Modélisation et décision de rétroactions épistémiques dans l'environnement AlgoPython

Sébastien Jolivet[1,2][0000−0003−3915−8465], Badmavasan Kirouchenassamy[2][0009−0003−6502−154X], Amel Yessad[2][0000−0001−7575−6433], and Vanda Luengo[2][0000−0003−3915−8465]

[1] IUFE & TECFA, , Université de Genève, Suisse
sebastien.jolivet@unige.ch
[2] Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
{prenom.nom}@lip6.fr

Abstract. AlgoPython est une plateforme d'apprentissage de la programmation en Python. Dans sa version actuelle elle propose des rétroactions limitées aux erreurs syntaxiques et à la validation de la tâche. Dans cet article nous étudions les différents éléments nécessaires à la mise en place d'un système de décision de rétroactions épistémiques adaptées à l'apprenant. Après avoir présenté le problème de la décision des rétroactions, nous analysons un algorithme de type *Reinforcement learning* pour l'aborder. Nous définissons alors deux éléments fondamentaux : la définition des éléments à prendre en compte pour permettre la décision (éléments relatifs à l'apprenant et à sa production) et la modélisation et la production des rétroactions. Enfin nous proposons une première preuve de concept permettant d'évaluer la production des rétroactions ainsi que le choix d'un algorithme de type *Reinforcement learning* pour la décision de rétroactions.

Keywords: apprentissage de la programmation · rétroactions épistémiques · décision · apprentissage par renforcement

1 Introduction

AlgoPython [3] est un environnement d'apprentissage de la programmation en Python qui, dans sa version actuelle, propose des rétroactions limitées. Elles portent essentiellement sur la syntaxe ou la (non)validation de la réponse, et ne sont pas adaptées à l'apprenant. Or, la littérature met en évidence deux éléments ; d'une part les rétroactions les plus efficaces sont celles liées à la tâche [30] et d'autre part il y a un impact positif de l'adaptation des EIAH aux caractéristiques des apprenants [27, 29, 1].

Dans le cadre de cet article, nous nous intéressons à la mise en place d'un système permettant de proposer des rétroactions relatives aux tâches et adaptées aux apprenants dans AlgoPython.

[3] AlgoPython : https://www.algopython.fr, consulté le 3 mai 2025

Pour illustrer la complexité du problème considérons l'exemple suivant. Un apprenant doit réaliser l'exercice proposé Fig. 1. Comme réponse il soumet le code présenté Fig. 2.

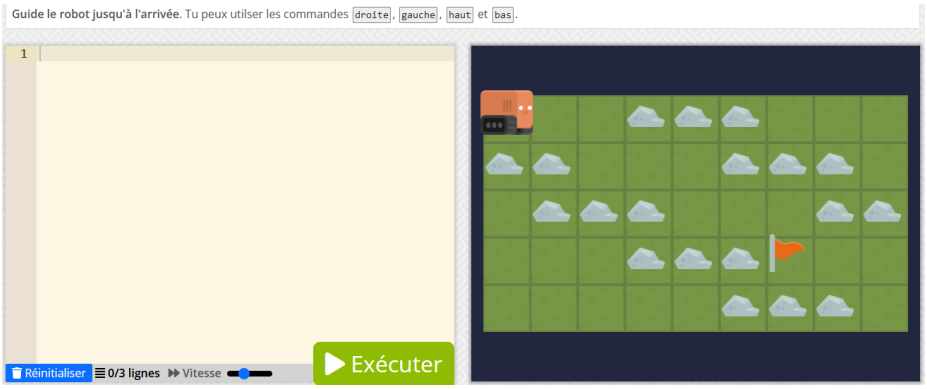


Fig. 1: Exercice de déplacement du robot

Même pour un exercice aussi simple, et pour une réponse aussi courte, les critères pouvant guider le contenu et le choix de la rétroaction sont nombreux : accompagner la modélisation de la situation ; aider tout d'abord à corriger l'erreur relative au compteur de la boucle ou au corps de la boucle ; proposer un rappel de cours ou proposer un exemple proche de la situation à traiter ; signaler explicitement à l'apprenant une erreur ou le laisser interpréter l'aide qu'on lui propose ; etc. Par exemple, les trois rétroactions proposées Fig. 8, Fig. 9 et Fig. 10 ont toutes une pertinence potentielle par rapport à la réponse de l'apprenant.

```
for i in range(4):
    droite(2)
bas(2)
```

```
for k in range(3):
    droite(2)
    bas(1)
```

Fig. 2: Par rapport à l'exercice présenté Fig. 1 : solution proposée par un apprenant, à gauche, et solution attendue, à droite.

S'ajoute aussi l'observation de Nogry & al. "Dans le cadre des EIAH, l'objectif à atteindre comporte deux niveaux : l'apprentissage (de la discipline enseignée et non de la manipulation du système) et la réalisation de tâches proposées par le système (résolution de problème, recherche d'informations, simulations...). Même si ces niveaux sont connectés, il n'y a pas de lien direct entre la réalisation de la tâche et l'apprentissage effectif, un échec dans la réalisation de la

tâche peut, dans certaines conditions, être bénéfique pour l'apprentissage" [19] (p. 267).

La complexité mise en évidence par cet exemple peut se résumer ainsi : *quelle est la bonne rétroaction à proposer à un apprenant, à un moment donné de son apprentissage, par rapport à la réponse qu'il a proposée pour permettre la réalisation de la tâche et l'apprentissage ?*

L'objectif global de notre travail est d'apporter une réponse à cette question en implémentant un système de décision de rétroactions épistémiques et adaptées, soit des rétroactions centrées sur la tâche à réaliser, les connaissances en jeu et qui prennent en compte l'apprenant et son activité. Nous présentons de premiers éléments dans cette contribution.

2 Questions de recherche

Il existe une littérature abondante relative aux rétroactions. On pourra par exemple se référer à [11] ou [30] pour les rétroactions en éducation en général, [23] propose un focus particulier sur les rétroactions formatives et [15] propose une méta-analyse des effets des rétroactions dans un environnement numérique. [24] et [14] proposent des éléments spécifiques aux rétroactions dans le contexte de l'apprentissage de la programmation.

