Zoom sur quelques erreurs récurrentes lors des premiers apprentissages en algorithmique

7

Marielle Léonard[1,2]

1. Université de Lille, ULR 4354, CIREL - Centre Interuniversitaire de Recherche en Éducation de Lille, 59000 Lille, France

2. Université de Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, 59000 Lille, France

L'apprentissage des bases de l'algorithmique est abordé à l'école obligatoire notamment en utilisant des environnements de programmation par blocs comme Scratch. Une des approches possibles pour cet apprentissage est la résolution de puzzles, c'est-à-dire de problèmes fermés dont la tâche est prescrite et dont l'évaluation est effectuée automatiquement par le système, déterminant si chaque solution soumise est valide ou non (Pelánek et Effenberger, 2022). Nous étudions les premiers apprentissages en algorithmique à travers la confrontation d'élèves de 7 à 15 ans avec des puzzles de programmation impliquant de commander un robot virtuel sur une grille, en langage Scratch.

L'enjeu de cette contribution est d'identifier des erreurs récurrentes et leur conceptualisation sous-jacente lors des premiers apprentissages en algorithmique, afin notamment de fournir des repères conceptuels aux enseignants en charge de l'initiation à ce domaine. Notre démarche repose sur la collecte de traces lors de l'activité des élèves dans l'environnement de programmation, et sur un processus d'analyse de ces traces d'interaction associant traitement automatisé et travail d'expert ancré dans le cadre d'analyse de la théorie des champs conceptuels (Vergnaud, 1991). Ce cadre théorique, qui sera présenté plus en détails dans la section « Cadre théorique », vise à comprendre la conceptualisation notamment dans le cas des activités cognitives complexes dont la programmation informatique fait partie. Une difficulté de ce travail d'analyse concerne le volume important des traces collectées, qui exclut une exploration exhaustive à la main. Il est nécessaire de se doter d'une stratégie pour combiner analyse qualitative d'expert et automatisation du traitement de données. Nous présentons cette stratégie et montrons le type de résultat qu'il est possible d'obtenir, en prenant un exemple représentatif qui traite de la notion de boucle (figure 1).

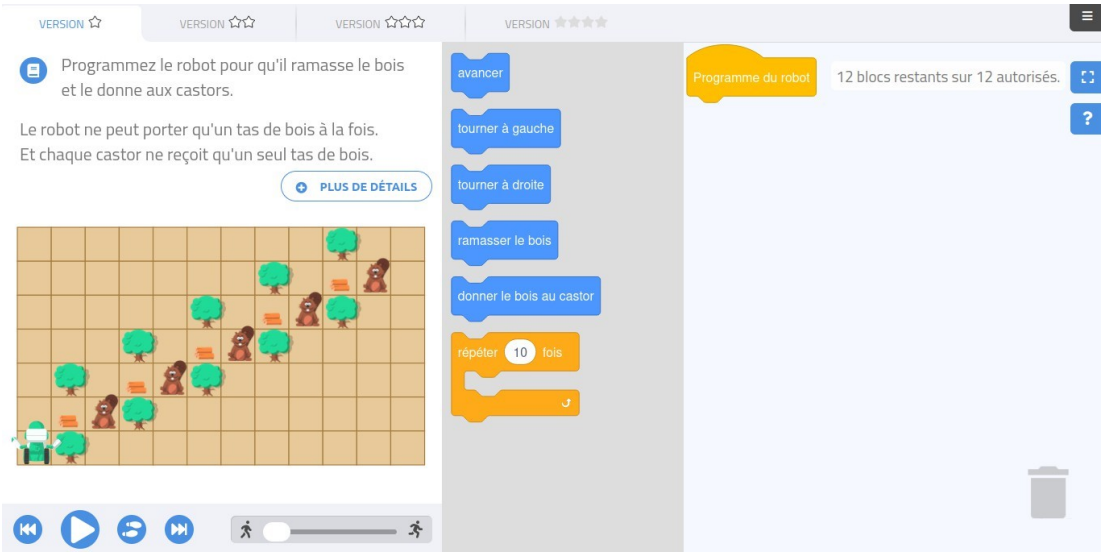

Figure 1 – Exemple de puzzle de programmation en langage Scratch dans l'environnement Algoréa : « Distribuer le bois aux castors » en version 1 étoile. © France-ioi.

Cadre théorique

Pour notre étude sur les premiers apprentissages en algorithmique, nous cherchons à nous placer dans un cadre d'analyse qui nous permette d'interpréter les erreurs des élèves et de comprendre leur raisonnement. La théorie des champs conceptuels élaborée par Vergnaud dans le but de comprendre la conceptualisation dans le cas des activités cognitives complexes répond à ce besoin. Cette théorie s'inscrit dans une approche constructiviste et cognitiviste de l'apprentissage. Vergnaud reprend dans ses travaux une large partie de l'approche constructiviste de Piaget, notamment le concept de schème et la théorie de l'adaptation (assimilation, accommodation, équilibration) (Piaget, 1935). En prenant pour unité d'analyse le couple sujet/situation, Vergnaud étudie le sujet lorsqu'il se trouve dans une situation où son but est de réaliser une tâche. Il fait l'hypothèse que toute action finalisée repose sur une « conceptualisation-en-acte », c'est-à-dire que les actions du sujet révèlent son activité cognitive sous-jacente, quand bien même celle-ci n'est pas verbalisée (Vergnaud, 1991).

Dans notre contexte, la tâche finalisée de l'élève est de concevoir un programme en langage Scratch pour que le robot virtuel effectue une certaine mission, formulée dans l'énoncé et évaluée automatiquement lors de l'exécution du programme. Nous faisons l'hypothèse que les traces d'interaction collectées rendent

compte d'une part significative des actions de l'élève sur l'interface, et sont de ce fait révélatrices de ce que Vergnaud nomme conceptualisation-en-acte.

Afin de préciser comment opère cette conceptualisation-en-acte, Vergnaud développe deux axes. Un premier axe est porté par les situations. Le second est porté par le sujet en action, à travers le concept de schème.

Dans le but de comprendre l'activité du sujet, Vergnaud invite à mener en premier lieu une analyse épistémologique des situations auxquelles le sujet est confronté. Cette analyse amène à regrouper les situations singulières en classes de situations suivant la manière dont elles sont traitées par le sujet. Il s'agit d'identifier des paramètres, appelés « variables de situation », qui permettent de différencier des situations proches. Deux cas se présentent suivant que le changement de valeur d'une variable de situation affecte ou non la structure du traitement de la situation par le sujet. Dans le cas où le sujet traite les deux situations de façon similaire, celles-ci appartiennent à la même classe. Dans le cas contraire, deux classes de situations sont distinguées en fonction de la valeur de la variable de situation.

