure 5*) et un planning référentiel (*Figure 6*) devant être respectées par l'ensemble du groupe.

Fig. 5 - Work Breakdown Structure

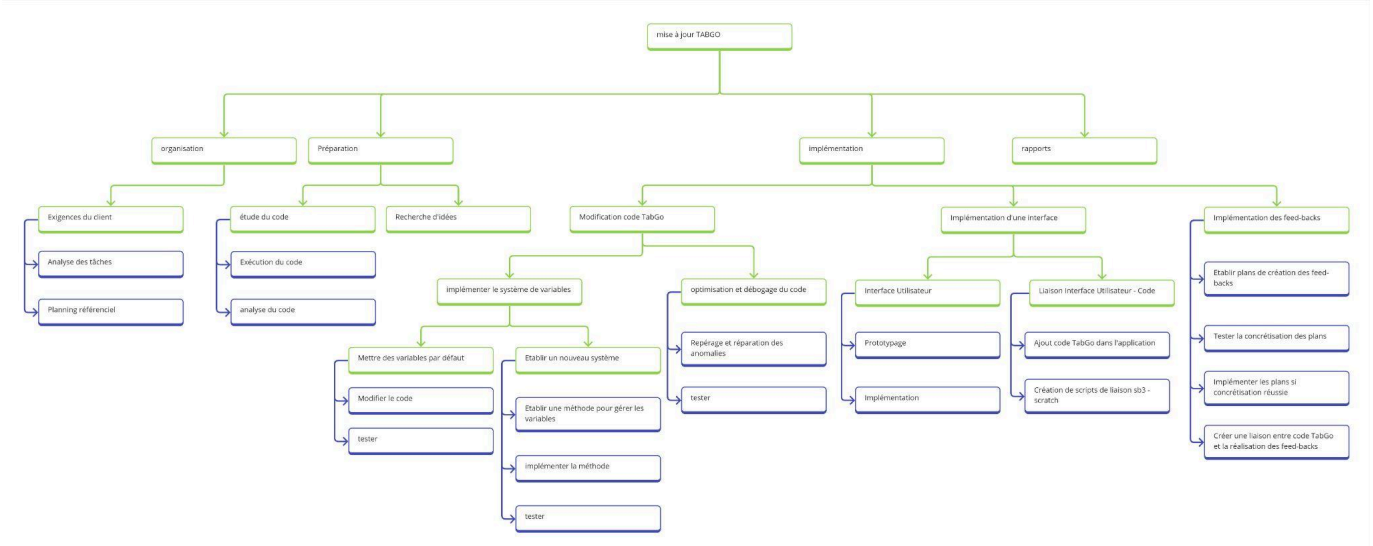
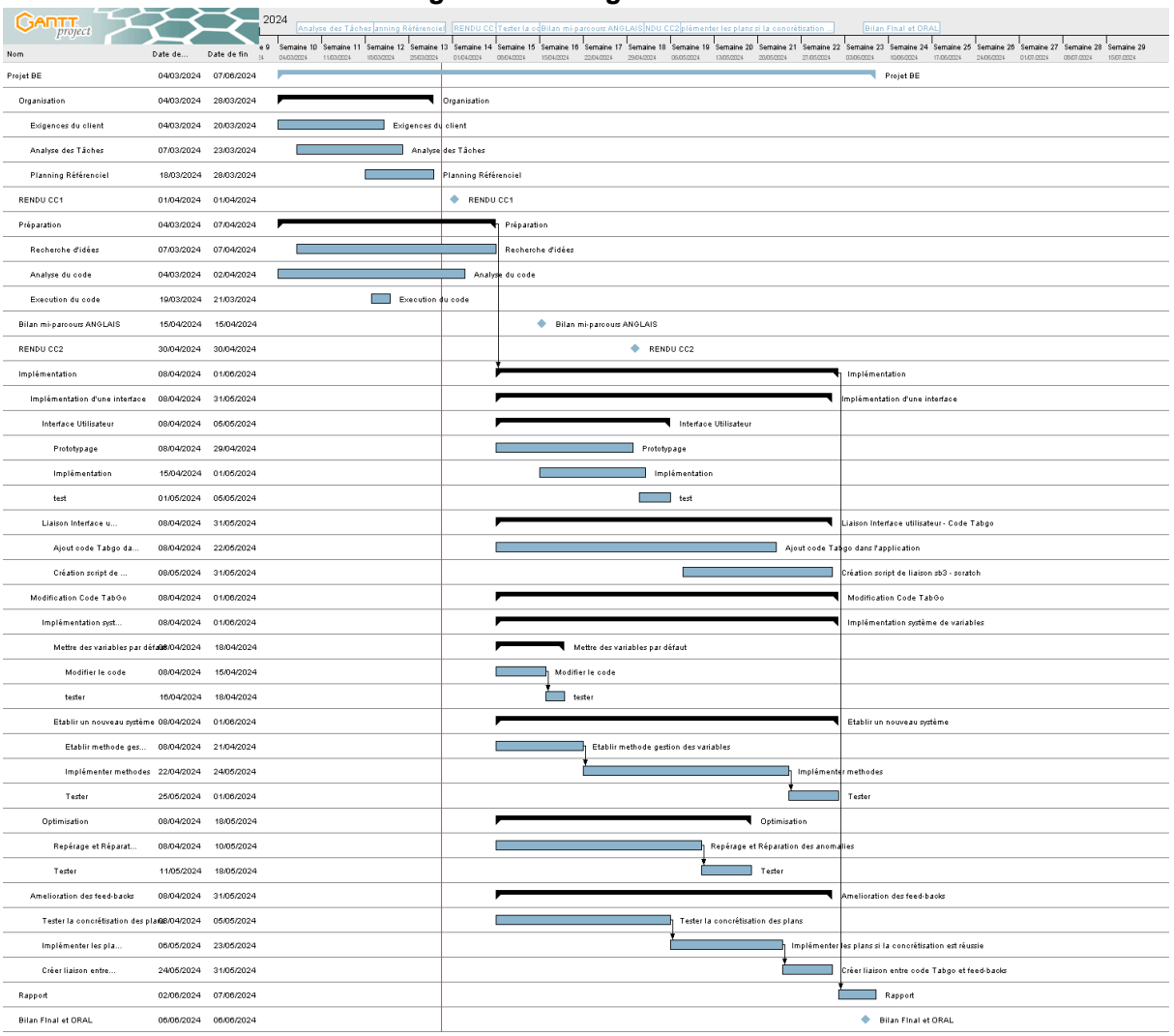


