Captation Automatisée et Visualisation de Traces de Programmation dans un Environnement de Programmation Graphique par Blocs

Jean-Marc Legrand

Centre de Recherche en Éducation de Nantes, Université de Nantes, mars 2019

Résumé L'entrée d'une part d'algorithmique et de programmation dans les programmes de l'école élémentaire et du collège en 2016 (Eduscol, 2016), avec une incitation institutionnelle à utiliser des environnements de programmation graphique par blocs de type `Scratch`[1], `mBlock`[2], `Phratch`[3], ou encore `Snap`[4], entraîne un besoin d'analyses didactiques nouvelles, pour de nouvelles situations dans un environnement nouveau.

Afin d'étudier la construction de concepts algorithmiques et informatiques chez les élèves, ou la mobilisation de ces concepts, l'analyse de la production finale — le programme construit —, même complétée par une analyse des interactions langagières, s'avère insuffisante. Les liens entre modification et test, ainsi qu'entre résultat et ajustement, doivent ainsi être élucidés afin de déterminer les régularités, les obstacles ou éléments facilitants de la situation mise en œuvre.

Dans cette optique, nous avons construit un outil permettant de capter, transcrire et visualiser les traces de programmation des élèves dans un environnement de programmation graphique par blocs (EPGB). Après une description rapide des EPGB, nous décrirons cet outil en donnant quelques exemples de premières analyses.

1 Les environnements de programmation graphique par blocs

Les descriptions évoquées dans cette partie sont communes à tous les EPGB.

1.1 Organisation générale

Un EPGB est un logiciel permettant à la fois de développer et d'exécuter un programme conçu dans un « langage entièrement visuel et fonctionnant donc par manipulation gestuelle » (Komis et al., 2017). La construction d'un programme se fait par « glisser-déposer » des instructions disponibles. En reprenant le vocabulaire utilisé par Komis et al. (2017), une instruction est un *bloc*. L'assemblage

1. https://scratch.mit.edu/
2. http://www.mblock.cc
3. http://www.phratch.com/
4. https://snap.berkeley.edu/

(la juxtaposition verticale) de blocs forme un *script* exécutable. Plusieurs scripts peuvent coexister, formant ainsi un *programme*. Les blocs ne peuvent s'emboîter que de façon logique, en raison de leurs formes, ce qui élimine les messages d'erreurs(Sáez-López et al., 2016), ou du moins limite les risques d'erreur lexicale, syntaxique ou sémantique.

Une instruction peut être générale (

), ou spécifique (

). Ces blocs, plus généralement, peuvent aussi être des variables, des opérateurs (

), des procédures, des structures de contrôles. Les instructions spécifiques nécessitent un ou plusieurs paramètres qui, dans les EPGB, sont initialisées avec une valeur par défaut. Ces valeurs peuvent être modifiées par une entrée clavier, ou en glissant-déposant soit des opérateurs soit des blocs représentant le contenu d'une variable.

1.2 Interface graphique

L'interface graphique est constituée d'une seule fenêtre divisée en différentes zones aux fonctionnalités différentes : une zone de catégories de commandes, une zone de palette de commandes (correspondant à la catégorie choisie), une zone de programmation dans laquelle on dépose les blocs, une zone d'exécution permettant de visualiser le résultats de l'exécution, ainsi que d'autres zones permettant de modifier les *personnages*, la *scène* ou encore les *costumes*. Un EPGB est à la fois support de programmation et fournisseur de rétroaction.

espace d'exécution La zone d'exécution est un espace essentiellement d'affichage graphique : déplacement des *personnages*, formes dessinées (à la manière de LOGO), image de fond ou images de costumes « tamponnés ». C'est aussi une zone d'interaction avec l'utilisateur : les *personnages* peuvent être cliqués, des instructions permettent d'afficher des messages (sous forme de phylactères), une instruction particulière permet de solliciter une entrée de la part de l'utilisateur (en affichant un message) tout en stockant cette valeur dans une variable prédéfinie (*réponse*). Enfin, le contenu des variables existantes peut être affiché ou non, par une instruction spécifique ou par action dans la zone de commande. Le contenu des variables affichées peut être directement modifié dans la zone de programmation (ce qui n'est pas sans conséquences didactiques).