Pour traiter une situation à laquelle il est confronté, le sujet mobilise des schèmes, qui sont des unités d'action déjà prêtes. Vergnaud approfondit le concept de schème introduit par Piaget en en donnant une définition analytique.

Composantes d'un schème selon Vergnaud :

- un but qui est la finalité de l'action ;
- des règles de conduite de l'action, qui se succèdent (règles de prise d'information, règles d'action proprement dite, règles de contrôle de l'action) ;
- des invariants opératoires de deux types, qui constituent la partie cognitive du schème :
  - « concept-en-acte » qui est « un concept tenu pour pertinent dans l'action en situation » (Vergnaud, 2013) ;
  - « théorème-en-acte » qui est « une proposition tenue pour vraie dans l'action en situation » (Vergnaud, 2013) ;
- des inférences, c'est-à-dire des raisonnements en situation qui visent à prendre en compte les spécificités d'une situation et à anticiper les résultats de l'action.

À titre d'exemple, nous renseignons de manière analytique le schème de dénombrement mentionné plusieurs fois dans les travaux de Vergnaud (1991, 2001, 2007).

Schème de dénombrement

- But : « Associer un nombre à une collection discrète » (Vergnaud, 2001).
- Règles de conduite de l'action.
  - Prise d'information : en amont de l'action, identifier les éléments à compter, repérer quelle est l'unité.
  - Action proprement dite : pointer les unités les unes après les autres en leur associant un mot nombre.
  - Contrôle : contrôle au cours de l'action de la coordination entre le regard, le geste de pointage et l'attribution du mot nombre, recomptage éventuel à titre de vérification.
- Invariants opératoires dont certains correspondent aux principes de Gelman (Gelman et Gallistel, 1978).
  - Concepts-en-acte : nombre, correspondance terme à terme, cardinal.
  - Théorèmes-en-acte : « le dernier mot prononcé est le cardinal de la collection », « l'ordre dans lequel on considère les éléments est indifférent ».
- Inférences : identifier le prochain élément à traiter ; maintenir deux ensembles : les éléments déjà pointés et ceux qui ne le sont pas encore.

Du point de vue expérimental, Vergnaud construit la théorie des champs conceptuels à partir d'expérimentations à l'échelle de la classe comme celle sur les structures additives en 1976 (Vergnaud et Durand, 1976). Notre objectif est de nous appuyer sur ce cadre théorique mais avec une méthode de collecte et d'analyse de traces d'interaction qui n'était pas envisageable techniquement au moment de l'élaboration de la théorie des champs conceptuels. Notre objectif est, à partir de ces traces d'interaction collectées dans l'environnement de programmation, de considérer les deux axes préconisés par Vergnaud : classes de situations et mobilisation de schèmes.

Dans notre contexte, les situations sont les puzzles de programmation que le sujet a pour but de résoudre. Nous avons déjà mené une analyse épistémologique pour les situations qui requièrent l'utilisation d'une boucle (bloc Scratch « répéter n fois »). Cette analyse a fait l'objet de deux communications (Léonard *et al.*, 2022a ; 2022b). Nous en reprenons les principaux résultats dans la section « Paliers de difficulté observables à large échelle ». Pour ce qui concerne les schèmes mobilisés par le sujet, une partie est

observable à travers les traces d'interaction de celui-ci avec l'environnement de programmation. Une autre partie, comme les gestes que le sujet pourrait effectuer devant sa machine, ne l'est pas dans le cadre de cette expérimentation, et restera inexplorée dans notre travail. Notre étude des schèmes mobilisés par le sujet comporte trois niveaux d'analyse, à des échelles de plus en plus fines. Nous en structurons la présentation à la manière d'un zoom dans les sections « Mobilisation de schèmes révélée par un indicateur de rapidité normalisée », « Zoom sur la procédure experte et les erreurs récurrentes révélées », « Zoom sur la conceptualisation-en-acte du sujet ». Pour chaque étape, nous présentons quelques résultats obtenus en nous appuyant sur un exemple « fil rouge » (figure 1).

Cadre expérimental et méthodologie

Les puzzles de programmation qui constituent notre cadre expérimental sont issus de l'édition 2022 du concours de programmation Algoréa[1]. La participation au concours Algoréa est organisée en milieu scolaire, sous la forme de sessions en temps limité, d'une durée de 45 minutes. Les élèves progressent dans des catégories de couleur en fonction de leurs résultats. Pour la présente étude, nous ne considérons que les deux catégories les plus faciles (blanche et jaune), dans lesquelles les notions algorithmiques de base sont abordées : séquence d'instructions, répétition, structure conditionnelle, variable et fonction. Les parcours dans ces catégories sont composés de six problèmes. Chaque problème comporte quatre versions de difficulté croissante repérée par des étoiles, de la version 1 étoile qui est la plus facile à la version quatre étoiles qui est la plus difficile. Nous avons donc 24 puzzles par parcours, chaque puzzle correspondant à une situation au sens de Vergnaud. Toutes les situations consistent à programmer des actions et des déplacements d'un robot virtuel sur une grille.

1. Site du concours Algoréa : https://algorea.org

L'objectif est de disposer, pour les mêmes situations, à la fois de données globales sur la population totale et de données locales selon la distinction de Pelanek (*global data / local data*) (Pelánek, 2017). Dans cette optique, nous collectons des données à trois échelles différentes.

— À l'échelle nationale : échantillon global des participations individuelles au concours Algoréa en langage Scratch – de 7800 à 92 400 élèves de 9 à 15 ans suivant la catégorie et le tour. Par exemple, 36 924 élèves ont été confrontés au puzzle « Distribuer le bois au castor » en version 1 étoile de la figure 1.
— À l'échelle de classes : neuf classes de collège (deux 6^{e}, trois 5^{e}, un 4^{e}, trois 3^{e}) – 252 élèves de 11 à 15 ans – qui ont

participé sur une année (deux ou trois sessions par élève). À cette échelle, nous disposons de l'enregistrement de deux types d'évènement horodaté.

  - Lorsqu'un élève lance le programme qu'il a conçu avec des blocs Scratch, celui-ci est transcrit et enregistré dans un format textuel (xml). Pour le puzzle de la figure 1, cela représente 1099 programmes collectés.
  - Les changements de puzzles sont aussi enregistrés, ce qui permet de calculer la durée passée par l'élève sur chaque puzzle.

- À l'échelle du sujet : 20 élèves de 7 à 15 ans (CE1 à la 3e). Pour chacun de ces sujets, nous disposons de l'enregistrement vidéo des trois sessions de l'année 2022 (enregistrement de l'écran et de la voix par Zoom) en plus des traces d'interaction comme à l'échelle précédente.

Ces choix méthodologiques se justifient par la consultation de travaux sur l'analyse de traces d'interaction.