Fig. 6 - Planning référentiel



4) Présentation personnelle

a) Attentes

En rappel avec l'organisation citée plus haut, je suis chargée d'accorder aux enfants malvoyants un retour accessible leur permettant d'avoir une représentation du code qu'ils ont créé. Tout d'abord, je vais commencer par expliquer ce qui était attendu de moi pour ce projet. Même si celui-ci a commencé avec l'idée d'être utilisé par des enfants malvoyants, il permettra également un travail collaboratif entre des enfants malvoyants et des enfants voyants. En tant que tel, ce retour d'information doit être adapté à tous, aussi bien voyants que malvoyants. De plus, je devais garder à l'esprit que ce projet est destiné à aider les enfants dans leur apprentissage de la programmation en Scratch, dans l'optique de passer leur brevet. Par conséquent, j'ai voulu adapter ce feedback au type d'exercices qu'ils feraient le plus : représenter des formes à l'aide des blocs Scratch. Il me fallait donc un feedback permettant de visualiser des formes en 2D, qui représenteraient les dessins réalisés sur Scratch.

b) Choix du Feedback

Après avoir listé les différentes possibilités pour réaliser un retour de la sorte, j'ai retenu quelques idées principales. Par exemple, la création d'une tablette tangible, où des pics seraient soulevés à l'emplacement des lignes du dessin. Une autre idée était de créer un robot qui pourrait être suivi par le son ou le toucher et qui tracerait le chemin du dessin. Ces deux idées nécessitaient des matériaux et une ingénierie complexes, et ne me semblaient pas très accessibles. De plus, je pouvais également étudier les environnements sonores, et la façon de représenter des mouvements et formes avec seulement des sons autour de soi. Cependant, cette approche avait déjà été étudiée et le résultat n'était pas fructueux. C'est alors que j'ai commencé à m'intéresser aux imprimantes et graveurs 3D. Grâce à eux, les enfants pourraient toucher et sentir la forme du dessin, et même l'emporter chez eux. Néanmoins, comme je n'avais pas d'imprimante ou de graveur 3D sous la main, j'ai concentré mes recherches sur le fichier d'entrée qu'ils acceptent : le SVG.

c) Description d'un SVG

Mon projet était donc de créer un convertisseur .sb3 vers .svg, qui retranscrirait le dessin normalement représenté sur Scratch sur un fichier .svg, afin de permettre l'impression du dessin. J'ai ainsi réalisé du reverse engineering afin de comprendre le fonctionnement des fichiers .svg.

