Construction et exploitation d'un référentiel de types de tâches d'apprentissage de la programmation

Sébastien Jolivet[1][0000−0003−3915−8465], Eva Dechaux[2][0000−0001−8579−1398], Anne-Claire Gobard[3][0000−0002−2866−084X] et Patrick Wang[4][0000−0003−3117−8189]

[1] IUFE & TECFA, Université de Genève, Suisse
sebastien.jolivet@unige.ch
[2] IUFE, Université de Genève, Suisse
eva.dechaux@unige.ch
[3] Lycée Kastler, Académie de Versailles, France
anne-clair.gobard@ac-versailles.fr
[4] Haute école pédagogique du canton de Vaud, Lausanne, Suisse
patrick.wang@hepl.ch

Résumé. Dans cette contribution, nous décrivons la construction d'un référentiel de types de tâches pour l'apprentissage de concepts fondamentaux de la programmation à l'aide d'une analyse systématique de moyens d'enseignement français et suisses. Cette analyse a permis d'identifier et structurer une centaine de types de tâches portant sur l'acte de programmer ainsi que sur les concepts de variables, instructions conditionnelles et boucles. Nous avons ensuite exploité ce référentiel pour analyser trois EIAH pour l'apprentissage de la programmation avec Python.

Mots-clés : Apprentissage programmation , modélisation du savoir , analyse d'EIAH

Abstract. This paper presents the construction of a repository of types of tasks that students might have to perform while learning the fundamentals of programming. To do so, we analyzed French and Swiss teaching and learning resources and identified around 100 types of tasks relating to the act of programming, the concepts of variables, conditional statements, and loops. We then used this repository to analyze three web-based tools designed to learn Python programming.

Keywords: Programming , Knowledge modelling , Tool Analysis

1 Introduction

L'apprentissage de la programmation est apparu dans les programmes scolaires du secondaire depuis maintenant quelques années aussi bien en France (SNT puis

NSI) qu'en Suisse romande (informatique en tant que discipline obligatoire). Les curriculums de ces disciplines mentionnent l'apprentissage de concepts fondamentaux de la programmation impérative. Si les grandes lignes sont définies en termes de notions à rencontrer, il n'y a pas ou peu d'éléments disponibles sur ce que signifie vraiment "apprendre à programmer" : quelles tâches doivent être rencontrées ? quelles procédures permettant de les réaliser doivent être enseignées ? basées sur quels savoirs ? Poser ces questions revient à interroger la transposition didactique externe [7] des savoirs de référence en informatique. Un des moyens d'étudier l'état de la transposition didactique est d'analyser des moyens d'enseignement de manière à identifier tâches, procédures et savoirs présents dans les premiers apprentissages de la programmation. Cet existant étudié, nous l'avons structuré (Sect. 3), puis nous avons questionné sa prise en compte dans trois EIAH d'apprentissage de la programmation avec le langage Python (Sect. 4).

2 Problématique et méthode

Décrire les savoirs qui doivent être travaillés pour l'apprentissage de la programmation n'est pas un problème nouveau. Par exemple, deux référentiels ont été produits, autour de la programmation récursive et de la programmation fonctionnelle, en lien avec le projet Comper [14]. Couderette [9] a, elle, étudié les effets de l'introduction de l'algorithmique dans le programme de mathématiques de seconde et a identifié des praxéologies algorithmiques ou informatiques pour décrire les tâches données aux élèves par des enseignants. Strock et Artaud [15] se sont aussi intéressé à l'apprentissage de l'algorithmique mais en étudiant un curriculum. Enfin, Chaachoua et al. [6] ont examiné l'articulation entre des tâches de programmation et des tâches d'apprentissage de la division euclidienne.

À notre connaissance, l'analyse de la transposition didactique liée à l'introduction de l'informatique comme discipline scolaire, réalisée en étudiant divers moyens d'enseignement n'a pas été faite. Ce constat, couplé au développement d'EIAH d'apprentissage de la programmation, notamment de Python, nous a amenés à formuler les deux questions de recherche suivantes :

QR1 Quelles tâches sont proposées dans les ressources d'enseignement, dans le cadre des premiers apprentissages d'un langage impératif ?

QR2 Quelles tâches sont présentes dans les EIAH d'apprentissage de Python ?

Pour décrire et structurer les tâches des moyens d'enseignement nous utilisons le modèle praxéologique issu de la théorie anthropologique du didactique [8]. Nous présentons maintenant ce modèle et notre méthode pour répondre à (QR1).