Un premier point concerne le niveau de granularité des données collectées. Parmi les niveaux de granularité (du plus grossier au plus fin) identifiés par Ihantola *et al.* (2015), la collecte de tous les programmes exécutés correspond au deuxième niveau sur six. Ce niveau de granularité est un compromis entre une granularité assez fine pour mener les analyses prévues d'une part, et un volume de stockage et une capacité de traitement de ces données d'autre part. En effet, collecter seulement le programme final (niveau de granularité le plus grossier) contraint à inférer des bribes de processus à partir de celui-ci. En revanche, collecter les programmes intermédiaires permet de rendre compte plus précisément du processus de résolution et des erreurs commises, ce qui est notre objectif. En ce qui concerne la collecte du flux continu d'activité par enregistrement vidéo de l'écran de l'utilisateur, elle ne fait pas partie des niveaux de granularité de Ihantola *et al.* car les auteurs n'ont répertorié que la collecte de données discrètes (évènements).

Un second point est relatif au format de collecte des programmes édités. Si des captures d'écran à partir de sessions enregistrées en vidéo peuvent être envisagées pour des analyses qualitatives à l'échelle réduite de quelques sujets comme dans l'étude de Fernandez *et al.* (2022), des analyses quantitatives à plus large échelle imposent un format textuel de collecte. Pelánek et Effenberger (2022) listent plusieurs possibilités de format textuel pour la collecte des programmes de l'utilisateur, parmi lesquels figure le format xml. Si l'enregistrement des programmes dans un format textuel permet des traitements à large échelle, il implique aussi de se doter d'outils pour régénérer les images afin d'être en mesure

de construire des visualisations facilement interprétables, ce que nous avons fait.

Ainsi, en mettant en place cette méthodologie de collecte et d'analyse de données à trois échelles différentes, nous visons à la fois des analyses très précises et une robustesse statistique en établissant une correspondance entre les analyses à ces trois échelles. Nous répartissons les données collectées de la manière suivante par rapport aux deux axes d'étude de la théorie des champs conceptuels.

— Les données relatives aux puzzles et les données décrivant globalement l'ensemble de la population confrontée à ces puzzles renseignent l'axe des situations.
— Les données locales renseignent l'axe des schèmes, de manière d'autant plus précise que la granularité est fine. L'interprétation de ces données devrait nous permettre d'identifier et de documenter les différentes composantes d'un schème présentes dans la définition analytique qu'en donne Vergnaud (voir la section « Cadre théorique »).

Paliers de difficulté observables à large échelle

À l'échelle large de l'ensemble des participations individuelles au concours Algoréa en langage Scratch, nous disposons du taux de réussite pour chaque puzzle et chaque niveau de classe. C'est à cette échelle que nous menons une analyse épistémologique des situations comme le préconise Vergnaud. Par une analyse *a priori* des situations, nous identifions des variables qui permettent de distinguer des classes de situations. Les taux de réussite nous aident en ce sens dans la mesure où des puzzles proches dont le taux de réussite est significativement différent ne sont très probablement pas appréhendés de la même manière par le sujet, sans que l'on puisse à cette échelle déterminer ce qui se passe du point de vue de l'activité du sujet.

Nous reprenons les puzzles pour lesquels nous avons déjà publié une étude et dont la résolution implique d'utiliser une boucle. Pour ces puzzles, nous avons identifié des classes de situations en lien avec l'identification de motifs, habileté qui relève de la pensée informatique (Csizmadia *et al.*, 2015). Nous avons défini un « motif » comme une entité repérable au sein d'un ensemble car répétée à l'identique ou avec des variations prédictibles. Cette définition nous a amenés à distinguer deux types de motifs dans le contexte de la programmation d'un robot virtuel sur une grille. D'une part, un « motif visuel », constitué de cases adjacentes, est observable sur la grille. D'autre part, un « motif algorithmique », constitué

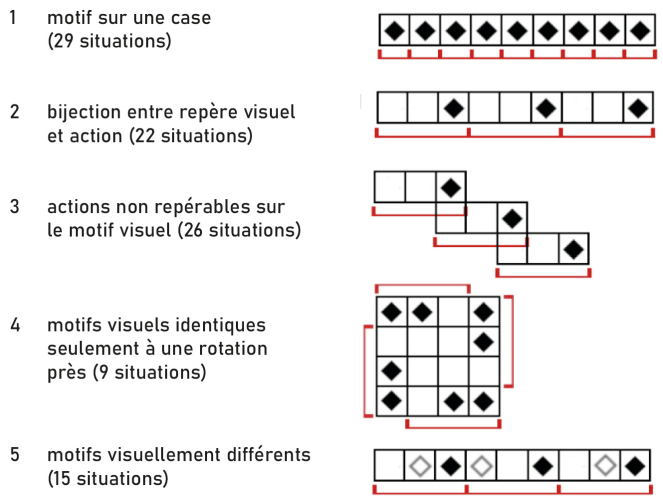

Figure 2 – Classes de situations relatives à l'identification de motifs dans un contexte de programmation d'un robot virtuel sur une grille.

d'une séquence d'actions que le robot répète dans le même ordre chronologique, n'est observable que lors de l'exécution effective de ces actions (Léonard *et al.*, 2022b).

Dans un programme conçu en langage Scratch, le motif algorithmique correspond à la séquence de blocs dans le corps de la boucle. Les classes de situations que nous avons identifiées constituent une gradation dans la difficulté de ces puzzles, qui dépend des caractéristiques du motif visuel et du degré de correspondance entre motif visuel et motif algorithmique. La figure 2 montre cette gradation. Les motifs, constitués de cases adjacentes, y sont délimités par les repères en rouge.

Dans la suite de cette contribution, nous nous concentrons sur la troisième classe de situations : le motif visuel s'étend sur plusieurs cases de la grille, les motifs occupent les deux dimensions de la grille, le système d'orientation du robot est relatif. Dans ces situations, certaines actions du robot (les actions de pivotement) ne sont pas directement repérables sur le motif visuel.

Le puzzle « Distribuer le bois aux castors » en version 1 étoile de la figure 1 est une situation singulière relevant de cette classe de situations. Nous construisons la courbe des taux de réussite relevés à l'échelle nationale pour ce puzzle sur les différents niveaux de classe et le mettons en regard des résultats pour la classe de situations dans laquelle il est catégorisé (figure 3).