SVG signifie Scalable Vector Graphics (graphique vectoriel évolutif). Il peut être comparé à une image représentée par des vecteurs et non des pixels, de sorte que l'échelle est modifiable à l'infini et que la forme peut être définie dans le fichier lui-même *(Figures 7 et 8)*. Les formes représentant la figure sur le SVG peuvent être soit des lignes, soit des courbes, et il est possible de remplir les formes, changer leur couleur, épaisseur, ainsi que modifier de nombreuses autres propriétés. Un fichier SVG se base sur un système de coordonnées mettant les points (0,0) en haut à gauche de l'image. Ensuite, après avoir défini les coordonnées de l'emplacement initial du curseur, il est possible de déplacer le curseur, soit relativement à la position actuelle, soit en indiquant une nouvelle position, le tout avec seulement des coordonnées. Il est évidemment possible d'indiquer le type de mouvement réalisé, et s'il inclut le tracé d'une ligne ou non.

Fig. 7 - Exemple de fichier SVG

```
<?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>
```

Fig. 8 - Résultat de l'exemple précédent

d) Description d’un .JSON de Scratch

Parallèlement à mon étude sur les .svg, j’ai également étudié comment étaient décrits les différents blocs et leur architecture dans un fichier .sb3, réalisant encore une fois du reverse engineering. Cependant, un fichier .sb3 ne contient pas lui-même les informations sur les blocs dont j’ai besoin. Il représente en fait un dossier compressé et renommé .sb3. C’est à l’intérieur de ce dossier que, parmi d’autres fichiers, se trouve un .json du nom de “project.json”. Il possède la description des différents blocs présents dans l’algorithme, ainsi que toutes les informations liées à ces blocs (Figure 9). Dans ce .json, seule la liste des blocs est utile. Chaque bloc est identifié par son nom et par ses attributs. Les attributs “next” et “parent” permettent de décrire la position des blocs relative aux autres. L’attribut “opcode” représente quant à lui l’identifiant de l’action effectuée par le bloc. En effet, tous les blocs ayant le même opcode vont réaliser la même action. Cependant, certains blocs peuvent avoir des valeurs en argument, comme le bloc avec l’opcode “motion_movesteps” (Figure 10) qui prend dans l’exemple la valeur 10. Cette valeur va être utilisée pour réaliser l’action indiquée par le bloc, et elle se situe entre guillemets dans les attributs “inputs” ou “fields” du bloc, en fonction du type d’entrée de la valeur. Pour finir, certains blocs comme les blocs de contrôle ou les blocs de procédure (blocs personnalisés) peuvent avoir un sous-ensemble de blocs qui leur est propre. Le nom du premier bloc de ce sous-ensemble est donc décrit dans l’espace “SUBSTACK” des “inputs”, comme on peut l’observer dans la Figure 9.

Fig. 9 - Exemple de description d’un bloc dans projet.json

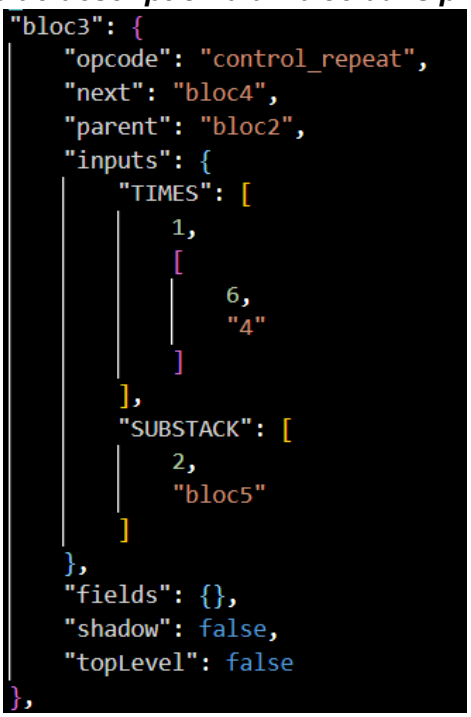
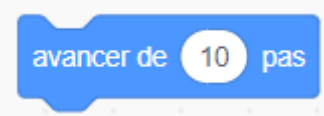


Fig. 10 - bloc motion_movesteps



e) Réalisation du convertisseur

Après avoir appris à lire et écrire les .sb3 et .svg, il était désormais temps de créer le convertisseur. J'ai décidé de faire cela en Python, car ce langage fournit de nombreuses fonctionnalités aussi bien en lecture/écriture de fichiers qu'en analyse de texte et en calcul. De plus, l'organisation que j'ai effectuée pour le créer est relativement simple : après avoir décidé de la création du convertisseur, l'objectif était d'ajouter une nouvelle fonctionnalité avant chaque réunion de groupe. Une réunion était organisée toutes les 1 à 2 semaines et, à chaque réunion, je pouvais apprendre quelle fonctionnalité ajouter avant la prochaine.