2.1 Modèle praxéologique

Le modèle praxéologique [8] postule que toute activité humaine peut être représentée à l'aide d'une praxéologie qui est définie par un quadruplet $[T, \tau, \theta, \Theta]$. Dans ce quadruplet, T est un *type de tâches*, les tâches le constituant pouvant être réalisées à l'aide de la *technique* τ. Le bloc $[\theta, \Theta]$ constitue le bloc du *logos*

et contient les éléments (définitions, propriétés, théorèmes du domaine) qui permettent de valider, justifier, expliquer le fonctionnement de la technique τ. Dans l'extension du modèle praxéologique T4-TEL [5], les techniques sont décrites à l'aide de types de tâches : nous parlons des *ingrédients de la technique*.

Par exemple si l'on considère le type de tâches *Déterminer le nombre d'itérations d'une boucle bornée donnée* (T_1), alors une technique τ_1 permettant de réaliser T_1 peut être décrite (partiellement) à l'aide des types de tâches *Identifier la structure de données parcourue par l'itérateur* et *Déterminer les valeurs prises par l'itérateur*. Une autre technique peut être décrite à l'aide du type de tâches *Exécuter pas à pas le programme et compter le nombre d'itérations*. Chacun de ces types de tâches admet sa propre technique, et le bloc du *logos* comprend les définitions de boucle bornée et d'itération, les propriétés de l'itérateur, etc.

2.2 Méthode d'analyse des moyens d'enseignement

Par programmation nous entendons, comme proposé par exemple dans [15], le travail qui consiste à produire un programme informatique permettant de réaliser une tâche t (trier une liste, déplacer un personnage, construire une figure avec une tortue, etc). Ce travail comprend le travail algorithmique et le travail d'implémentation. Nous limitons notre analyse de la manière suivante :

- Nous nous centrons sur les premiers apprentissages et plus précisément les types de tâches élémentaires de l'activité de programmation et ceux qui concernent les variables, les boucles et les instructions conditionnelles.
- Nous ne nous intéressons pas aux tâches liées à l'utilisation d'un langage spécifique (par exemple l'indentation dans le langage Python).

Ces limites fixées, nous avons procédé à l'analyse systématique, outre les curriculums, d'un corpus de ressources d'enseignement composé de ressources de natures différentes (notamment, manuels scolaires [2,4,11], cahier d'exercices [10], site Web contenant des moyens d'enseignement de l'informatique [1]) provenant d'institutions différentes (françaises et suisses et de niveaux scolaires différents). Nous avons listé l'ensemble des éléments de praxéologies identifiés dans ce corpus, il s'agit essentiellement des types de tâches et de quelques éléments relatifs aux techniques ou au bloc du *logos*.

Il est notable, dans le corpus étudié, que les techniques ne sont pas présentées de manière explicite. Par exemple on ne trouve pas la réponse à une question du type "comment fait-on pour concevoir une boucle bornée ?", celle-ci étant quasi exclusivement traitée au moyen d'exercices. Certains ingrédients de techniques font l'objet d'exercices spécifiques, mais le travail permettant à l'élève de "recoller les morceaux" n'est pas pris en charge dans les documents étudiés.

Les éléments du bloc du *logos* sont largement absents des ressources du corpus. Et, même si l'on rencontre parfois une définition, l'articulation entre les éléments du bloc du *logos* et les techniques est totalement absente. On ne trouve par exemple pas de justification du type "je peux utiliser telle ou telle structure comme itérateur dans une boucle bornée car. . .".

Dans la suite de cet article, nous nous concentrons sur le bloc praxique $[T, \tau]$. Le travail d'analyse systématique réalisé nous a permis d'obtenir une liste non or-

ganisée d'environ deux cents types de tâches. Un premier travail d'identification des types de tâches identiques, modulo des différences de vocabulaire, a produit une liste réduite d'une centaine de types de tâches. Le travail de structuration de ces types de tâches est l'objet de la section suivante.