La question identifiée à la fin de l'introduction peut être abordée selon différents angles. Au regard des données déjà disponibles, et des traces accessibles depuis la plateforme, nous avons fait le choix de nous intéresser particulièrement aux productions de l'apprenant (plusieurs milliers de soumissions de code à différents exercices déjà disponibles au commencement de notre travail) et aux informations que l'on peut en déduire. Dans cet article nous nous concentrons sur la question des erreurs. Elles vont être exploitées à la fois comme critère de décision de la rétroaction et participer à caractériser le profil de l'apprenant.

Les différentes étapes du processus de décision sont décrites dans la figure Fig. 3. L'étape 3 n'est pas traitée dans cet article, nous formalisons les étapes 1 et 2 au travers des questions de recherche suivantes.

(QR1) : quelle modélisation des rétroactions pour permettre leur production et leur décision ?

(QR2) : quel type de modèles de décision pour proposer des rétroactions épistémiques adaptées à l'apprenant ?

La seconde est travaillée dans les sections 4 et 5. Les réponses apportées permettent d'aborder la première question dans la section 6.

3 Cadre théorique

Répondre aux questions de recherches 1 et 2 amène à représenter le savoir en jeu, et sa mobilisation, dans les tâches proposées dans AlgoPython. Pour disposer d'un cadre permettant d'aborder de manière commune ces différentes questions

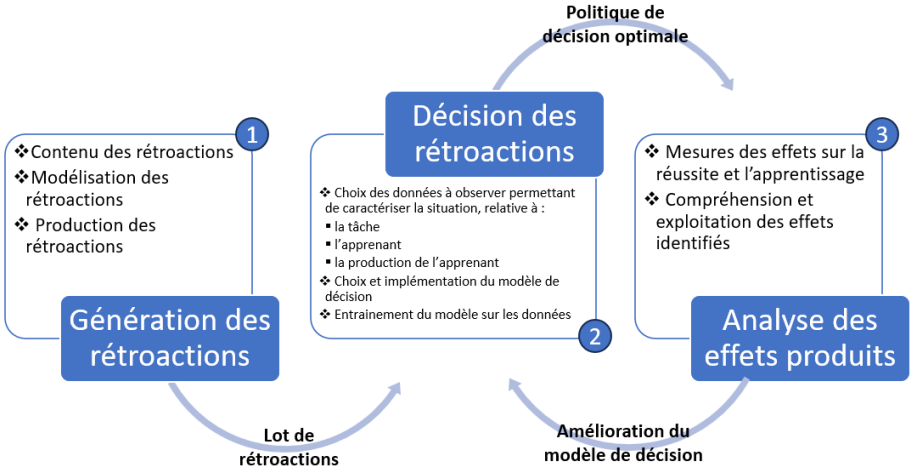


Fig. 3: Étapes pour la mise en œuvre d'un système de décision des rétroactions

nous avons fait le choix d'exploiter la modélisation de l'activité de programmation [4] telle que proposée dans [13]. Appuyée sur le modèle praxéologique de la théorie anthropologique du didactique [7], cette modélisation met en évidence des types de tâches, la manière de les réaliser (nommées *techniques*) et les éléments de savoir (nommés *logos*) utiles pour justifier et guider la mise en œuvre des *techniques*. Ainsi, lorsque l'apprenant réalise une tâche dans AlgoPython, cette tâche appartient à un type de tâches, elle est réalisée par le moyen d'une technique qui mobilise différents éléments du logos. Par exemple lorsque l'apprenant doit réaliser la tâche *Guide le robot jusqu'à l'arrivée* dans le contexte défini (voir Fig. 1) il met en œuvre, notamment, le type de tâches *Concevoir une boucle bornée* qui va être réalisé à l'aide d'une technique pouvant être (partiellement) décrite de la manière suivante : *) identifier la ou les instructions à répéter **) déterminer le nombre de répétition ***) implémenter en Python. Les éléments du logos associés sont la définition d'une boucle bornée, la notion de corps d'une boucle, la notion d'itération et d'itérateur, etc. La figure Fig. 4 illustre cette approche pour deux types de tâches, l'un relatif aux boucles bornées et l'autre relatif aux algorithmes.

Le référentiel exploité permet de décrire l'activité attendue de l'apprenant mais pas les erreurs de celui-ci. Or, divers travaux relatifs aux rétroactions montrent la pertinence qu'il y a à prendre en compte l'erreur [5, 25].

Relativement au processus décrit figure Fig. 3 cette prise en compte peut se situer à la fois dans le contenu des rétroactions, dans la connaissance de l'apprenant et dans la production de l'apprenant. C'est donc un élément que nous considérons comme important pour notre travail.

Par ailleurs, dans la plupart des travaux disponibles relatifs aux erreurs dans l'apprentissage de la programmation il n'y a que des classifications d'assez haut

[4] Le référentiel est disponible à l'adresse https://link.infini.fr/ref_prog

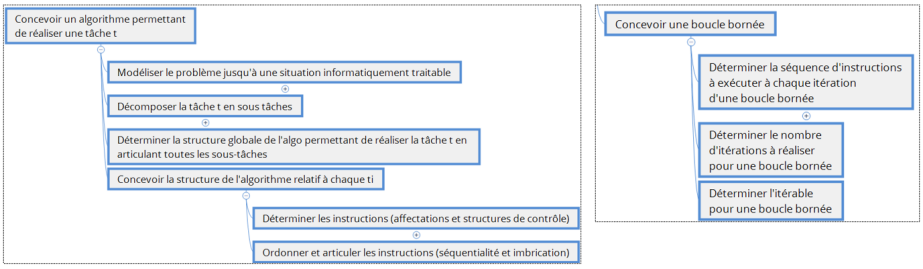


Fig. 4: Types de tâches *Concevoir un algorithme permettant de réaliser une tâche t* et *Concevoir une boucle bornée* ainsi que des ingrédients de leurs techniques respectives. Extrait de https://link.infini.fr/ref_prog

niveau [21]. Or, s'agissant de premiers apprentissages de la programmation, une caractérisation des erreurs trop globale (erreur sur la boucle par exemple) n'est pas efficiente pour orienter le choix de la rétroaction.

Pour conserver le cadre homogène que nous visons, nous exploitons le référentiel cité précédemment pour définir un ensemble d'erreurs. Pour cela, nous associons aux différents types de tâches des erreurs possibles. Par exemple, relativement aux types de tâches présentés Fig. 4, nous définissons les erreurs suivantes : par rapport aux boucles bornées (erreur relative au corps de la boucle ; erreur relative au nombre d'itérations ; erreur dans la définition de l'itérable) ; par rapport aux algorithmes (erreur dans la décomposition du problème en sous-tâches ; erreur dans le choix des structures de contrôle ; erreur dans l'ordre des instructions ; instruction manquante ; instruction inutile ; etc.).