1.3 Exécution des scripts

Les scripts peuvent être exécutés soit en réponse à un événement (touche enfoncée, « drapeau vert » cliqué...), soit par un clic direct sur un des blocs du script.

2 Description de l'outil de captation

Dans le cadre d'une recherche visant notamment à étudier les possibilités d'exploitation des liens entre pensée algorithmique et pensée algébrique, nous

avons conçu une situation visant à utiliser l'apprentissage de concepts d'algorithmique et de programmation afin d'aider des élèves de cycle 4 à entrer dans l'algèbre élémentaire. Afin d'analyser finement cette situation, nous avions besoin de compléter les transcriptions d'interactions langagières entre les élèves des binômes étudiés par l'analyse du processus de construction ou de modification du programme en jeu. L'analyse de la construction d'un programme par des élèves utilisant un EPGB nécessite de transcrire une certaine catégorie d'*évènements de programmation*, que nous définirons comme étant « des phénomènes considérés comme localisés et instantanés, survenant dans un EPGB à un instant bien déterminé ». Ainsi, l'ajout d'un bloc est un événement de programmation issu d'une action de l'utilisateur, l'affichage d'un résultat étant quant à lui un évènement interne au programme. Notons qu'une même action peut déclencher des événements différents, en fonction du contexte.

On trouve déjà dans `Pixel'Art` un enregistrement des traces de l'activité de l'élève(Declercq and Tort, 2018). Nous proposons ici un système de captation plus général en instrumentant `Snap`.

2.1 Objectifs

L'objectif principal de cet outil est de récolter automatiquement les actions et évènements de programmation produits par les élèves afin de rechercher des motifs, des régularités, des traces de problématisation (Orange, 2012) (ou la manifestation des ces traces). Ce recueil doit bien sûr être directement associé à un élève ou un groupe d'élève travaillant sur un même poste, ce qui implique leur identification.

Les évènements récoltés doivent être d'une granularité suffisamment fine et être catégorisés pour un traitement plus global. Ils doivent aussi bien entendu être stockés pour des manipulations et analyses ultérieures. Afin de couvrir l'ensemble des éléments hypothétiquement pertinents, ces données doivent nous permettre de :

- visualiser les changements, les modifications apportées au programme,
- visualiser les essais testés par les élèves : le moment des essais, les valeurs testées, les réactions (arrêt, modifications) aux rétroactions,
- visualiser les résultats obtenus lors de l'exécution des programmes,
- reconstruire l'état complet du programme à un instant t quelconque (soit reconstruire l'*histoire du programme*).

En outre, nous souhaitons que l'utilisation de cet outil n'entraîne pas de difficultés supplémentaires pour l'enseignant-expérimentateur, ni pour les élèves. De plus, cet outil doit être le moins invasif possible afin de pouvoir être utilisé aisément dans des environnements très variés (matériels et réseau).

Ainsi, nous avons développé un dispositif permettant de :

- identifier les utilisateurs [5],

5. Dans les situations étudiées, l'« utilisateur » est soit l'enseignant, soit un élève, soit un binôme.

— distribuer une version instrumentée d'un EPGB (en l'occurence `Snap`), permettant de capter les évènements de programmation,
— récolter et stocker l'événement capté, ainsi que l'utilisateur à l'origine ou destinataire de l'événement, et le moment de son déclenchement (relativement au lancement de l'EPGB),
— permettre à l'enseignant-expérimentateur de distribuer aisément un « programme de base » aux élèves,
— permettre à cet enseignant ou au chercheur de consulter le programme final de l'élève, afin éventuellement de le présenter, le discuter voire le modifier en groupe classe,
— visualiser les données stockées, suivant les besoins de l'analyse.

2.2 Événements captés

Les événements de programmation captés sont classés en trois catégories :

1. modification de l'environnement,
2. modification de l'état du programme,
3. modification de la structure du programme.