3 Types de tâches d'apprentissage de la programmation

La structuration globale, que nous motivons dans les lignes suivantes, est présentée en Fig. 1. Tout d'abord nous considérons, comme évoqué précédemment, que programmer c'est produire un programme P_t qui réalise une tâche t. Nous identifions donc une première catégorie de types de tâches que nous nommons *Produire*. Or, pour produire un programme, en adoptant une posture anthropologique, nous identifions deux situations : 1) nous disposons d'un programme $P_{t'}$ qui réalise déjà une tâche t' proche de t ; 2) nous ne disposons pas d'un tel programme. Ceci nous amène donc à distinguer deux situations que nous avons nommées *Code existant* et *Feuille blanche*. Que l'on soit dans le cas initial de la

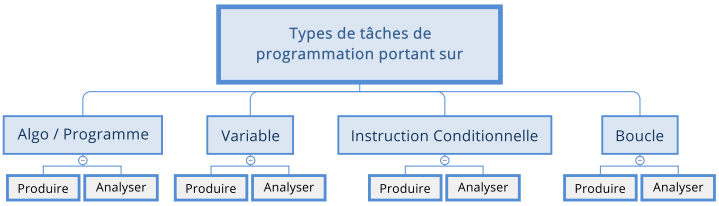

Fig. 1. Structuration globale des types de tâches.

feuille blanche, avec un premier code produit, ou dans le cas d'un code existant, nous avons, à un moment, un programme $P_{t'}$ qui ne réalise pas t. Il faut donc analyser $P_{t'}$ pour le modifier jusqu'à obtenir P_t. Ceci nous amène à définir une seconde catégorie de types de tâches que nous nommons *Analyser*. D'une manière générale, ces types de tâches sont des ingrédients de techniques des types de tâches de la catégorie *Produire*. Ils peuvent cependant avoir une existence autonome dans des exercices présents dans des ressources d'apprentissage. Ces types de tâches relatifs à l'objet P_t constituent la catégorie *Programme*.

Dans un deuxième temps, pour pouvoir produire P_t, ou travailler dessus, il faut mobiliser des objets et structures élémentaires. Nous en considérons trois : la première, *Variable*, concerne l'objet fondamental que sont les variables ; les deux autres, *Instruction conditionnelle* et *Boucle*, concernent les deux structures de contrôles incontournables. Au sein de chacun de ces ensembles, les types de tâches sont organisés selon les deux activités *Produire* et *Analyser*.

Une fois cette structuration définie, nous avons organisé les différents types de tâches identifiés au sein de chaque catégorie. Les différents types de tâches de la catégorie *Programme* sont présentés en Fig. 2.

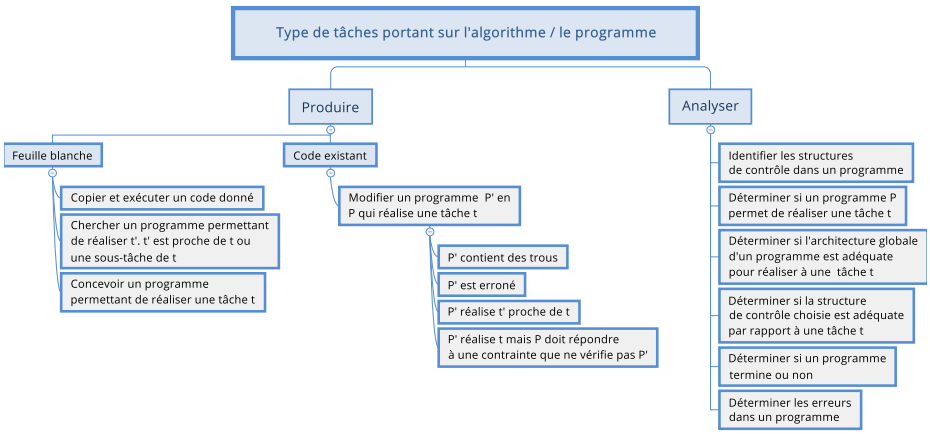

Fig. 2. Les types de tâches de la catégorie *Programme*.

Deux éléments sont importants à noter. Premièrement, il ne s'agit pas d'une liste exhaustive mais de types de tâches existants dans le corpus étudié. Deuxièmement, les types de tâches proposés sont de niveaux de granularité différents. Ainsi, le type de tâches *Concevoir un programme permettant de réaliser une tâche t* (noté T_P) est le plus global, tandis que le type de tâches *Déterminer une structure de contrôle pertinente par rapport au problème* est un ingrédient d'une technique permettant de réaliser T_P. Pour des raisons de place et de lisibilité nous ne présentons pas dans les arbres l'intégralité des types de tâches identifiés. Nous avons conservé tous ceux qui permettent de mener à bien l'analyse présentée dans la Sect. 4, mais avons supprimé des types de tâches ingrédients de technique comme par exemple *Identifier les variables nécessaires pour concevoir un programme* qui fait l'objet d'exercices spécifiques dans le corpus mais est en fait un ingrédient d'une technique de T_P.