4 Le problème de la décision des rétroactions

Le problème de la décision des rétroactions consiste à choisir, parmi un ensemble de rétroactions disponibles, la plus adaptée à l'apprenant. C'est-à-dire celle qui permet d'atteindre efficacement le ou les objectifs du dispositif (réussite de l'apprenant, meilleur apprentissage des concepts, réduction du temps d'apprentissage, réduction des abondons, etc.).

Or, ce problème est difficile puisque les données (l'état de l'apprenant, les erreurs produites, etc.) sur lesquelles s'appuie la décision sont souvent incertaines, bruitées et partielles. De plus, l'impact de la décision est, lui aussi, incertain. Ceci nous amène à considérer des modèles d'apprentissage automatique (AA) pour rechercher la meilleure association possible entre une situation (apprenant, production, tâche) et une rétroaction.

Ces modèles AA ont été abondamment utilisés dans le domaine des EIAH (en particulier dans les travaux des communautés AIED et EDM) pour la modélisation de l'apprenant [8, 31, 17], la prédiction de son comportement [16], l'adaptation des parcours d'apprentissage [1, 3], les tests adaptatifs [28], etc. Il existe trois

types de modèles d'AA : l'apprentissage non supervisé [10], l'apprentissage supervisé [9] et l'apprentissage par renforcement (Reinforcement Learning ou RL) [26].

Or, les éléments suivants suggèrent une bonne adéquation entre notre problème de décision de rétroactions et les caractéristiques d'un problème RL et :

- Pas de supervision, uniquement un signal de retour/récompense. Contrairement aux modèles d'AA supervisé, ici le modèle n'apprend pas sur des exemples annotés par un superviseur humain puisque les experts ont du mal à choisir la rétroaction la mieux adaptée à la situation de l'apprenant,
- Le retour est généralement retardé et non instantané. En effet, la rétroaction décidée par le système ne doit pas donner nécessairement lieu à une récompense immédiate puisque son effet ne peut être évalué positivement simplement du fait que l'apprenant ait réussi la tâche en cours, des effets sur les apprentissages à moyen et long terme peuvent être privilégiés,
- La prise en compte du temps (les expériences de l'apprenant sont ordonnées dans le temps, les variables ne sont pas indépendantes et identiquement distribuées, non i.i.d),
- Les actions de l'agent RL à l'instant t impactent les données reçues à l'instant $t+1$. En effet, une rétroaction décidée par le système peut influencer l'apprenant et ses futures tentatives de résolution. Ceci est une caractéristique intrinsèque des modèles RL qui les différencient des modèles supervisés ou non supervisés,
- L'état de l'environnement perçu par l'agent RL est incertain; Les informations observées sur l'apprenant sont généralement bruitées, incomplètes et incertaines.

Ceci justifie d'écarter la piste des modèles supervisés ou non supervisés et notre choix d'explorer l'approche RL pour apprendre un modèle de décision des rétroactions. Nous présentons maintenant plus précisément cette approche.

Les problèmes RL sont généralement modélisés comme des processus de décision markovien (en anglais Markov decision process, MDP). Un MDP est un quadruplet $\{S, A, T, R\}$ définissant :

- un ensemble d'états S qui peut être fini, dénombrable ou continu; cet ensemble définit l'environnement tel que perçu par l'agent RL ;
- un ensemble d'actions A qui peut être fini, dénombrable ou continu. L'agent RL interagit avec son environnement en décidant une action de cet ensemble. Dans notre cas, cet ensemble est le corpus des rétroactions disponibles ;
- une fonction de transition $T : S \times A \times S \rightarrow [0; 1]$; cette fonction définit l'effet des actions de l'agent RL sur l'environnement: $T(s, a, s')$ représente la probabilité de se retrouver dans l'état s' en effectuant l'action a, sachant que l'on était précédemment dans l'état s ;
- une fonction de récompense $R : S \times A \times S \rightarrow \mathbb{R}$; elle définit la récompense (positive ou négative) reçue par l'agent pour être passé de l'état s à s' en ayant effectué l'action a. Dans notre cas, la valeur de la récompense dépend de l'objectif de la politique de décision des rétroactions (réussite immédiate de l'apprenant, apprentissage des concepts de programmation, etc.).

Dans la suite nous présentons la construction de la partie $\{S, A\}$ du MDP. La partie $\{T, R\}$ n'est pas abordée dans cette contribution.

5 Définition de l'état de l'environnement du MDP

Un élément clé pour résoudre un problème RL est la définition de l'état de l'environnement. Nous avons choisi de le définir à l'aide du profil des erreurs de l'apprenant, de l'analyse de son code erroné et des caractéristiques de la tâche en cours. Nous ne revenons pas sur cette dernière, elle est basée sur le référentiel présenté section 3. Dans cette même section 3, nous avons défini un ensemble d'erreurs. L'enjeu est de relier un code erroné à une ou plusieurs de ces erreurs. La section suivante détaille les éléments mis en place pour y parvenir.

5.1 Identification des erreurs

Notre approche, pour identifier les erreurs dans un code soumis, est d'exploiter la représentation des codes sous forme d'un arbre syntaxique abstrait (AST) et l'algorithme de Zang et Shasha [32] (AZS). Nous illustrons le travail à l'aide des codes proposés Fig. 2.

Du code à la comparaison d'AST Dans sa version initiale AZS [5] permet de récupérer l'ensemble des opérations de modifications (organisées en trois opérateurs d'édition : insertion ; suppression ; mise à jour) nécessaires pour transformer un AST (celui de l'apprenant) en un autre (un code correct). La difficulté est d'interpréter ces modifications en terme d'erreurs de notre ensemble prédéfini d'erreurs. Nous revenons brièvement sur son traitement.

La bibliothèque Zhang-Shasha par défaut fournit des informations insuffisantes par rapport à nos besoins. Par exemple, si deux instructions conditionnelles se trouvent à des emplacements distincts dans le code, et qu'une erreur est présente dans l'une d'elles, on ne sait pas de laquelle il s'agit dans le code incorrect.