Modification de l'environnement : nous considérons comme évènement de modification de l'environnement toute action de l'utilisateur modifiant l'environnement global de l'interface.

Ainsi, nous enregistrerons les évènements de lancement de l'interface, de création/chargement/sauvegarde de programmes, de modification de l'affichage, de navigation dans les catégories de blocs ou encore de changement quant à l'affichage ou non du contenu d'une variable. Nous enregistrerons aussi certaines actions effectuées avec la souris.

Modification de l'état du programme : Nous entendons par « modification de l'état du programme » tout changement de l'état d'exécution d'au moins un des scripts le composant.

Ainsi, nous prendrons en compte les actions déclenchant le lancement (l'exécution) du programme ou de seulement l'un de ses scripts, en précisant la liste des processus — scripts — en cours, `Snap` mettant en œuvre un pseudo-parallélisme entre les scripts exécutés. Ce fonctionnement peut s'avérer problématique : il n'est pas rare de voir les élèves lancer l'exécution du « programme », c'est-à-dire le script lancé lors de l'appui sur le « drapeau vert » (

), puis de déclencher, de manière fortuite ou non, l'exécution concurrente d'un autre script, par exemple en cliquant sur un des blocs, alors que l'action visée était de déplacer ce bloc. De même, les actions mettant le programme en pause [6], ou le

6. Une des raisons motivant le choix de `Snap` étant que cet EPGB dispose d'un mode « pas-à-pas » très utile pour le débogage — et évitant ainsi la multiplication des 

. Les autres raisons étant l'existence d'un historique des déplacements de bloc, la possibilité de déterminer les instructions visibles pour les élèves, et une modification aisée des sources de l'EPGB.

terminant, seront enregistrées. Les terminaisons non déclenchées par l'utilisateur (terminaison normale du programme, ou arrêt sur une erreur) le seront aussi. Enfin, l'attente d'une entrée de la part de l'utilisateur ainsi que la validation d'une entrée seront aussi captées.

Certains de ces changements d'état du programme seront accompagnés d'une capture de la zone d'exécution, qui est la partie visible du programme pour l'utilisateur. Ceci est nécessaire notamment si l'on veut identifier la rétroaction visuelle du programme qui aurait pu causer l'arrêt manuel par l'utilisateur ou une modification résultante.

Modification de la structure du programme Toute action de l'utilisateur modifiant l'implantation de l'algorithme est considérée comme étant une modification de la structure du programme.

Actuellement, nous captons les actions suivantes :

— le chargement ou la création d'un nouveau programme,
— la création, le déplacement, la suppression, la copie des blocs,
— la navigation dans l'historique (revenant à des déplacements ou suppressions de blocs),
— la création, la suppression, le renommage de variables locales ou globales,
— le changement d'un paramètre d'une instruction, que ce soit la modification d'une valeur ou d'une expression,
— le déplacement, l'insertion ou la suppression de blocs renvoyant une valeur.

2.3 Événements non pris en compte

Concernant la version actuelle du dispositif, certains événements ne sont pas considérés, étant accessoires sur les situations étudiées. Il en est ainsi des modifications des *costumes*, des *personnages* ou des *scènes*, de même que toute action impliquant la production ou la modification de sons. Nous ignorons aussi les déplacement manuels d'un *personnage*.

En outre, pour des raisons de difficultés techniques non encore résolues, la modification par les élèves de procédures créées (script définissant un *nouveau bloc*) est bloquée.

D'autres limitations existent aussi : par exemple la position des blocs n'est pas renseignée, ce qui ne permet pas de visualiser la disposition des blocs effectuée par les élèves. Les différents programmes associés à plusieurs *personnages*, sont assimilés à un seul programme. Toutes ces limitations sont susceptibles d'être levées ultérieurement.