Dans cette classe de situation, le puzzle choisi est une des instances les plus faciles. Son taux de réussite s'échelonne entre 70 % en classe de cours moyen 1re année (CM1) et 90 % en classe de 3e. Il est situé en-dehors de la zone interquartile (zone qui comprend la moitié des puzzles appartenant à cette classe de situations). En effet, plusieurs difficultés mises en évidence par ailleurs ne se

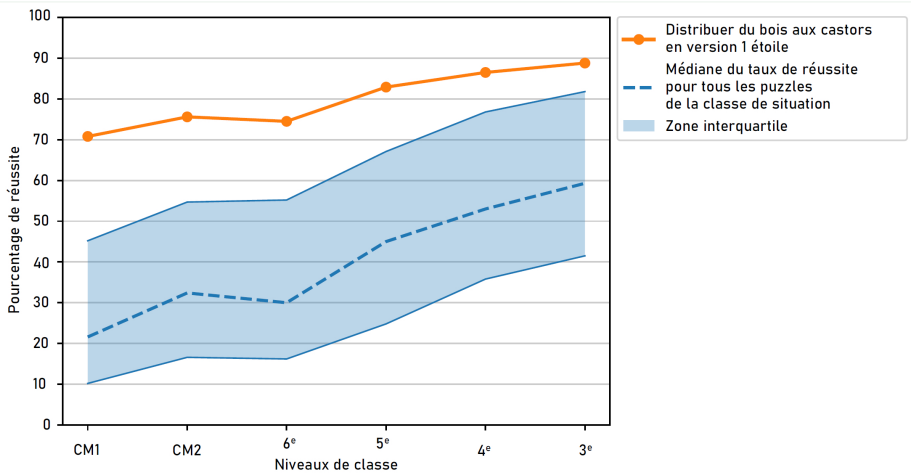

Figure 3 – Taux de réussite pour le puzzle « Distribuer le bois aux castors » version 1 étoile en fonction des niveaux de classe.

retrouvent pas dans cette situation : boucle unique *vs.* séquence de boucles, présence de points de repère concernant le motif (bois, castors, disposition des arbres qui guide le parcours du robot), pas d'instructions avant la boucle, pas de cases adjacentes identiques qui appartiennent au même motif. En conséquence, cette situation devrait faciliter l'identification des difficultés spécifiques à cette classe de situations.

Mobilisation de schèmes révélée par un indicateur de rapidité normalisée

Dans cette section, nous étudions la mobilisation de schèmes de manière quantitative, en analysant les exécutions de programme et leur validité. La difficulté d'un puzzle se traduit dans les traces d'interaction par une augmentation du nombre d'essais et du temps passé sur la tâche. Nous mettons en correspondance la variation de la valeur de ces indicateurs avec deux cas dans lesquels peut se trouver le sujet face à une situation, qui sont décrits dans de la théorie des champs conceptuels : un premier cas où le sujet mobilise un schème opérationnel pour traiter la situation de manière experte en mobilisant un schème déjà prêt et un second cas où le sujet entre dans un processus d'accommodation où il remet en cause le système de schèmes précédemment établi.

Afin d'illustrer le propos, nous proposons de partir de la visualisation de la session d'un sujet singulier sous forme de frise chronologique (figure 4). Ce type de visualisation, qui met en évidence l'aspect temporel de l'activité du sujet, est répertorié parmi les techniques de visualisation (Khan et Khan, 2011). Il a

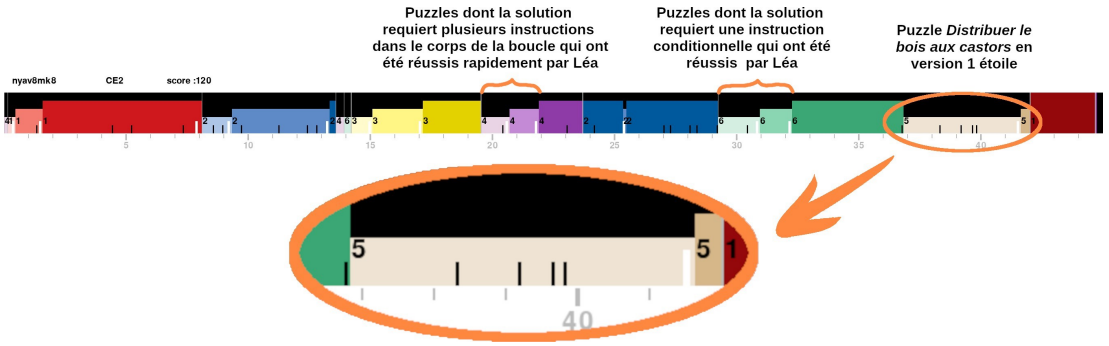

Figure 4 – Visualisation d'une session d'élève sous forme de frise chronologique, avec zoom sur le puzzle « Distribuer le bois aux castors » en version 1 étoile.

déjà été mobilisé dans le contexte de la programmation par blocs (Fernandez *et al.*, 2022).

Les problèmes du parcours sont numérotés de 1 à 6. Chaque problème est représenté par une couleur dédiée. Chaque version est représentée par une hauteur et une nuance, des versions 1 étoile, les plus claires et plus basses, jusqu'aux versions 4 étoiles, les plus foncées et plus hautes. Léa[2] est élève de cours élémentaire 2e année (CE2). Elle fait partie des élèves dont les sessions ont été enregistrées en vidéo. Léa a abordé le puzzle 5 « Distribuer le bois aux castors » en version 1 étoile de la figure 1 en fin de session. Auparavant, elle a validé le problème 4 en versions 1 et 2 étoiles assez facilement, ce qui signifie qu'elle a franchi le palier du passage de une à plusieurs instructions dans le corps de la boucle. Elle possède aussi une première maîtrise de l'instruction conditionnelle (bloc « si/sinon » en Scratch) qui est mise en évidence par la validation des versions 1 et 2 étoiles du problème 6. Pour le puzzle « Distribuer le bois aux castors » en version 1 étoile dont nous détaillons l'analyse, nous observons que plusieurs essais et une durée de près de 5 minutes ont été nécessaires à Léa pour réussir le puzzle. La procédure n'est donc pas experte, Léa est très probablement entrée dans un processus d'accommodation qui a abouti positivement, ce que nous argumenterons dans les sections suivantes.

2. Le prénom a été changé.

Au-delà de l'analyse de frises chronologiques de sujets singuliers, nous cherchons à réaliser une analyse quantitative de l'ensemble de nos données à l'échelle intermédiaire des classes.

À cette fin, nous nous appuyons sur des travaux de recherche relatifs à la construction d'indicateurs à partir de traces d'interaction et à l'établissement de profils en rapport avec la résolution d'un puzzle donné. Pelanek définit quatre profils de performance basés sur une combinaison de deux indicateurs : temps passé sur la tâche et catégorisation des erreurs en « commune » et « non

commune » (Pelánek, 2018). Jiang *et al.* (2022) utilisent une méthode de *clustering* (technique de *machine learning* visant à établir des groupements dans des données) afin de caractériser quatre profils à partir des programmes intermédiaires soumis : ceux qui abandonnent rapidement (*quitters*), ceux qui sont loin d'une solution valide lors des premiers essais et s'en approchent plus ou moins sans aboutir (*approachers*), ceux qui sont loin d'une solution valide lors des premiers essais puis s'en approchent jusqu'à trouver (*solvers*), ceux qui soumettent un programme valide dès les premiers essais (*knowers*).