Pour répondre à cette limitation, une méthode a été développée pour extraire et structurer les informations de l'AST sous forme de nœuds adaptés (de l'arbre AST). Le résultat de ce travail est illustré dans les Fig. 5 et Fig. 6. De plus, l'algorithme de calcul de distance de Zhang-Shasha a été réimplémenté [6] afin d'intégrer le contexte dans lequel les opérations de transformation doivent être appliquées.

[5] https://github.com/timtadh/zhang-shasha/
[6] Ces éléments sont mis à disposition sur https://pypi.org/project/asterrdetection/

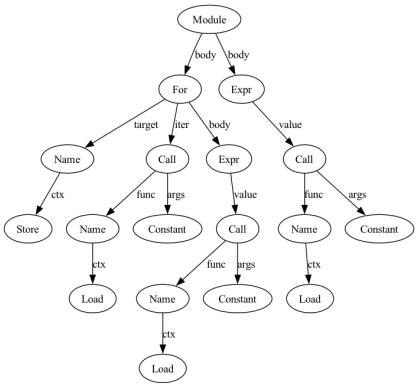


Fig. 5: AST sous sa forme originale

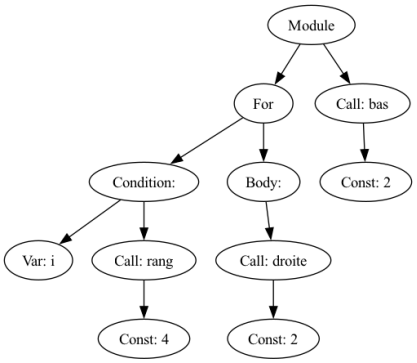


Fig. 6: AST reimplementé

En comparant le code incorrect au code correct à l'aide du calcul de la distance de Zhang-Shasha réimplémentée, il est possible de déterminer l'ensemble des opérations nécessaires pour transformer le code incorrect en la solution correcte. Voici ce qui est généré par la comparaison des deux codes de la Fig. 2 :

Listing 1.1: Liste des opérations d'édition à l'issue de notre réimplémentation de Zang-Shasha dans le cas des deux codes de la Fig. 2

```
{'type': 'update', 'path': ['Module', 'For[0]', 'Condition:[0]', 'Var: i[0]'], 'current': 'Var: i', 'new': 'Var: k'},
{'type': 'update', 'path': ['Module', 'For[0]', 'Condition:[0]', 'Call: range[1]', 'Const: 4[0]'], 'current': 'Const: 4', 'new': 'Const: 3'},
{'type': 'match', 'path': ['Module', 'For[0]', 'Condition:[0]', 'Call: range[1]'], 'current': 'Call: range', 'new': 'Call: range'},
{'type': 'match', 'path': ['Module', 'For[0]', 'Condition:[0]'], 'current': 'Condition:', 'new': 'Condition:'},
{'type': 'match', 'path': ['Module', 'For[0]', 'Body:[1]', 'Call: droite[0]', 'Const: 2[0]'], 'current': 'Const: 2', 'new': 'Const: 2'},
{'type': 'match', 'path': ['Module', 'For[0]', 'Body:[1]', 'Call: droite[0]'], 'current': 'Call: droite', 'new': 'Call: droite'},
{'type': 'insert', 'path': ['Module', 'For[0]', 'Body:[1]', 'Call: bas[1]', 'Const: 1[0]'], 'current': None, 'new': 'Const: 1'},
{'type': 'insert', 'path': ['Module', 'For[0]', 'Body:[1]', 'Call: bas[1]'], 'current': None, 'new': 'Call: bas'},
{'type': 'match', 'path': ['Module', 'For[0]', 'Body:[1]'], 'current': 'Body:', 'new': 'Body:'},
{'type': 'match', 'path': ['Module', 'For[0]'], 'current': 'For', 'new': 'For'},
{'type': 'delete', 'path': ['Module', 'Call: bas[1]', 'Const: 2[0]'], 'current': 'Const: 2', 'new': None},
{'type': 'delete', 'path': ['Module', 'Call: bas[1]'], 'current': 'Call: bas', 'new': None},
{'type': 'match', 'path': ['Module'], 'current': 'Module', 'new': 'Module'}
```

Des AST comparés à l'identification des erreurs Les informations (Voir Listing 1.1) obtenues ne sont pas directement interprétables du point de vue des erreurs. Par exemple, une opération de suppression d'un nœud suivie d'une insertion d'un autre nœud peut indiquer que la section de code concernée n'est pas située au bon endroit. Un autre exemple est le cas où le code apprenant et le code correct diffèrent notamment dans le nommage d'une variable, ceci ne doit pas être interprété comme une erreur alors que des modifications remontent lors de la comparaison des AST.

Pour rendre ces opérations exploitables en tant qu'erreurs logiques de programmation nous avons conçu une méthode permettant de transformer cette

liste brute d'opérations d'édition en une liste d'erreurs interprétables, en appliquant un ensemble de règles que nous ne développons pas ici. Le package Python développé est rendu public sur pypi.org et est disponible ici [7].

Voici le résultat obtenu après l'application de ces règles avec des erreurs interprétables et compréhensibles :

Listing 1.2: Liste des erreurs de l'apprenant à partir des opérations de modification du Listing 1.1

```
('MISSING_CALL_INSTRUCTION', 'BAS', 'Module > For > Body: > Call: bas'),
('MISSING_CONST_VALUE', '1', 'Module > For > Body: > Call: bas > Const: 1')
('UNNECESSARY_CALL_STATEMENT', 'BAS', 'Module > Call: bas')
('UNNECESSARY_CONST_VALUE', '2', 'Module > Call: bas > Const: 2')
('CONST_VALUE_MISMATCH', 'Const: 4', 'Const: 3', 'Module > For > Condition: > Call: range > Const: 4')
```

5.2 Prise en compte de l'apprenant

L'identification des erreurs dans un code, telle que présentée dans la section précédente, donne une information ponctuelle sur une partie de l'activité de l'apprenant, elle n'est cependant pas nécessairement représentative de son profil. Nous intégrons donc, dans la définition de l'état de l'environnement du système de décision, un profil de l'apprenant. L'approche mise en œuvre est la définition d'un *profil des erreurs de l'apprenant* (ses limites sont discutées dans la conclusion). Nous nous inspirons des approches *Buggy Model* [6] en définissant un vecteur qui enregistre les fréquences pondérées des erreurs réalisées par l'apprenant. La pondération est faite selon des considérations temporelles (les erreurs récentes sont plus pondérées que celles des soumissions antérieures). Ce *profil des erreurs de l'apprenant* est mis à jour grâce à la méthode de détection des erreurs présentée section 5.1. Il constitue un composant important de l'état de l'environnement que perçoit l'agent RL et qui lui permet d'apprendre par essais-erreurs (explore-exploit tradeoff) une politique optimale de décision des rétroactions.