Ces deux éléments sont aussi valides pour les Fig. 3, 4 et 5 qui présentent les différents types de tâches identifiés dans le corpus et relatifs aux variables, aux boucles et aux instructions conditionnelles.

Pour continuer la structuration, dans la catégorie *Analyser* nous avons identifié deux genres de tâches. Ils précisent l'activité à mettre en œuvre pour mener l'analyse, et donc les techniques et les éléments du *logos* permettant de réaliser les tâches de ces catégories :

— *Identifier* : pour cette catégorie, en s'appuyant sur les éléments du logos tels que les définitions des structures ou des règles syntaxiques (délimitation des instructions exécutées dans une structure de contrôle par exemple), l'apprenant doit lire dans un code existant, l'information visée.
— *Déterminer* : pour ces types de tâches, l'apprenant va devoir agir sur le code existant (en l'exécutant ou en le simulant) pour réaliser la tâche.

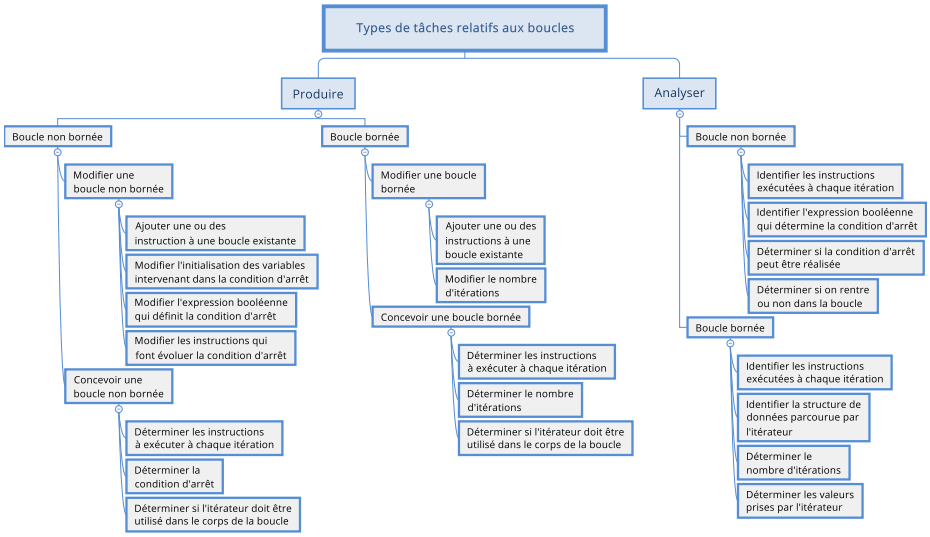

Fig. 3. Les types de tâches de la catégorie *Boucle*

Les éléments du logos mobilisés concernent d'une part l'exécution ou à la simulation d'un code et d'autre part les objets ou structures rencontrés.

Dans la catégorie *Produire* nous avons défini deux genres de tâches :

— *Modifier* : dans cette catégorie il y a déjà un code existant qu'il faut faire évoluer. Nous avons des types de tâches de ce genre dans la famille de types de tâches *Programme* et pour les deux structures de contrôle *Boucle* et *Instruction conditionnelle*. D'une manière générale la modification est guidée par l'objectif de corriger ou adapter un programme. Dans des ressources d'apprentissage elle peut être contrainte par une *limitation du nombre de lignes* ou *l'obligation ou interdiction d'utiliser une structure*.

— *Concevoir* : les types de tâches de ce genre de tâches correspondent à la situation où il n'y a encore aucun élément disponible du programme P qui doit réaliser la tâche t.

Pour le genre de tâche *Concevoir*, le type de tâches le plus générique est celui de la catégorie *Programme* qui est défini par *Concevoir un programme permettant de réaliser une tâche t*. Sa technique mobilise de nombreux autres types de tâches présents dans les arbres. Pour les structures de contrôle nous avons proposé, en dessous de chaque type de tâches global (i.e., *Concevoir une boucle bornée*), les types de tâches ingrédients des techniques qui font l'objet d'exercices spécifiques dans le corpus analysé.