De ces travaux, nous retenons l'idée d'établir des profils fondés sur une combinaison d'indicateurs construits à partir des traces d'interaction. Dans cette optique, nous construisons un indicateur qui mesure la rapidité avec laquelle un sujet résout un puzzle de programmation, indépendamment de sa dextérité et de la longueur de la solution de référence. Cet indicateur de rapidité est normalisé, c'est-à-dire qu'il est rendu indépendant de la vitesse du sujet pour placer un bloc dans l'éditeur et du nombre de blocs à déplacer pour obtenir la solution de référence. Ainsi, il permet de comparer des résolutions entre elles et de procéder à des traitements statistiques tout en rendant compte, au niveau individuel, de la durée irréductible à l'exécution de la tâche et du temps éventuellement nécessaire à un processus d'accommodation.

Lorsque le sujet a résolu un puzzle, une valeur basse (voire négative) de l'indicateur de rapidité normalisée signifie que cette résolution a été aisée. À l'inverse, une valeur élevée de l'indicateur de rapidité normalisée, souvent concomitante à un nombre d'essais élevé lui aussi, signifie que le sujet a rencontré des difficultés lors de la résolution du puzzle. Pour une explication détaillée de la construction de cet indicateur, nous invitons le lecteur à consulter l'article *Détermination de profils relatifs à la mobilisation de schèmes lors de la résolution de puzzles de programmation* (Léonard *et al.*, 2023). Notez que le nom de l'indicateur a été changé pour plus de clarté, mais la construction reste absolument inchangée.

À partir de la combinaison de cet indicateur de rapidité normalisée et du nombre d'essais, nous établissons des profils qui caractérisent le type de résolution pour un couple sujet/puzzle.

— Expert : la résolution du puzzle est experte, la réussite ne nécessite qu'un seul essai, et l'indicateur de rapidité normalisée est inférieur à 0,5.
— Ajustement : la réussite intervient en deux ou trois essais, avec un indicateur de rapidité normalisée inférieur à 0,5.
— Accommodation : la réussite intervient après une résolution qui est marquée par un indicateur de rapidité normalisée supérieur à 0,5 ou un nombre d'essais strictement supérieur à trois (ou les deux).

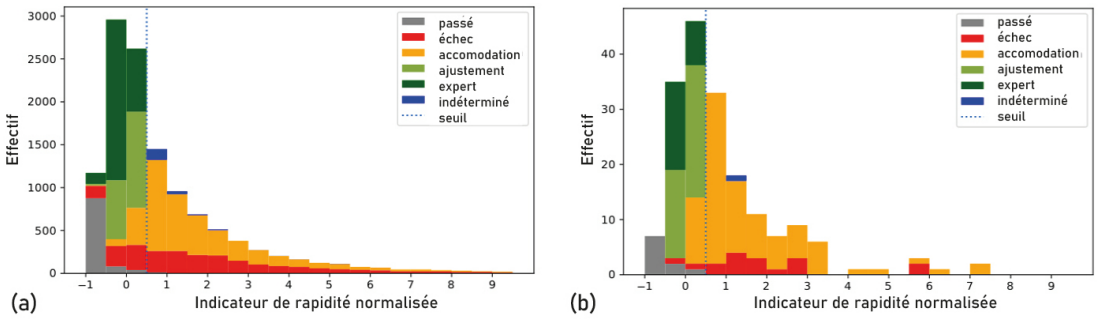

Figure 5 – Histogrammes des profils de résolution. (a) Tous les puzzles des catégories blanche et jaune du concours Algoréa 2022. (b) Puzzle « Distribuer le bois aux castors » en version 1 étoile. Source : Léonard *et al.* (2023).

— Échec : tous les programmes exécutés sont invalides.
— Passé : le puzzle a été ouvert mais aucun programme n'a été exécuté.
— Indéterminé : le type de résolution ne peut pas être déterminé (réussite en un seul essai avec un indicateur de rapidité normalisée supérieur à 0,5).

Nous reprenons ici l'histogramme des profils pour toutes les confrontations sujet/puzzle des élèves des neuf classes de collège de notre étude en 2022 (figure 5a). Nous montrons (figure 5b), la répartition des profils pour les seules confrontations avec le puzzle « Distribuer le bois aux castors » pour ces mêmes élèves et les 20 élèves dont la session a été enregistrée en vidéo.

Nous observons que l'allure générale du diagramme en barres est conservée. Cependant, la répartition des profils est sensiblement différente pour ce puzzle.

Le profil accommodation est très répandu pour le puzzle « Distribuer le bois aux castors » version 1 étoile (49,5 % contre 31,5 % pour l'ensemble des puzzles) alors que les profils expert et échec le sont moins. Nous déduisons de cette répartition particulière que ce puzzle est porteur d'une difficulté qui engendre un processus d'accommodation pour près de la moitié des sujets. Cette difficulté est surmontée pendant la résolution car le taux d'échec est bas (9,3 % pour notre échantillon). Notre objectif est d'identifier la nature de cette difficulté : erreurs commises, conceptualisation erronée.

Si nous revenons à l'exemple de Léa, nous relevons que cette élève, avec cinq essais et un indicateur de rapidité normalisée de 0,82 a un profil accommodation pour ce puzzle. Son indicateur de rapidité normalisée se situe entre 0,5 et 1, la tranche la plus répandue pour le profil accommodation, ce qui nous incite à étudier son activité face à ce puzzle de manière plus approfondie.

Profils	Ensemble des puzzles	Puzzle « Distribuer le bois aux castors » version 1 étoile
Expert	23,4	13,2
Ajustement	15,5	22,0
Accommodation	31,3	49,5
Échec	19,5	9,3
Passé	8,6	5,5
Indéterminé	1,6	0,5

Tableau 1 – Répartition des profils de résolution (en pourcentage).

Zoom sur la procédure experte et les erreurs récurrentes révélées

Nous considérons dans cette section l'ensemble des programmes soumis par les élèves sur le puzzle « Distribuer le bois aux castors » version 1 étoile déjà abordé précédemment.

La figure 6 montre les programmes non valides soumis par les élèves sur ce puzzle. Ces programmes sont triés par ordre décroissant d'occurrences.