6 Définition de l'ensemble des actions du MDP : modélisation et production des rétroactions

Les travaux relatifs aux rétroactions dans l'apprentissage sont nombreux. Concernant plus spécifiquement les rétroactions dans le contexte de l'apprentissage de la programmation nous pouvons citer le travail de Keunig et al. [14], qui propose une revue systématique de la littérature sur la génération automatique de rétroactions pour des exercices de programmation, et l'utilisation des travaux de Narciss [18] par Branthöme dans le cadre de l'environnement Pyrates [4].

Différents travaux ([2], [20]) visent actuellement à la mise en place de bot permettant de générer de manière dynamique des rétroactions à l'aide de LLM. Nous avons fait le choix de ne pas suivre cette voie principalement pour pouvoir contrôler expérimentalement nos choix. En particulier vis à vis du cadre

[7] urlhttps://pypi.org/project/asterrdetection/

théorique proposé et du modèle de décision que nous souhaitons mettre en place. En effet, la génération dynamique avec LLM a comme conséquence que la décision de la rétroaction est une boîte noire, alors que c'est une de nos questions de recherche. Nous envisageons dans une itération future, quand nous aurons validé expérimentalement cette étape de génération, de produire des rétroactions à partir des LLM tout en utilisant le modèle proposé pour ensuite disposer d'un lot de rétroactions statiques au service de la décision (voir Fig. 7) permettant le passage à l'échelle de notre approche. Dans notre système les rétroactions sont donc prédéfinies et annotées selon différentes caractéristiques, y compris dans l'objectif de pouvoir contrôler l'effet des différentes caractéristiques.

6.1 Modèle de rétroactions

Notre modèle de rétroaction est présenté Fig. 7. Nous détaillons tout d'abord les différents éléments du modèle et leur raison d'être, puis nous l'illustrons avec quelques exemples de rétroactions.

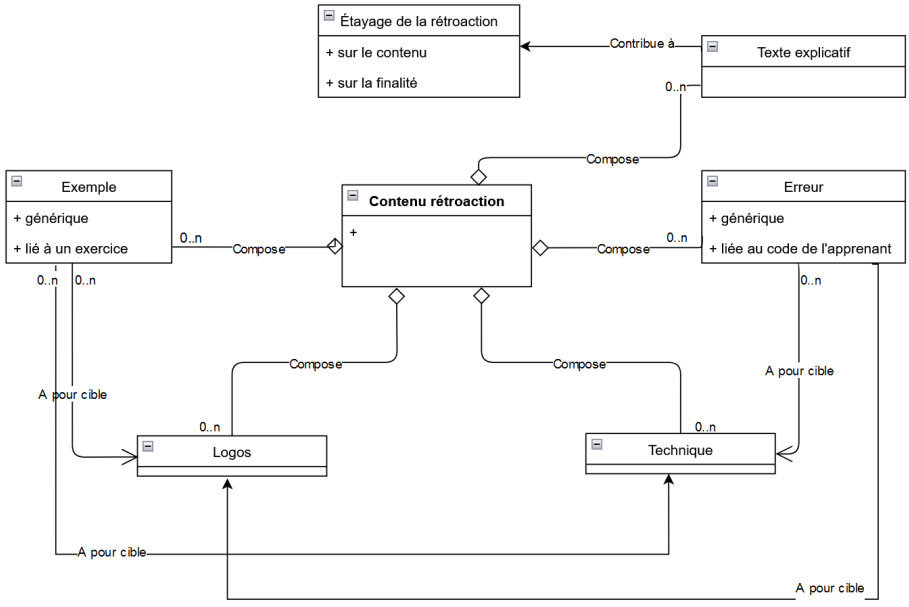


Fig. 7: Modélisation des rétroactions dans AlgoPython

Nos rétroactions sont en lien avec la tâche à réaliser. Le premier constituant est donc un élément relatif à la tâche, il porte sur une *technique* (i.e. "comment réaliser la tâche", voir Fig. 10) ou sur le *logos* (i.e. "savoir à mobiliser pour réaliser la tâche", voir Fig. 8).

Lorsqu'une rétroaction est proposée à l'apprenant il est confronté à plusieurs tâches : 1) comprendre son contenu ; 2) interpréter la ou les raisons pour lesquelles elle lui est proposée ; 3) l'exploiter pour agir sur son travail.

C'est l'intention d'étayage [22] de ces tâches qui nous amène à augmenter le contenu "technique / logos" des éléments suivants.

Pour aider à comprendre et exploiter le contenu de la rétroaction, nous pouvons :

- l'enrichir avec un exemple qui illustre le savoir ou une partie de la technique, voir Fig. 9 et Fig. 10
- pointer explicitement une erreur fréquente associée à la technique ou au logos évoqué, voir Fig. 8 et Fig. 9
- instancier le contenu de tout ou partie de la rétroaction à la tâche à résoudre en intégrant à son contenu des éléments de l'énoncé ou des éléments de la réponse de l'apprenant
- ajouter une phrase qui indique le statut d'un contenu et/ou la raison pour laquelle ce contenu est proposé

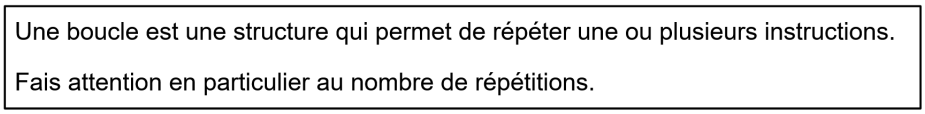


Fig. 8: Rétroaction du type : logos avec l'erreur "For Loop Incorrect number of iteration" pointée

6.2 Production et annotation des rétroactions

Une rétroaction est un assemblage de différents éléments identifiés Fig. 7. Pour produire les rétroactions relatives à la tâche, nous définissons la règle suivante : une rétroaction contient *a minima* un élément de type technique ou un élément de type logos (pour contrôler l'expérimentation et pour évaluer l'effet du contenu sur un apprenant nous faisons le choix, dans un premier temps, de ne pas produire de rétroaction contenant simultanément logos et technique).