Pour me lancer dans la création du convertisseur, j'ai tout d'abord commencé par écrire les méthodes de lecture du .json et d'écriture du .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, afin de donner le SVG résultant ("result.svg"). Par la suite, j'ai choisi de parcourir l'ensemble des blocs décrits dans le JSON grâce à du pattern matching. Pour chacun d'entre eux, je crée un objet de type Bloc regroupant toutes les caractéristiques utiles du bloc, comme le nom du bloc suivant, ses valeurs associées, etc., avant de les ajouter dans un dictionnaire qui associe à chaque nom de bloc son objet décrivant ses caractéristiques. Cette étape permet de faciliter le parcours de l'algorithme.

Pour continuer, j'ai créé une méthode permettant de parcourir l'ensemble des blocs de l'algorithme en commençant par le premier et en continuant jusqu'à arriver sur un bloc n'ayant pas de bloc suivant. Chaque bloc est analysé et l'action décrite par son opcode est alors réalisée. Bien évidemment, initialement, peu de blocs pouvaient être reconnus par le convertisseur : il y avait seulement les blocs pour poser et relever le stylo, ainsi que ceux permettant d'avancer ou de changer d'orientation. À la fin de l'analyse de chaque bloc, une chaîne de caractères était générée, contenant la valeur nécessaire à écrire dans le SVG pour réaliser l'action. Les seuls blocs qui pouvaient ajouter des lignes dans le SVG étaient les blocs de mouvement. Ceux-ci étaient donc ma priorité pour ajouter le traitement dans mon convertisseur.

Après avoir implémenté l'ensemble des blocs de mouvement possibles, j'ai ensuite ajouté les fonctionnalités des blocs de contrôle, qui initient l'analyse d'un

sous-ensemble de blocs. Le parcours du sous-ensemble est réalisé de la même façon que le parcours global, je pouvais donc réutiliser la fonction écrite précédemment pour le faire. Cependant, certains blocs de contrôle doivent suivre des conditions pour parcourir leur sous-ensemble, et ces conditions peuvent être dictées par des calculs. J'ai donc ajouté par la suite un système permettant de détecter les blocs de calculs (blocs opérateurs) et de trouver la valeur résultante. Il me fallait également ajouter un système de variables semblable à celui présent sur Scratch. En effet, sur Scratch, il est possible d'initialiser une variable avec un nom donné, et de modifier sa valeur, ou utiliser sa valeur dans l'input d'un autre bloc. Par conséquent, j'ai permis cela en même temps que l'ajout des calculs et des conditions, pour obtenir finalement un code compatible avec tous les blocs de contrôle existants, ainsi que la majorité des blocs opérateurs.

Finalement, pour ce qui est de l'analyse des blocs, la dernière fonctionnalité ajoutée fut l'utilisation de blocs personnalisés. Un bloc personnalisé permet de définir un sous-ensemble de blocs qui peuvent être appelés à tout moment dans l'algorithme principal. Comme leur définition dans le .json est différente de celle des blocs classiques, il a fallu créer de nouveaux cas de pattern matching, mais le parcours du sous-ensemble était identique à celui des blocs de contrôle.

f) Adaptation ivy Bus

Une fois le SVG pouvant être généré, une adaptation peut être réalisée pour l'exécution du code. En effet, il ne faut pas oublier que ce convertisseur est créé pour être utilisé avec le .sb3 généré par TaBGO. Néanmoins, pour l'instant, il faut tout d'abord exécuter le code TaBGO, puis ensuite exécuter le convertisseur afin qu'il génère le SVG. Cela peut se révéler difficile pour un professeur, et encore plus pour un enfant malvoyant. Ainsi, M. Philippe Truillet nous a exposé un projet disponible sur GitHub : le bus Ivy. Ivy est une bibliothèque pouvant être ajoutée à la majorité des langages de programmation. Une fois connecté au bus Ivy, le programme peut alors envoyer des messages et/ou s'abonner à différents types de messages, le tout sans avoir à passer par l'intermédiaire d'un serveur. Cela permet de faire communiquer facilement les programmes entre eux, peu importe leur langage de programmation.