3 Exemples de visualisations et débuts d'analyse

À ce jour, quatre types de visualisations ont été mise en œuvre :

1. reconstitution de l'histoire du programme pour un utilisateur,

2. représentation de la variation du nombre d'instructions (par programme et par scripts, pour un utilisateur),
3. recherche de l'occurrence d'un événement (apparition d'une structure de boucle) et représentation du nombre d'événements ayant précédé l'apparition de cet occurrence, pour un ou plusieurs utilisateurs,
4. représentation de la structure organisationnelle globale de la construction du programme, pour un ou plusieurs utilisateurs.

À ceci s'ajoute la possibilité — pour le moment manuelle — d'établir des statistiques concernant une classe d'événements, par exemple en cherchant le ratio moyen « lancement par événement »/« lancement par clic ».

3.1 Reconstitution de l'histoire du programme

Élément essentiel de l'analyse, cette visualisation (fig.1) représente l'évolution de la structure du programme en fonction du temps sur l'axe horizontal, les différents scripts formant le programme sur l'axe vertical. Pour chaque évènement, son type est indiqué – éventuellement avec des précisions —, ainsi que le moment de l'apparition de l'événement relativement au lancement de l'EPGB. Ce moment peut-être ajusté en précisant un décalage (« offset ») afin de faciliter la synchronisation avec une éventuelle captation audio et/ou vidéo.

Les éléments utiles pour l'analyse, même s'ils ne modifient pas la structure du programme, sont aussi représentés : lancement / pause / arrêt des scripts (ainsi que le mode de lancement ou de terminaison), entrée utilisateur, affichage ou non du contenu d'une variable. Chaque changement d'état du programme est accompagné d'une capture d'écran de la zone d'exécution.

Afin de faciliter l'analyse, le script destinataire d'une modification est mis en valeur, et il est possible de n'afficher que ces scripts. Chaque script est identifié par un entier qui est l'identifiant du bloc de tête de ce script, ce qui permet notamment d'identifier le script exécuté ou stoppé. Les modifications apportées au script ont des représentations diverses : en rouge un changement lié à une ou plusieurs instructions (ajout, déplacement etc...), dans un cartouche bleu-canard les modifications des paramètres des instructions. Une icône est éventuellement rajoutée pour préciser l'origine d'une modification ou les blocs dupliqués ou déplacés.

Une analyse succincte des premières transcriptions à fait apparaître certaines régularités. Ainsi on retrouve des éléments concordants avec les observations de Briant (2013) dans un autre cadre : lors de la construction d'un programme nécessitant une variable dans un rôle de compteur (Sajaniemi, 2002), un groupe a ainsi testé 

, s'attendant à ce que l'ordinateur résolve ensuite cette tâche. Autre régularité, lors de l'analyse d'une phase de généralisation d'un algorithme, plusieurs groupes passent par un opérateur avant d'aboutir à la nécessité de la variable dans un rôle de valeur fixe(Sajaniemi, 2002) : `répéter 5 fois` devient `répéter (6-1)` fois. Tout se passe comme si les élèves considéraient que, le nombre d'itérations devant changer pour s'adapter à la demande de l'utilisateur, un calcul pouvait "changer" des valeurs en d'autres valeurs.

3.2 Variation du nombre d'instructions

La variation du nombre d'instructions de chaque script formant le programme est visualisée (fig.2) sous forme de courbes empilées, l'axe horizontal étant là encore l'axe temporel des événements. Dans le cas d'un événement modifiant la structure du programme, il est possible d'accéder au contenu de chaque script au moment de cet événement ; on réutilise dans ce cas la reconstitution de l'histoire du programme vue précédemment.

Cette représentation a pour objectif d'aider à repérer certains moments significatifs dans la construction du programme. Ainsi, une suppression massive d'instructions peut correspondre à une prise de conscience par les élèves de l'invalidité de leur script — ou d'une de ses parties — et des hypothèses qui ont contribué à sa construction : ils annulent tout ou partie des modifications apportées. Cette suppression massive peut aussi indiquer que les élèves ont trouvé une solution plus efficace ou plus économique, remplaçant par exemple une répétition de motifs par une boucle.