Les différents types de tâches identifiés dans le corpus de ressources permettent de rendre compte de la transposition didactique réalisée dans différents moyens d'apprentissage. Même si l'étude d'autres moyens d'apprentissage permettait de mettre en évidence d'autres types de tâches, la structure globale ne

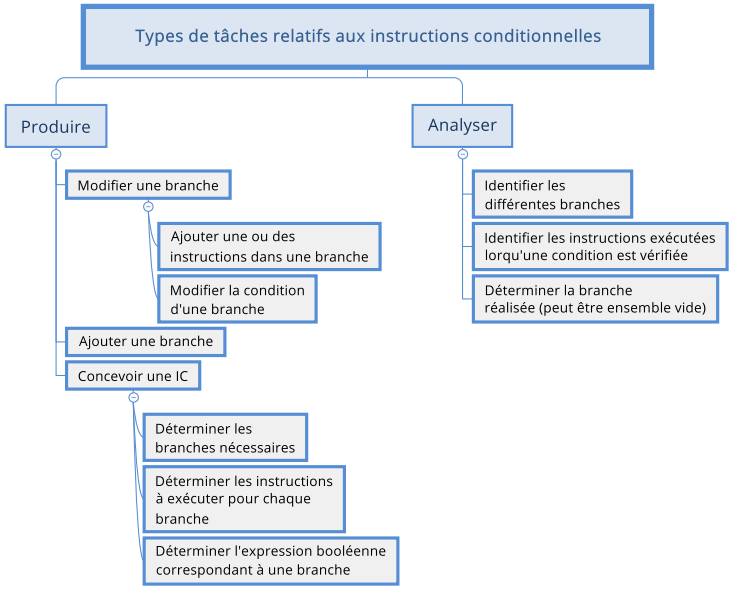

Fig. 4. Les types de tâches de la catégorie *Instruction conditionnelle*.

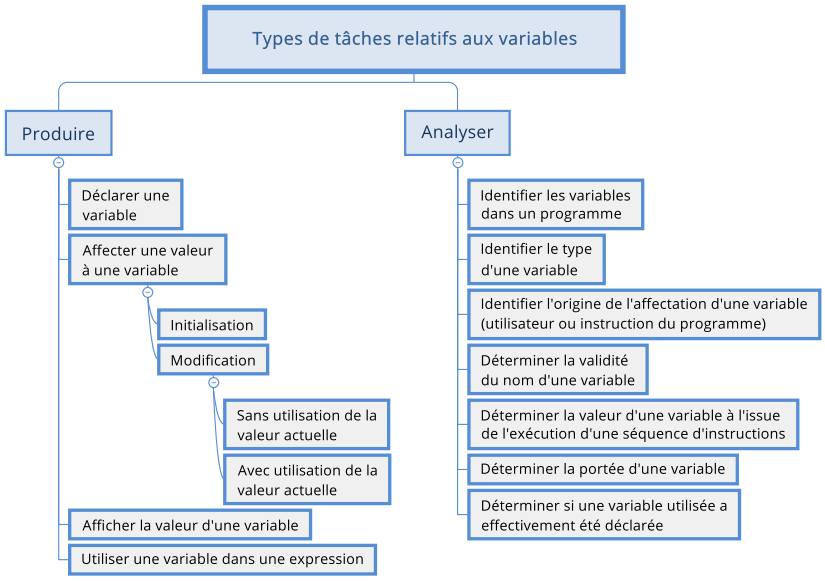

Fig. 5. Les types de tâches de la catégorie *Variable*

serait pas remise en cause et le travail présenté peut être complété. Nous présentons maintenant l'exploitation de ce travail pour analyser trois EIAH.

4 Analyse de trois EIAH

Pour répondre à (QR2), nous étudions trois EIAH qui visent l'apprentissage de Python par des débutants : AlgoPython [5], Citizen Code Python [6], et Pyrates [7] (notés AP, CCP et Py dans la suite de ce texte). Ces trois EIAH sont tous des applications Web récentes (moins de trois ans) ; les trois admettent au moins une version gratuite. AP est proposé par la société Génération 5. Il est présenté comme couvrant tous les aspects de l'algorithmique et de la programmation en Python du niveau lycée et permettant un apprentissage progressif. Il est composé de huit séries d'exercices qui traitent les notions qui nous intéressent (plus les fonctions et les listes), chaque série étant composée de 10 à 24 exercices. CCP a été développé par la société Tralalère et IOI France avec le soutien d'Amazon Future Engineer. Il propose une "initiation à la programmation pour tous" et présente la spécificité de pouvoir réaliser les tâches en Python ou en Blockly. Il est organisé en 11 saisons, chaque saison étant constituée d'un ensemble de 3 à 10 exercices. Py est un environnement développé dans le cadre d'une thèse [3] et propose huit niveaux, chacun étant une tâche de déplacement à réaliser à l'aide d'un programme en Python. Le mémo proposé permet de constater que les notions de variable, boucle et instruction conditionnelle sont abordées.