L'objectif ici était de relier le code TaBGO avec mon convertisseur grâce à un bus Ivy. Pour cela, il faut importer la librairie sur Processing, ainsi que sur Python, et faire en sorte que TaBGO envoie un message sur le bus lors de la création du fichier .sb3. Ce message sera immédiatement reçu par le convertisseur qui s'occupera alors de générer le SVG résultant du .sb3 créé par TaBGO. Ainsi, le SVG peut être récupéré directement après la génération du fichier .sb3.

g) Différents types de rendu

Après m'être assuré que mon convertisseur acceptait aussi bien les .sb3 générés par TaBGO que les .sb3 générés par Scratch lui-même, une nouvelle idée de retour d'expérience est apparue. Notre référent du projet, M. Philippe Truillet, avait accès à des Ozobots. Les Ozobots sont de petits robots suiveurs de ligne qui font du bruit et peuvent lire certaines combinaisons de couleurs sur leur ligne et effectuer l'action associée à la couleur. Dans cette optique, nous avons eu l'idée de faire rouler le robot sur les lignes générées par le SVG. Par conséquent, l'Ozobot pourra suivre le dessin et son mouvement pourra, entre autres, être entendu par les personnes malvoyantes. Afin de rendre cela possible, il faut tout d'abord épaissir la ligne sur le SVG, ce qui était possible grâce à l'ajout d'un simple attribut. J'ai ensuite ajouté la possibilité de modifier l'échelle du dessin afin d'augmenter la distance entre les lignes pour que le robot puisse les suivre plus librement. Enfin, le robot pouvait se déplacer dans le dessin comme prévu, mais il ne savait pas quoi faire lorsqu'il était confronté à la fin d'une ligne. La résolution de ce problème fut la tâche la plus difficile : j'ai dû ajouter les couleurs rouge et bleue à chaque extrémité d'une ligne. Une extrémité se trouve au début et à la fin du dessin, ou lorsque le stylo est levé ou baissé. Changer la couleur dans un SVG n'était pas le plus compliqué, mais calculer les coordonnées du premier et du dernier mouvement, et aller dans le sens inverse pour créer les extrémités rouges et bleues, était un grand défi, car de nouvelles coordonnées et orientations devaient être calculées pour chaque bloc déplaçant le curseur. Cependant, avec beaucoup de détermination, le travail a finalement été accompli et le robot peut maintenant faire demi-tour et poursuivre sa route.

5) Bilan personnel

Sur le plan organisationnel, devoir ajouter une nouvelle fonctionnalité avant chaque réunion fut une méthode très efficace pour moi, qui m'a permis de progresser régulièrement sur le projet sans pour autant me sentir bouleversée. Cela eut l'effet d'un casse-tête hebdomadaire à résoudre qui n'était pas déplaisant. Un avantage à avoir une partie assez indépendante du travail des autres membres du groupe est que je pouvais avancer à mon rythme, bien que parfois nos parties se chevauchaient, comme par exemple les tests effectués sur le .sb3 généré par TaBGO, qui devaient placer des valeurs spécifiques pour chaque bloc, une partie dont Loan Vigouroux était chargé d'implémenter. Cela n'a pas causé de frein et a favorisé les échanges et les propositions entre nous.

Dans l'ensemble, ce projet a été une grande source d'expérience pour moi, à la fois dans les domaines du travail d'équipe et de l'apprentissage autonome. Un

premier défi pour nous tous a été de commencer sur un projet déjà existant, d'appréhender les problèmes et les objectifs, et de comprendre le code déjà créé que nous devions utiliser. Comme nous ne nous connaissions pas très bien, nous avons dû nous familiariser avec les habitudes de travail de chacun et trouver des moments où nos plannings étaient alignés afin d'organiser des réunions régulières. Cependant, malgré ces difficultés initiales, chacun a joué son rôle et a participé d'une manière ou d'une autre à ce projet. Les réunions ont été très utiles et je suis reconnaissante à M. Philippe Truillet d'avoir donné de son temps pour nous rencontrer régulièrement et nous aider à nous orienter dans la bonne direction.

En termes d'opportunités d'apprentissage, ce projet m'a permis de m'améliorer dans de nombreux domaines. J'ai acquis des connaissances sur les fichiers .svg ainsi que sur les fichiers .sb3 et .json, et sur les différentes méthodes d'accessibilité pour les personnes malvoyantes. De plus, j'ai fait des progrès significatifs dans les domaines des regex et du pattern matching, lorsque j'ai dû décrypter les différents blocs et leurs propriétés depuis le fichier .json. J'ai également révisé les mathématiques et la trigonométrie, où j'ai dû résoudre des problèmes mathématiques intéressants. Enfin, j'ai également acquis de l'expérience en matière de rédaction de rapports et de présentation orale, M. Truillet nous aidant continuellement et nous donnant des conseils pour notre présentation finale.

Pour conclure, je garde finalement une note très positive de ce projet qui, j'espère, pourra être utilisé dans des cas concrets et pourra servir pour le bien commun.