Ce contenu peut être complété par les différents éléments présentés dans la section 6.1. Pour une série d'exercices d'AlgoPython nous produisons trois catégories de rétroactions : celles relatives au thème de la série en général ; celles relatives à une erreur spécifique de l'apprenant et celles spécifiques à un type d'énoncé.

Pour qu'une rétroaction puisse être décidée il est nécessaire qu'elle soit décrite. Cette description est basée d'une part sur les éléments décrits dans la Fig. 7 auxquels sont ajoutés les informations suivantes :

- Le ou les types de tâches auxquels associer la rétroaction.
- Le cas échéant, le ou les exercices spécifiques auxquels elle est destinée.
- La ou les erreurs (voir section 5.1) pour lesquelles elle fait sens.

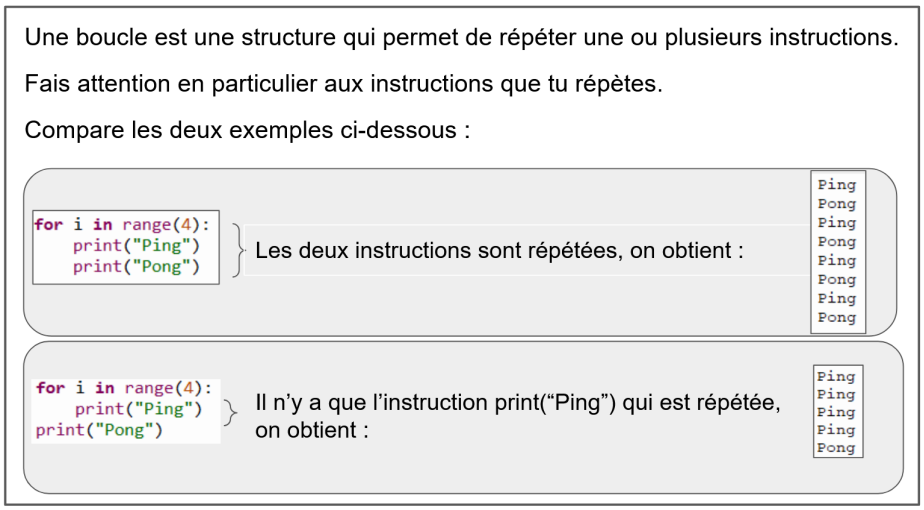


Fig. 9: Rétroaction du type : logos avec l'erreur "For loop body mismatch" pointée et un exemple non lié à un exercice spécifique

7 Preuve de concept sur l'adéquation de l'approche RL pour la décision des rétroactions

Nous avons mené un premier travail exploratoire pour vérifier qu'un framework RL est adéquat pour l'apprentissage d'une politique optimale de décision des rétroactions avant de le déployer dans la plateforme AlgoPython. Pour cela, nous avons lancé des simulations sur des données synthétiques. Dans le cadre de ces simulations, nous avons fait des choix pour la génération des données simulées :

- l'état de l'environnement est restreint à des informations sur la tâche en cours, aux erreurs de la production en cours et à l'état de l'apprenant,
- l'état d'un apprenant est simplifié et est défini uniquement par son profil des erreurs (généré de manière aléatoire),
- les différentes erreurs d'un profil des erreurs sont indépendantes les unes des autres, c'est à dire, l'apparition d'une erreur n'influence pas l'apparition d'une autre erreur,
- on considère 5 types d'apprenants différents, un apprenant appartient à un type si la majorité des fréquences de ses erreurs appartient à un intervalle prédéfini.

En plus de ces choix de simulation, nous générons les données selon la politique de décision (*Behavior policy*) suivante : on associe à chaque type d'apprenants simulé une rétroaction adaptée (voir section 6). Cette dernière, lorsqu'elle est décidée pour l'apprenant simulé à l'état S_t lui permet de corriger

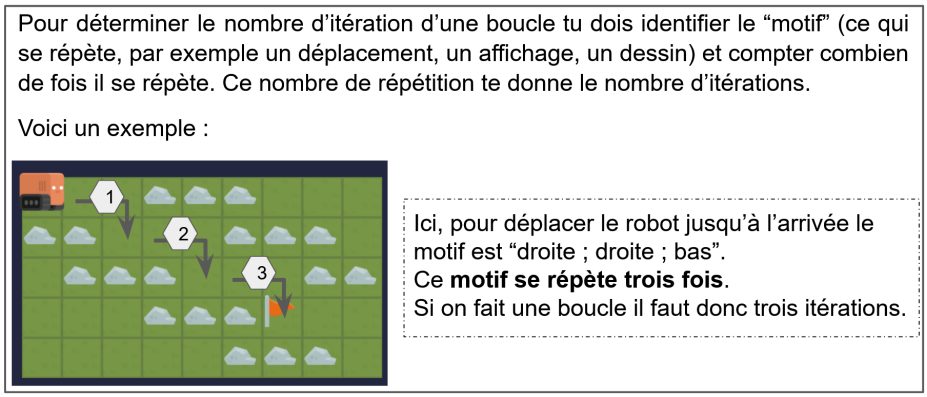


Fig. 10: Rétroaction du type : technique et exemple lié à un exercice spécifique

son erreur à l'état S_{t+1}, c'est à dire que la fréquence de cette erreur dans le profil des erreurs de l'apprenant diminue.

Bien entendu, ces choix de génération de données simulées ne correspondent aucunement à la réalité de données réelles issues d'expérimentations écologiques avec de vrais apprenants. Elles permettent néanmoins de vérifier l'efficacité des approches RL à apprendre une politique de décision cohérente avec les données simulées.

La question de recherche à laquelle nous voulons répondre à travers ces simulations, qui rejoint la QR2 (Voir section 2), est : **Est-ce qu'un modèle RL peut apprendre à décider la rétroaction la plus adaptée pour chaque type d'apprenants simulé ?**

Types d'apprenant simulés Nous avons défini 5 types d'apprenants bien distincts. Un apprenant appartient à un type i si la plupart des fréquences de ses erreurs appartiennent à l'intervalle $[1-0.2*(i), 1-0.2*(i-1)]$ sachant que la fréquence d'une erreur est une valeur dans $[0, 1]$. Concrètement, les apprenants du type 1 sont les apprenants qui font le plus d'erreurs et ceux du type 5 sont ceux qui font le moins d'erreurs.