4.1 Méthode d'analyse des EIAH

Dans chaque EIAH nous n'analysons que les exercices identifiés comme portant sur les variables, les boucles ou les IC. Nous commençons par décrire les EIAH à l'aide d'une grille basée sur les types de tâches présentés dans la section précédente. Ces EIAH exploitent en effet les appels de fonctions comme outils dans les programmes à produire même si elles n'ont pas nécessairement été étudiées.

D'autre part, il arrive fréquemment qu'il existe plusieurs programmes P permettant de réaliser une tâche donnée t. Or, dans ces EIAH, il y a l'intention de faire produire un programme spécifique (par exemple qui utilise une structure de contrôle particulière). Pour tenir compte de ces intentions, nous identifions donc les exercices pour lesquels il existe plusieurs solutions (en restant dans le champ de solutions signifiantes), et laissons de côté celles bloquées par l'EIAH.

Présentation de la grille de description. Pour décrire les EIAH, nous avons construit une grille utilisée pour chaque exercice. Nous commençons par déterminer le ou les types de tâches de la catégorie *Programme* mobilisé(s). Ensuite, nous déterminons s'il existe au moins une résolution (qui a un sens) mobilisant :

5. `https://www.algopython.fr`
6. `https://www.futureengineer.fr/`
7. `https://py-rates.fr`

— uniquement des appels de fonctions prédéfinies et/ou des affectations ;
— uniquement une ou des IC (+ fonctions prédéfinies et/ou affectations) ;
— uniquement une ou des boucles bornées (+ fonctions prédéfinies et/ou affectations) ;
— uniquement une ou des boucles non bornées (+ fonctions prédéfinies et/ou affectations) ;
— une combinaison de boucles et/ou d'IC (+ fonctions prédéfinies et/ou affectations).

Pour chacune des solutions existantes, nous vérifions si cette solution est valide dans l'environnement.

Pour chaque solution valide, nous avons analysé les types de tâches mobilisés dans chacune des catégories (variable, boucles, IC). Enfin, si la solution demande l'utilisation de plusieurs structures de contrôle, nous regardons les relations existantes entre ces structures (succession ou imbrication) et combien d'instructions contiennent ces dernières. Ces éléments peuvent être considérés comme des marqueurs d'une certaine complexité des exercices.

Remplissage de la grille de description. Afin de remplir la grille nous résolvons chaque exercice. Sur la base de ces résolutions, nous identifions les différents types de tâches présents dans chaque exercice. Afin de nous assurer de la validité de notre grille nous avons procédé à un double codage à l'aveugle pour l'ensemble des exercices d'AP. Celui-ci nous a permis de constater une concordance des descriptions obtenues pour 42 exercices sur 45. Les écarts ont été réglés à l'aide du choix suivant : pour produire la description, nous décrivons uniquement les types de tâches prescrits par l'exercice. Ainsi, dans la description, nous n'associons pas à un type de tâches des ingrédients d'une de ses techniques. Par exemple si l'exercice mobilise le type de tâches *Concevoir une boucle bornée* nous ne lui associons pas le type de tâches *Déterminer les instructions à exécuter à chaque itération*, sauf dans le cas où celui-ci fait l'objet d'une question spécifique.

4.2 Description de trois EIAH

Dans cette section nous présentons quelques éléments significatifs obtenus à partir de l'analyse des grilles de description. Tout d'abord dans la Table 1 nous présentons les différents types de tâches de la catégorie *Programme* identifiés dans chaque EIAH. Les exercices présents dans les EIAH sont constitués d'une unique question, ce qui entraîne qu'un seul type de tâches de cette catégorie est présent par exercice.