Le programme incorrect le plus souvent soumis est un programme dans lequel il manque le bloc « tourner à gauche » en fin de boucle. Plus d'un tiers des élèves (34 %) soumettent ce programme erroné. L'erreur commise est le non replacement du robot relativement au motif visuel qu'il doit parcourir lors de l'itération suivante. En utilisant des expressions régulières, nous cherchons tous les programmes qui comportent une erreur de ce type. Pour 52 % des sujets (89 sur les 172 qui ont soumis au moins un programme pour ce puzzle), cette erreur de non replacement du robot relativement au motif visuel est présente dans au moins l'un des programmes exécutés. Concernant les neuf classes de collège qui ont participé à notre étude, 46 % des élèves de 6^{e}, 45 % des élèves de 5^{e}, 61 % des élèves de 4^{e} et 57 % des élèves de 3^{e}, ont commis cette erreur. Elle est donc très répandue et elle ne régresse pas avec l'âge des élèves au collège.

Nous regardons à présent quel est le profil de résolution des élèves qui ont commis cette erreur. Pour une part significative des élèves, la soumission de ce programme va de pair avec un échec dans la résolution du puzzle, ce qui n'est pas le cas par exemple pour le programme d'id 742 qui est le deuxième plus fréquemment soumis. Dans ce dernier cas, l'erreur concerne le nombre d'itérations à renseigner dans le bloc « répéter ». Ce nombre est laissé à sa valeur par défaut, mais il est ensuite rectifié dans tous les cas. En revanche, l'absence du bloc « tourner à gauche » correspond à une difficulté conceptuelle récurrente, que nous proposons de creuser.

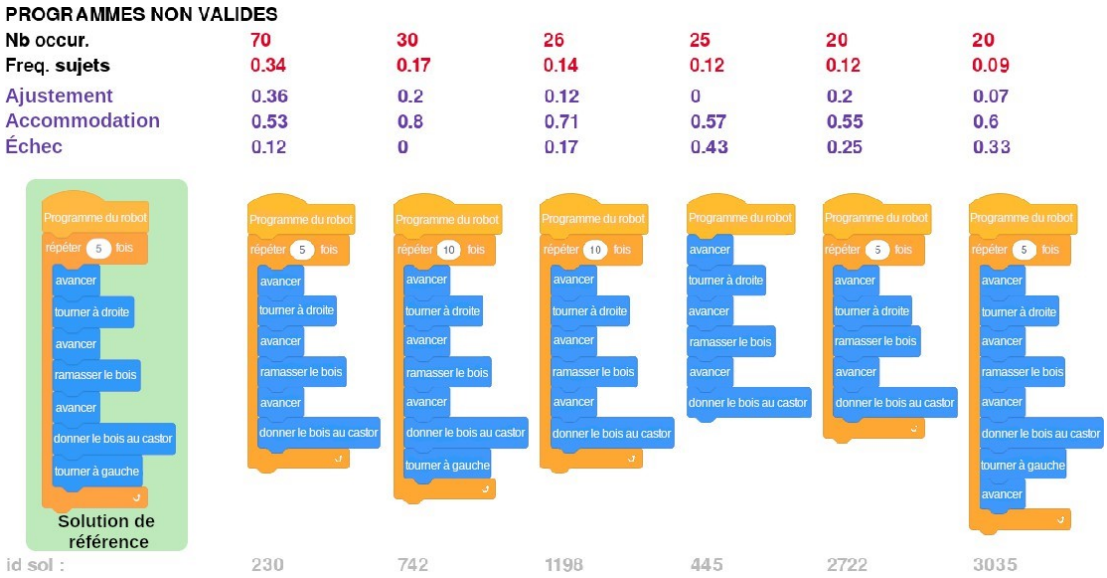

Figure 6 – Programmes non valides soumis par ordre décroissant d'occurrences. Crédit : France-ioi.

Zoom sur la conceptualisation-en-acte du sujet

Dans cette section, nous approfondissons l'étude de l'erreur récurrente mise en évidence dans la section précédente : absence du bloc « tourner à gauche » en dernière position de la séquence dans le corps de la boucle. Nous retrouvons cette erreur dans les traces d'interaction de plusieurs élèves qui ont bénéficié de sessions individuelles, certains pendant deux années. Pour ces élèves, nous disposons à la fois d'une visualisation de la succession des programmes soumis pour chaque puzzle et de l'enregistrement vidéo de la session. La succession des programmes soumis, bien qu'ayant une granularité assez fine, reste une collecte de données discrètes. Avec l'enregistrement vidéo, nous passons à un flux continu d'informations du point de vue de la manipulation de l'interface (mouvements de souris, déplacement et suppression de blocs. . .), qui de plus est accompagné des verbalisations du sujet. Nous sommes ainsi mieux en mesure d'identifier les schèmes mobilisés lors de la résolution du puzzle et donc d'appréhender plus finement la conceptualisation-en-acte du sujet.

Nous reprenons l'exemple de Léa et générons la succession des programmes qu'elle a soumis pour le puzzle « Distribuer le bois aux castors » version 1 étoile (figure 7).

La soumission n° 4 de Léa correspond à l'erreur la plus fréquemment commise sur ce puzzle. Nous mettons en relation ces traces d'activité avec l'extrait vidéo correspondant, ce qui nous permet d'accéder au point de vue subjectif du sujet.

Lors de l'ouverture du puzzle, Léa fait part d'une impression de complexité. Lorsque l'expérimentatrice lui demande ce qui lui donne cette impression, sa réponse concerne le nombre d'éléments sur la grille : « Quand on voit tous les trucs là. . . tous les castors. . . tous les bois... euh ». Cette première impression est cohérente avec le fait que les puzzles dont la grille est épurée des éléments inutiles sont mieux réussis.

Cela dit, Léa identifie rapidement la classe de situation des problèmes itératifs, même si un manque de dextérité dans le maniement de la souris provoque l'interversion de blocs dans la première soumission. Léa mobilise des schèmes déjà construits pour cette classe de situation, ce que révèlent ses verbalisations.

— Schème d'identification du motif qu'elle se représente comme une marche : « il suffit de répéter plusieurs marches ».
— Schème de dénombrement des motifs : « 1, 2, 3, 4, 5 répéter ».

En revanche, Léa ne voit pas d'emblée l'erreur de non replacement du robot par rapport au motif visuel. Une fois que le robot a distribué le bois, il a effectué l'ensemble des actions attachées au motif visuel. L'invariance de la position du robot par rapport au motif visuel à parcourir lors de l'itération à venir n'est pas considérée.