Types de rétroactions Étant dans une cadre simulé, nous avons simplifié le problème d'adaptation d'une rétroaction à un apprenant comme suit : on associe à chaque type d'apprenants une et une seule rétroaction adaptée. Ainsi, toutes les autres rétroactions ne sont pas adaptées.

Fonction action-valeur ou récompenses cumulatives La fonction de récompense R a été définie comme suit :

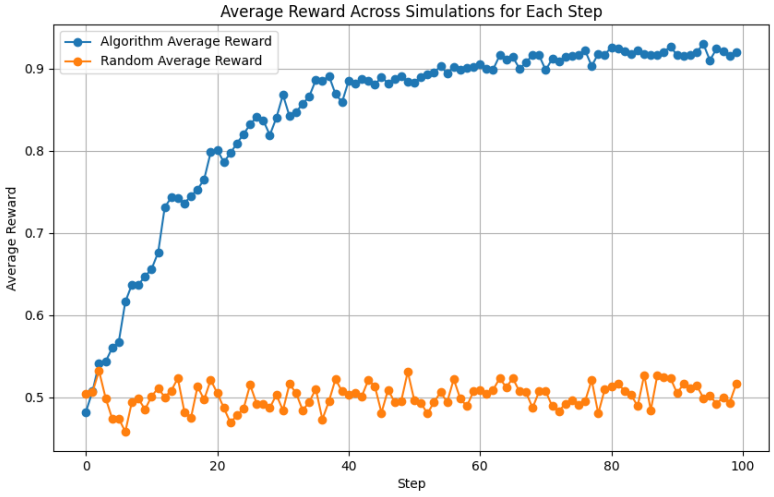
$$R(s_t, a_t) = \begin{cases} 1 & \text{Si la rétroaction } a_t \text{ est celle adaptée au type de} \\ & \text{l'apprenant dont le profil des erreurs est inclus dans } s_t \\ 0 & \text{Sinon} \end{cases}$$

De manière itérative, l'agent RL calcule les récompenses cumulatives reçues pour chaque décision de rétroaction a à un état s comme suit (t étant le nombre d'étapes d'apprentissage) :

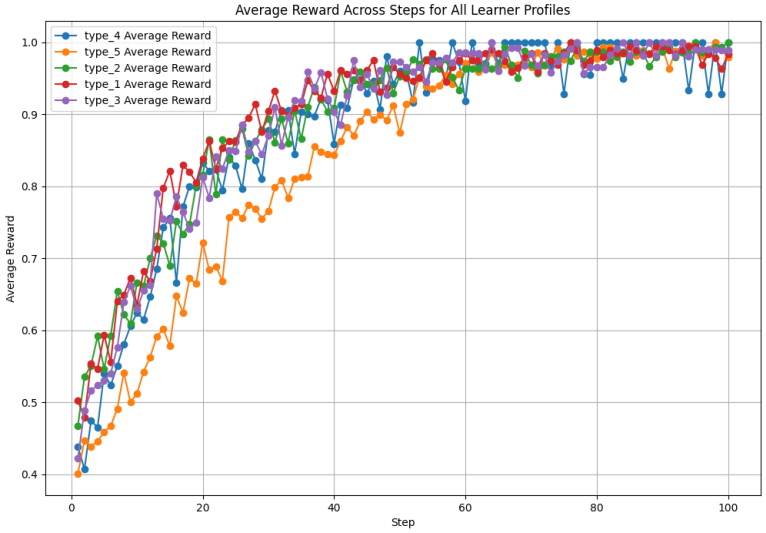
$$\begin{aligned} Q_{t+1}(s,a) &= \frac{1}{t}\left(r_1(s,a) + r_2(s,a) + \cdots + r_t(s,a)\right) \\ &= \frac{1}{t}\left((t-1)Q_t(s,a) + r_t(s,a)\right) \\ &= \frac{1}{t}\left(tQ_t(s,a) + r_t(s,a) - Q_t(s,a)\right) \\ &= Q_t(s,a) + \frac{1}{t}\left(r_t(s,a) - Q_t(s,a)\right) \end{aligned}$$

C'est la mise à jour itérative de cette fonction de valeur Q qui permet à l'agent RL d'apprendre à décider la bonne rétroaction à chaque état de l'environnement, c'est à dire, la rétroaction qui maximise les récompenses reçues. Dans cette simulation, une récompense est reçue lorsque la rétroaction permet de corriger une ou plusieurs erreurs de l'apprenant simulé.

Modèles RL utilisés Pour répondre à notre (QR2.1), nous avons testé deux modèles RL : Linear Contextual Multi-Armed Bandits avec Thompson Sampling ; Neural Contextual Multi-Armed Bandits avec Thompson Sampling. Nous présentons ici les résultats du modèle *Linear Contextual Multi-Armed Bandits avec Thompson Sampling* mais les résultats obtenus avec l'autre modèle testé sont similaires. Pour entraîner ce modèle sur nos données simulées, nous avons lancé 1000 simulations. Dans chaque simulation, nous avons généré 100 états de manière aléatoire en veillant à ce que chaque type d'apprenant soit représenté de manière équitable. Puis, nous avons calculé la moyenne des récompenses sur l'ensemble des simulations. Fig. 11(a) (respectivement la Fig. 11(b)) montre l'évolution de la moyenne des récompenses pour l'ensemble des types d'apprenant (respectivement par type d'apprenants) au fur et à mesure de l'apprentissage. On constate que la moyennes des récompenses augmente et se stabilise rapidement autour de 10 pas d'apprentissage. Pour mieux apprécier ce résultat, nous avons représenté par la courbe orange la moyenne des récompenses obtenues avec une politique de décision aléatoire sur les 100 pas de temps (Fig. 11(a)). On peut constater que la politique RL performe beaucoup mieux que la politique aléatoire et que la politique apprise (target policy) et la politique de génération des données (behavior policy) sont très proches. Ainsi, les résultats de ces premières simulations montrent qu'il est possible d'apprendre à un modèle RL la



(a) Moyenne des récompenses



(b) Moyenne des récompenses par type d'apprenants

Fig. 11: Linear Contextual Multi-Armed Bandits avec Thompson Sampling

politique de décision qui a été utilisée pour la génération des données. Ces résultats renforcent notre hypothèse sur l'adéquation des modèles RL à la décision des rétroactions adaptées.