3.3 Recherche de la première occurrence d'un événement : la boucle

L'objectif est ici d'indiquer le nombre d'événements ayant précédé l'apparition d'un bloc correspondant à une boucle, en agrégeant plusieurs sessions. Cette représentation (fig.3) a été mise en place dans le cadre d'une expérimentation visant à étudier le moment et les conditions d'apparitions de la nécessité économique de la boucle (Ada Lovelace, citée par Nguyen (2005)) dans un EPGB. On peut représenter plusieurs sessions simultanément.

Les types d'événements pris en compte peuvent être modifiés — on modifie ainsi la granularité —, et le nombre d'instructions composant le programme juste avant l'apparition de la boucle est indiqué.

On a ainsi pu constater que certains élèves n'ont pas construit cette nécessité, malgré les 638 instructions qu'ils ont utilisées pour représenter des motifs répétés.

Il est évidemment possible d'adapter cette représentation afin de rechercher des occurrences d'autres événements (utilisation d'une instruction particulière, d'une valeur particulière, d'un type d'opérateur booléen etc...)

3.4 Structure organisationnelle globale

Il s'agit de visualiser (fig.4) l'instrumentalisation de l'outil par les élèves. L'EPGB modifié fourni simplifie la récupération des programmes servant de base pour une séance, de même que les sauvegardes. Nous représentons sous forme de graphe les événements de lancement de session (en violet), de chargement d'un programme (orange) et de sauvegarde du programme (vert). Les événements sont représentés par les nœuds du graphe, placés verticalement dans l'ordre de leur manifestation. La distance entre deux nœuds consécutifs est proportionnelle au temps séparant les deux événements. On a aussi accès, pour chaque événement répertorié, à l'évènement précédent et éventuellement à la capture d'écran de la

zone d'exécution qui lui est liée, ce qui peut aider à donner une raison à une sauvegarde (validation) ou un nouveau chargement (invalidation).

On peut ainsi observer que certains élèves ont une structure de construction très linéaire : un chargement en début de séance, une sauvegarde une fois le résultat cherché obtenu. D'autres se sont appropriés l'outil en effectuant un nouveau chargement à de nombreuses reprises : une voie explorée étant invalidée, les élèves ont préféré re-charger le programme de base de la séance plutôt que de déconstruire leurs scripts.

4 Conclusions et perspectives

L'outil de captation automatisée et de visualisation de traces de programmation dans un EPGB que nous avons présenté ici est opérationnel — même s'il n'est pas encore complet — et a été testé auprès de plusieurs classes avec 3 enseignants. Ces enseignants ont pu utiliser notre outil de façon autonome, sans notre présence. En outre, certaines histoires reconstituées ont été présentées lors d'un atelier auprès de chercheurs et étudiants découvrant l'outil et ses visualisations pour la première fois. Après un petit temps d'adaptation, des informations pertinentes ont pû être extraites quant au cheminement des élèves au sein du problème. D'autre part, s'il permet une étude qualitative, complétée avec des captations audio et vidéo (cette analyse est en cours), des études quantitatives sont aussi possibles — à ce jour sur environ 250 sessions correspondant à 100000 événements.