Léa a besoin d'une incitation à utiliser le mode pas à pas. Cette exécution des instructions une par une l'amène à prendre conscience de l'erreur après l'exécution de la première itération : « Ah ça y est,

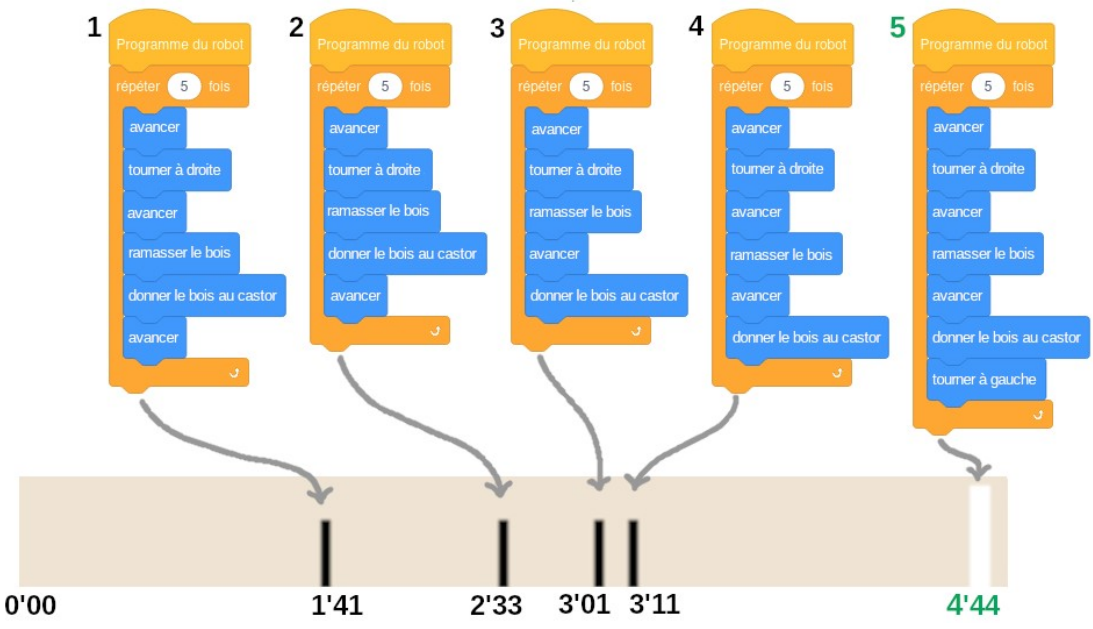

Figure 7 – Succession des programmes soumis par Léa sur le problème « Distribuer le bois aux castors » en version 1 étoile. Crédit : France-ioi.

je viens de comprendre ». Sans hésitation, elle place alors le bloc de rotation manquant en dernière position dans le bloc répéter. En même temps, elle donne une explication à son niveau de la cause de l'erreur : « tourner à gauche sinon il va foncer », ce qui prouve qu'une analyse de l'exécution a eu lieu, que la correction n'est pas de l'ordre de la stratégie par essai-erreur, stratégie répandue qui consiste à effectuer des modifications du programme au hasard puis à lancer systématiquement une exécution après chaque modification sans prendre le temps de l'analyse. Par sa remarque, Léa nous donne aussi une indication sur l'élément de l'interface qui l'a aidée à prendre conscience de son erreur : l'arbre dans lequel le robot ne doit pas foncer. L'indication de Léa est cohérente avec des résultats que nous ne détaillons pas dans cette contribution, qui concernent la présence d'éléments sur la grille : ces éléments sont aidants s'ils contraignent le parcours du robot.

Dans le système d'orientation relative, la position du robot comprend la case de la grille sur laquelle il se trouve (ses coordonnées en arrière-plan) et son orientation. Lorsqu'il se représente l'exécution, le sujet doit donc maintenir ces deux informations en mémoire. C'est la gestion de l'orientation du robot qui s'avère être la plus difficile. En effet, à l'inverse de la succession de cases occupées par le robot que le sujet se représente souvent en pointant les cases de la grille avec sa souris, celle-ci n'est pas visualisable avant l'exécution du programme.

Si nous interprétons cette erreur de non replacement du robot en position adéquate pour aborder le motif de l'itération suivante par rapport à la théorie des champs conceptuels, elle correspond à un théorème-en-acte qui est manquant : « Au début de chaque itération, le robot est toujours positionné de la même manière par rapport au motif qu'il va parcourir ». Ce théorème peut être mis en lien avec le concept plus avancé d'invariant de boucle, qui peut être défini simplement comme une propriété qui est vraie avant et après chaque itération.

Nous mettons en relation ce défaut de conceptualisation avec le fait que le programme est perçu comme commandant une suite d'actions du robot virtuel sur la grille, et non comme une succession d'états du système, états liés à l'évolution de variables au cours de l'exécution du programme. La dialectique entre exécution séquentielle d'actions et succession de situations qui présentent des propriétés statiques a été identifiée par Rogalski dans le contexte de l'initiation à la programmation textuelle au lycée (Rogalski, 1987). Dans le contexte de la programmation en langage Scratch d'un robot virtuel sur une grille, la succession d'états est très peu visible pour l'utilisateur. Lorsqu'il lance l'exécution de son programme, celui-ci perçoit une succession de déplacements et d'actions du robot virtuel plutôt qu'une succession d'états de la grille dont le robot, avec sa position et son orientation, fait partie.

Le replacement du robot par rapport au prochain motif avant chaque itération n'est pas perçu comme une action. Il fait partie du motif algorithmique mais pas du motif visuel. Ce replacement du robot est donc souvent ignoré, au moins en première intention.

Nous concluons en renseignant les composantes du schème identifié.

Éléments de définition analytique du schème de replacement du robot relativement au motif

- But : replacer le robot en position adéquate pour aborder l'itération suivante.
- Règles de conduite de l'action :
  - prise d'information : repérer la position du robot une fois qu'il a parcouru le motif visuel ; puis repérer la position que doit avoir le robot pour aborder le motif suivant ;
  - action proprement dite : à la séquence de blocs déjà présents dans le bloc répéter, ajouter les blocs qui permettent d'amener le robot dans la position voulue ;
  - contrôle : lancer une exécution du début du programme (en mode pas à pas) ou se représenter mentalement cette exécution, afin de vérifier la bonne jonction entre le traitement des deux premiers motifs.
- Invariants opératoires :
  - concept-en-acte : invariant de boucle ;
  - théorèmes-en-acte : « Au début de chaque itération, le robot est toujours positionné de la même manière par rapport au motif qu'il va parcourir ».

Conclusion

Dans cette contribution, nous avons montré comment nous combinons les apports respectifs de la théorie des champs conceptuels et de l'analyse de traces afin d'étudier la résolution de puzzles de programmation avec un langage par blocs. D'un côté, la théorie des champs conceptuels nous donne des clés pour structurer l'analyse de nos données et les interpréter. Réciproquement, la possibilité d'articuler des traitements statistiques sur de larges échantillons avec la précision d'observation d'une approche qualitative est de nature à apporter une robustesse aux résultats établis en se plaçant dans ce cadre d'analyse.