On peut noter qu'aucun type de tâches n'appartient au genre de tâches *Analyser*. Dans le genre de tâches *Produire*, le type de tâches *Concevoir un programme permettant de réaliser une tâche t* est très majoritaire dans l'ensemble des EIAH, et même le seul présent dans Py. La situation est presque identique dans CCP où la seule exception est le premier exercice de plusieurs saisons qui demande de copier un code valide proposé et de l'exécuter. Dans AP, pour les premiers exercices de chaque série, il est proposé dans l'énoncé un programme très proche

Table 1. Effectif des types de tâches de la catégorie *Programme* par EIAH.

Types de tâches		AP	CCP	Py
Analyser un code existant		0	0	0
Feuille blanche	Copier et exécuter	0	6	0
	Concevoir un programme	26	43	8
Code existant : Modifier un programme P' en P qui réalise une tâche t	P' est erroné	1	0	0
	P' réalise t' proche de t	15	0	0
	P' contient des trous	3	0	0
Nombre d'exercices décrits		45	49	8

du code à produire (même nombre de lignes, même structure...), ceci représente un tiers des exercices proposés. On rencontre de manière anecdotique un exercice basé sur la correction d'un code erroné et trois exercices avec un code à trous à compléter. On peut constater que ces trois EIAH exploitent peu la diversité des types de tâches identifiés dans les moyens d'enseignement.

Dans la Table 2 nous présentons les différents types de tâches, relatifs aux variables, boucles et IC, présents dans les trois environnements.

Table 2. Effectif des types de tâches (hors catégorie *Programme*) par EIAH.

Type de tâches	AP	CCP	Py
Produire une boucle bornée	21	105	10
Utiliser la valeur d'une expression	5	0	0
Affecter une valeur à une variable, initialisation	9	24	3
Affecter une valeur à une variable, modification	0	13	0
Utiliser une variable dans une expression	6	0	3
Ajouter une ou des instructions à une boucle existante	2	0	0
Concevoir une IC	12	16	2
Déterminer les instructions à exécuter à chaque itération	1	2	0
Modifier les instructions qui font évoluer la condition d'arrêt	1	0	0
Produire une boucle non bornée	6	12	2

Quelques observations sur les différents EIAH.

- Dans CCP, il est impossible de passer un paramètre dans les fonctions de déplacement (`gauche()` et `droite()`) ce qui rend nécessaire d'utiliser une boucle dès que le déplacement dépasse deux occurrences et explique la surreprésentation du type de tâches *Produire une boucle bornée*.
- Pour contraindre l'utilisation de certaines structures de contrôle, divers moyens sont mis en œuvre. Les trois EIAH utilisent la limitation du

nombre de lignes de code. CCP et AP interdisent parfois explicitement l'utilisation d'une structure en renvoyant un message d'erreur spécifique, par exemple si l'on écrit `for` dans une situation où `while` est attendu. Enfin, Py est le seul environnement, parmi ceux étudiés, qui exploite l'aléatoire dans la tâche t à réaliser pour "forcer" l'utilisation d'une structure plutôt qu'une autre puisque le code produit doit fonctionner pour une tâche t non totalement connue à l'avance.
— La nature des exercices, déplacement d'un robot ou d'un avatar, dans CCP et Py, amène à devoir utiliser massivement des fonctions de déplacement, éventuellement avec paramètres, avant que la fonction ait été étudiée comme objet.

Éléments de synthèse. D'une manière globale on peut constater que les trois EIAH se situent plutôt dans une approche où c'est l'action qui prime et que l'articulation avec le bloc du *logos* (globalement peu présent) est laissée à la charge de l'utilisateur qui doit aller chercher l'information et se débrouiller pour l'articuler avec la tâche qu'il réalise. Ce constat amène à interroger le rôle dans l'apprentissage que peuvent, ou non, jouer ces EIAH. Ceci rejoint par ailleurs le questionnement posé dans [13]. Une analyse complémentaire, des trois EIAH, basée sur celle présentée dans cet article, est proposée dans [12].

5 Conclusion et perspectives

Dans cette contribution nous avons étudié la manifestation de la transposition didactique en étudiant les types de tâches présents dans un corpus de ressources d'apprentissage de la programmation. Nous avons proposé une structuration de ces types de tâches qui a été exploitée pour décrire et analyser trois EIAH d'apprentissage de Python.

Pour conclure, nous proposons trois perspectives. Premièrement, prolonger le travail initié dans cet article et dans [12], en réalisant un recensement et une étude systématique des EIAH se positionnant sur le même segment que ceux étudiés.