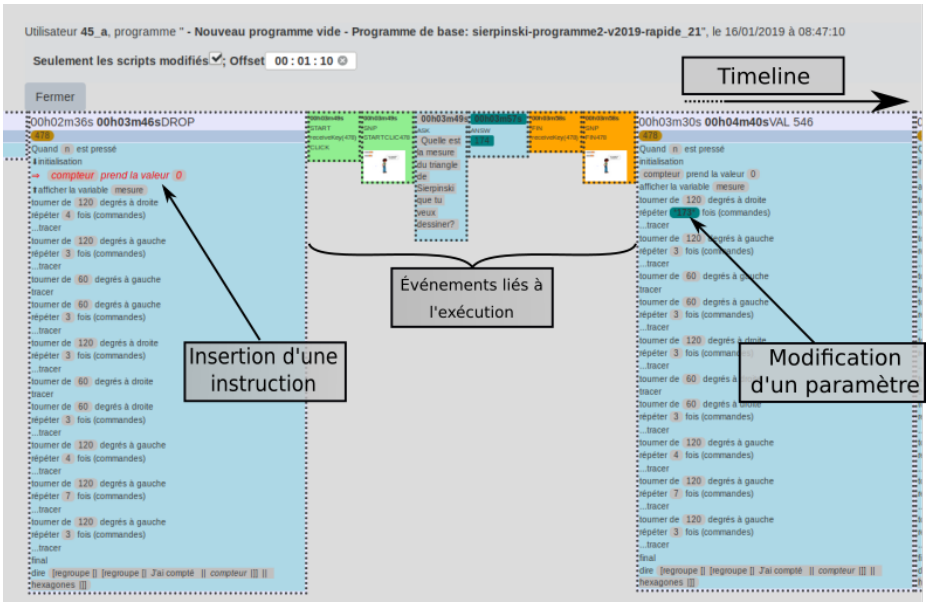

Figure 1. Reconsitution de l'histoire d'un programme

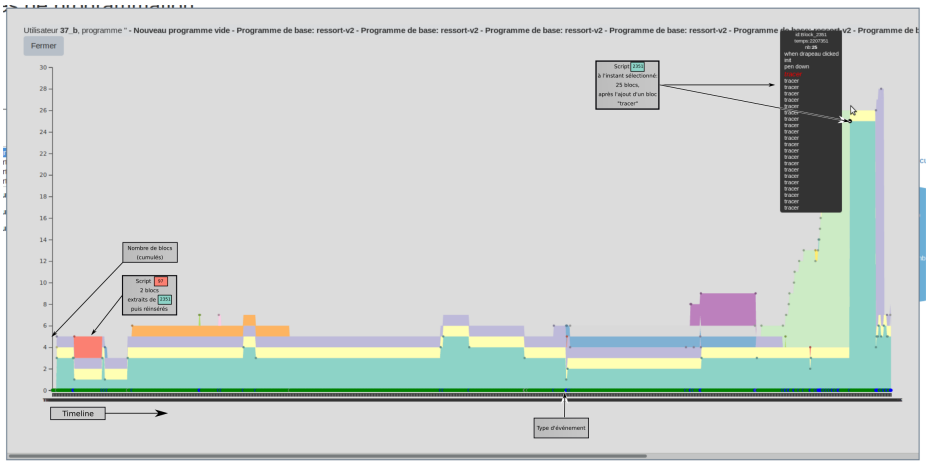

Figure 2. Variation du nombre d'instructions

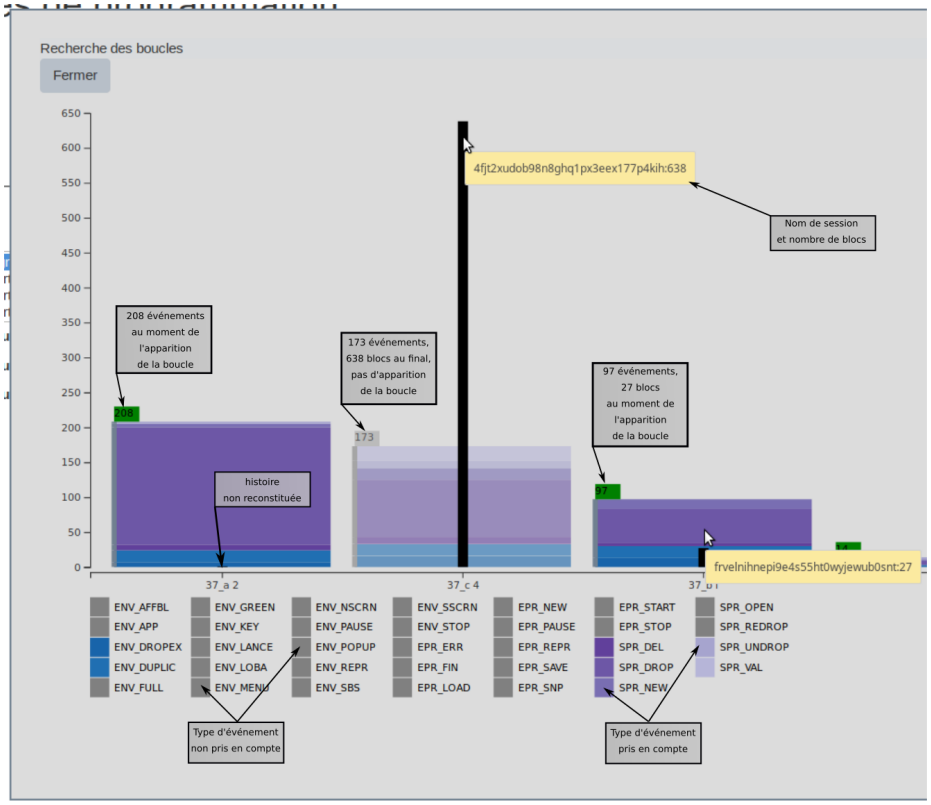

Figure 3. Recherche de la première occurrence d'une boucle

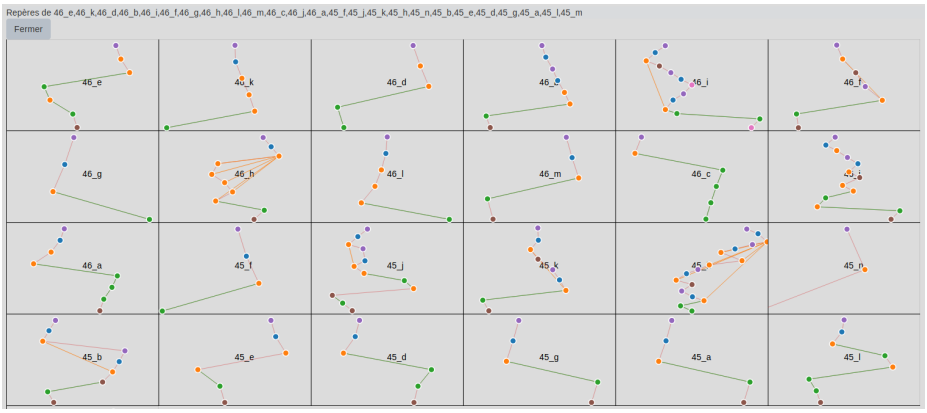

Figure 4. Structure organisationnelle globale

Bibliographie

Briant, N. (2013). *Étude Didactique de La Reprise de l'algèbre Par l'introduction de l'algorithmique Au Niveau de La Classe de Seconde Du Lycée Français*. Theses, Université Montpellier II - Sciences et Techniques du Languedoc.

Declercq, C. and Tort, F. (2018). Organiser l'apprentissage de la programmation au cycle 3 avec des activités guidées et/ou créatives. In *RJC EIAH 2018*, Besançon, France.

Eduscol (2016). Algorithmique et programmation.

Komis, V., Touloupaki, S., and Baron, G.-L. (2017). Analyse cognitive et didactique du langage de programmation ScratchJr. In Henry, J. E. ., Nguyen, A., and Vandeput, E., editors, *L'informatique et Le Numérique Dans La Classe : Qui, Quoi, Comment ?*, Informatique, pages 109–121. Presses Universitaires de Namur, Namur.

Nguyen, C. T. (2005). *Etude Didactique de l'introduction d'éléments d'algorithmique et de Programmation Dans l'enseignement Mathématique Secondaire à l'aide de La Calculatrice*. Theses, Université Joseph-Fourier - Grenoble I.

Orange, C. (2012). *Enseigner Les Sciences ; Problèmes, Débats et Savoirs Scientifiques En Classe*. Le Point Sur... Pédagogie. de Boeck.

Sáez-López, J.-M., Román-González, M., and Vázquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school : A two year case study using "Scratch" in five schools. *Computers & Education*, 97 :129–141.

Sajaniemi, J. (2002). An empirical analysis of roles of variables in novice-level procedural programs. In *Proceedings IEEE 2002 Symposia on Human Centric Computing Languages and Environments*, pages 37–39, Arlington, VA, USA. IEEE Comput. Soc.