Nous avons mis en œuvre cette méthode sur un exemple de puzzle qui requiert la mobilisation de la notion de boucle. Elle nous a permis d'identifier des erreurs fréquentes d'élèves d'école élémentaire et de collège (7 à 15 ans) lors de la confrontation à cette notion de base de l'algorithmique. À titre d'illustration, nous avons interprété l'une de ces erreurs en termes de schème inopérant et appréhendé la conceptualisation-en-acte sous-jacente à cette difficulté. Une de nos perspectives de recherche est de reproduire cette démarche pour mener des investigations sur l'ensemble des concepts abordés lors de l'initiation à la programmation à l'école obligatoire. Celles-ci nous amènerait à identifier et à documenter un ensemble d'erreurs et de stratégies fréquentes lors de la résolution de puzzles de programmation avec un langage par blocs.

Nous pensons que cette compréhension des erreurs et de la conceptualisation-en-acte des élèves est de nature à outiller les enseignants qui initient leurs élèves à l'informatique. Les résultats obtenus à l'issue de nos recherches pourraient conduire à formuler des recommandations aux enseignants afin de leur permettre de mieux appréhender à la fois les enjeux de savoirs et les difficultés fréquentes pour lesquelles une vigilance particulière s'impose. Par exemple, pour le cas d'étude présenté dans cette contribution, l'erreur est persistante tout au long des années de collège et une intervention pédagogique ciblée serait nécessaire. Un tel document pourrait conseiller aux enseignants de faire remarquer la position initiale du robot, puis de faire exécuter une première fois les instructions présentes dans la boucle avec le mode pas à pas, pour amener à discuter de la position du robot à l'issue de cette exécution par rapport à sa position initiale. La préconisation serait d'en tirer les conséquences en termes de programmation d'une boucle, en attirant particulièrement l'attention sur la jonction entre les deux premières itérations.

Notre travail de recherche vise la compréhension des difficultés inhérentes à l'apprentissage des notions de base en algorithmique dans un environnement de programmation par blocs, et contribue ainsi à étayer les enseignants dans leur compétence à ajuster leurs scénarios pédagogiques et à accompagner leurs élèves pour surmonter ces difficultés.

Références

Csizmadia, A., Curzon, P., Dorling, M., Humphreys, S., Ng, T., Selby, C., et Woollard, J. (2015). *Computational thinking. A guide for teachers*, Computing at school. https://www.researchgate.net/publication/327302966_Computational_thinking_-_a_guide_for_teachers

Effenberger, T., et Pelánek, R. (2022). Design and analysis of microworlds and puzzles for block-based programming, *Computer Science Education*, vol. 32, n° 1, p. 66-104. https://doi.org/10.1080/08993408.2020.1832813

Fernandez, C., Freitas, J. A., Lopes, R. d. D., et Blikstein, P. (2022). *Using video analysis and learning analytics to understand programming trajectories in data science activities with Scratch*, IDC '22 : Proceedings of the 21st Annual ACM Interaction Design and Children, Braga, Portugal, p. 253-260. https://doi.org/10.1145/3501712.3529742

Gelman, R., et Gallistel, C. R. (1978). *The child's understanding of number*, Harvard University Press.

Ihantola, P., Vihavainen, A., Ahadi, A., Butler, M., Börstler, J., Edwards, S. H., Isohanni, E., Korhonen, A., Petersen, A., et Rivers, K. (2015). Educational data mining and learning analytics in programming : Literature review and case studies, *Proceedings of the 2015 Innovation and Technology in Computer Science Education (ITiCSE) on Working Group Reports*, p. 41-63. https://doi.org/10.1145/2858796.2858798

Jiang, B., Zhao, W., Zhang, N., et Qiu, F. (2022). Programming trajectories analytics in block-based programming language learning, *Interactive Learning Environments*, vol. 30, n° 1, p. 113-126. https://doi.org/10.1080/10494820.2019.1643741

Khan, M., et Khan, S. S. (2011). Data and information visualization methods, and interactive mechanisms : A survey, *International Journal of Computer Applications*, vol. 34, n° 1, p. 1-14.

Léonard, M., Peter, Y., Secq, Y., et Fluckiger, C. (2022a). Computational Thinking : Focus on Pattern Identification, *Lecture Notes in Computer Science*, vol. 13 450, p. 187-200. https://doi.org/10.1007/978-3-031-16290-9_14

Léonard, M., Secq, Y., Peter, Y., et Fluckiger, C. (2022b). « Pensée informatique : approche didactique de l'identification de motifs », dans *L'informatique, objets d'enseignement et d'apprentissage. Quelles nouvelles perspectives pour la recherche ?*, p. 113-125. https://hal.science/hal-03697888/

Léonard, M., Bouton, M., et Peter, Y. (2023). Détermination de profils relatifs à la mobilisation de schème lors de la résolution de puzzles de programmation, *Actes de la onzième conférence sur*

les environnements informatiques pour l'apprentissage humain, Brest, France.

Pelánek, R. (2017). Bayesian knowledge tracing, logistic models, and beyond : an overview of learner modeling techniques, *User Modeling and User-Adapted Interaction*, vol. 27, n° 3, p. 313-350. https://doi.org/10.1007/s11257-017-9193-2

Pelánek, R. (2018). Exploring the utility of response times and wrong answers for adaptive learning, *Proceedings of the fifth annual ACM conference on learning at scale*, p. 1-4. https://doi.org/10.1145/3231644.3231675

Piaget, J. (1935). *La naissance de l'intelligence chez l'enfant*, Delachaux et Niestlé Neuchatel.

Rogalski, J. (1987). Acquisition de savoirs et de savoir-faire en informatique, *Cahiers de Didactique des Mathématiques*, vol. 43.

Vergnaud, G. (1991). « La théorie des champs conceptuels », dans *Recherches en didactique des mathématiques*, La Pensée Sauvage.

Vergnaud, G. (2001). Forme opératoire et forme prédicative de la connaissance, *Investigações em Ensino de Ciências*, vol. 17, n° 2, p. 287-304.

Vergnaud, G. (2007). Représentation et activité : deux concepts étroitement associés, *Recherches en éducation*, vol. 4, p. 9-22. https://doi.org/10.4000/ree.3889

Vergnaud, G. (2013). Qu'est-ce que la pensée ?, *La nouvelle revue de l'adaptation et de la scolarisation*, vol. 63, n° 3, p. 277-299. https://doi.org/10.3917/nras.063.0277

Vergnaud, G., et Durand, C. (1976). Structures additives et complexité psychogénétique, *Revue française de pédagogie*, p. 28-43. https://doi.org/10.3406/rfp.1976.1622

Pour citer ce chapitre :

Léonard, Marielle (2024). « Zoom sur quelques erreurs récurrentes lors des premiers apprentissages en algorithmique », dans Cédric Fluckiger, Laetitia Boulc'h, Sandra Nogry et Christophe Reffay (dir.), *Enseigner, apprendre, former à l'informatique à l'école : regards croisés*, Université Paris Cité, p. 135-154. https://doi.org/10.53480/2024iecare07u/