Deuxièmement, exploiter ces travaux comme guide pour la production ou l'évolution d'EIAH d'apprentissage de la programmation (diversification des tâches, proposition de parcours et de rétroactions, etc.).

Troisièmement, avec des finalités plus larges que l'unique étude d'EIAH, poursuivre la modélisation du savoir ébauchée dans cet article en complétant les praxéologies (en couverture en ajoutant différentes praxéologies, en profondeur en raffinant certains types de tâches, en complétude en ajoutant les techniques et le bloc du *logos*) et en intégrant un travail sur les erreurs. Ceci se faisant tout d'abord avec les praxéologies institutionnelles, puis en étudiant les praxéologies mises en œuvre par les apprenants. L'identification des praxéologies mises en œuvre peut notamment s'appuyer sur une étude des traces d'activités des utilisateurs, en particulier les différents codes successifs produits pour parvenir à

la réponse. L'étude et la compréhension des écarts entre praxéologies institutionnelles et praxéologies des apprenants est à la fois un enjeu didactique et de formation des enseignants.

Références

1. Modulo - Catalogue de ressources informatique. https ://apprendre.modulo-info.ch/ (2022)
2. Bays, S. : Numérique et sciences informatiques, 1ère. Prépas Sciences, ellipses edn. (2019)
3. Branthôme, M. : Apprentissage de la programmation informatique à la transition collège-lycée. Revue STICEF **28**(3) (2021). https://doi.org/10.23709/STICEF.28.3.1, `http://sticef.org/num/vol2021/28.3.1.branthome/28.3.1.branthome.htm`, medium : pdf Publisher : STICEF Version Number : 1
4. Canu, C. : Numérique et sciences informatiques, 1ère. Ellipses edn. (2019)
5. Chaachoua, H. : T4TEL, un cadre de référence didactique pour la conception des EIAH. In : Pilet, J., Vendeira, C. (eds.) Actes du séminaire de didactique des mathématiques 2018. pp. 8–25. IREM de Paris - Université Paris Diderot, Paris (2018), `https://hal.archives-ouvertes.fr/hal-02421410/document`
6. Chaachoua, H., Crisci, R., Tchounikine, P. : Un modèle praxéologique de référence pour des praxéologies mixtes dans des tâches de programmation. In : Florensa, I., Ruiz Munzon, N. (eds.) Pré-actes de CITAD7. pp. 361–372. Barcelona, Spain (2022)
7. Chevallard, Y. : La transposition didactique. Grenoble : La pensée sauvage (1985)
8. Chevallard, Y. : L'analyse des pratiques enseignantes en théorie anthropologique du didactique. Recherches en Didactique des Mathématiques **19**(2), 221–265 (1999)
9. Couderette, M. : Enseignement de l'algorithmique en classe de seconde : une introduction curriculaire problématique. Annales de Didactique et de Sciences Cognitives. Revue internationale de didactique des mathématiques (21), 267–296 (2016)
10. Divoux, C., Gouygou, C., Jospeh, N., Lassus, G., Magnier, L., Saës, G. : Cahier d'algorithmique 2de. Belin : éducation, belin/humensis edn. (2018)
11. Groz, B., Waller, E., Nancel, M., Beaudouin-Lafon, M., Marce, O. : NSI 1ère. Hachette (2021)
12. Jolivet, S., Dechaux, E., Gobard, A.C., Wang, P. : Description et analyse de trois EIAH d'apprentissage de Python. In : Actes de l'atelier APIMU : 11ème Conférence sur les Environnements Informatiques pour l'Apprentissage Humain. Brest (2023)
13. Jolivet, S., Grugeon-Allys, B. : Modélisation de parcours d'apprentissage adaptés à l'apprenant dans un EIAH. In : Florensa, I., Ruiz Munzón, N. (eds.) Pré-actes de la 7e conférence internationale sur la théorie anthropologique du didactique (CITAD7). pp. 92–106. Barcelonne (2022)
14. Sablayrolles, L., Lefevre, M., Guin, N., Broisin, J. : Design and Evaluation of a Competency-Based Recommendation Process. In : International Conference on Intelligent Tutoring Systems. pp. 148–160. Springer (2022)
15. Strock, J.M., Artaud, M. : Du logos des organisations algorithmiques dans l'enseignement secondaire. Educação Matemática Pesquisa : Revista do Programa de Estudos Pós-Graduados em Educação Matemática **21**(4) (2019), number : 4