8 Conclusion et perspectives

Dans cet article nous avons présenté les choix et premiers éléments mis en place pour permettre de décider des rétroactions épistémiques dans l'environnement AlgoPython. Pour répondre à la première question de recherche (QR1), nous avons proposé un modèle de rétroactions permettant à la fois leur production et leur annotation.

En ce qui concerne la deuxième question de recherche (QR2), nous avons cherché à y répondre en explorant une modélisation de la décision de type RL avec une définition de l'état de l'environnement, notamment basée sur les erreurs de l'apprenant. À cette occasion nous avons aussi produit une méthode d'analyse des codes soumis avec la mise à disposition d'un package Python original [8]. Enfin, nous avons fait une première évaluation exploratoire avec des données simulées afin de vérifier que le modèle converge de façon satisfaisante.

Ces travaux ont été réalisés en exploitant le référentiel de types de tâches de programmation proposé dans [13] qui a fournit un socle commun pour aborder nos différentes questions.

Nous engageons maintenant une première expérimentation, avec une dizaine d'enseignants, avec un premier déploiement de l'algorithme qui va travailler sur la décision des 75 rétroactions produites pour les séries *instructions*, *boucle For* et *variables* d'AlgoPython. Cette expérimentation en conditions réelles nous permettra d'obtenir un premier ensemble de données relatives à la version d'AlgoPython augmentée par les rétroactions.

Le travail de modélisation des rétroactions va être enrichi et exploité à l'aide de trois travaux, menés en parallèle, qui débutent :

- faire évaluer par des enseignants et des élèves le contenu des rétroactions produites
- explorer l'exploitation d'IA génératives avec une approche RAG et chain-of-tought [12] pour permettre une production (semi)automatisée de contenus de rétroactions
- améliorer le dispositif d'annotation d'erreur présenté dans la section 4.1

Ils doivent permettre d'améliorer le contenu des rétroactions et la connaissance de l'apprenant permettant à l'algorithme de décision des rétroactions de travailler dans des conditions optimisées du point de vue de l'apprentissage humain.

Au niveau du modèle décisionnel, la suite du travail porte d'une part sur la fonction de récompense (section 4) et d'autre part sur la modélisation de l'apprenant (section 5.2) pour mieux informer l'agent RL. En effet, actuellement

[8] https://pypi.org/project/asterrdetection/

nous modélisons uniquement le profil des erreurs des apprenants qui n'informe pas sur l'évolution de leur niveau de maîtrise des connaissances. Nous souhaitons traiter cette limite en explorant des modèles de traçage des connaissances tel que le modèle BKT (Bayesian Knowledge Tracing) [31].

References

1. Aleven, V., McLaughlin, E.A., Glenn, R.A., Koedinger, K.R.: Instruction based on adaptive learning technologies. Handbook of research on learning and instruction **2**, 522–560 (2016)
2. Balse, R., Valaboju, B., Singhal, S., Warriem, J.M., Prasad, P.: Investigating the potential of gpt-3 in providing feedback for programming assessments. In: Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1. p. 292–298. ITiCSE 2023, Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3587102.3588852, https://doi.org/10.1145/3587102.3588852
3. Bassen, J., Balaji, B., Schaarschmidt, M., Thille, C., Painter, J., Zimmaro, D., Games, A., Fast, E., Mitchell, J.C.: Reinforcement learning for the adaptive scheduling of educational activities. In: Proceedings of the 2020 CHI conference on human factors in computing systems. pp. 1–12 (2020)
4. Branthome, M.: Apprentissage de la programmation informatique: analyses et ressources pour accompagner la transition collège-lycée. PhD Thesis, Université de Bretagne occidentale-Brest (2023), https://theses.hal.science/tel-04397329/
5. Brooks, C., Carroll, A., Gillies, R.M., Hattie, J.: A matrix of feedback for learning. Australian Journal of Teacher Education (Online) **44**(4), 14–32 (2019)
6. Brown, J.S., Burton, R.R.: Diagnostic models for procedural bugs in basic mathematical skills. Cognitive science **2**(2), 155–192 (1978)
7. Chevallard, Y.: Concepts fondamentaux de la didactique : perspectives apportées par une approche anthropologique. Recherche en didactique des mathématiques **12**(1), 73–112 (1992)
8. Corbett, A.T., Anderson, J.R.: Knowledge tracing: Modeling the acquisition of procedural knowledge. User modeling and user-adapted interaction **4**, 253–278 (1994)
9. Cunningham, P., Cord, M., Delany, S.J.: Supervised learning. In: Machine learning techniques for multimedia: case studies on organization and retrieval, pp. 21–49. Springer (2008)
10. Ghahramani, Z.: Unsupervised learning. In: Summer school on machine learning, pp. 72–112. Springer (2003)
11. Hattie, J., Timperley, H.: The power of feedback. Review of educational research **77**(1), 81–112 (2007), publisher: Sage Publications Sage CA: Thousand Oaks, CA
12. Jacobs, S., Jaschke, S.: Leveraging Lecture Content for Improved Feedback: Explorations with GPT-4 and Retrieval Augmented Generation. In: 2024 36th International Conference on Software Engineering Education and Training (CSEE&T). pp. 1–5. Germany (2024). https://doi.org/10.1109/CSEET62301.2024.10663001, http://arxiv.org/abs/2405.06681, arXiv:2405.06681 [cs]
13. Jolivet, S., Dechaux, E., Gobard, A.C., Wang, P.: Construction et exploitation d'un référentiel de types de tâches d'apprentissage de la programmation. In: Iksaal, S., Pélissier, C., Gilliot, J.M., Marzin-Janvier

(eds.) Actes de la onzième Conférence sur les Environnements Informatiques pour l'Apprentissage Humain (EIAH). pp. 154–165. Brest (2023), https://eiah2023.sciencesconf.org/data/pages/actesEIAH2023.pdf

14. Keuning, H., Jeuring, J., Heeren, B.: A Systematic Literature Review of Automated Feedback Generation for Programming Exercises. ACM Transactions on Computing Education **19**(1), 1–43 (Jan 2019). https://doi.org/10.1145/3231711, https://dl.acm.org/doi/10.1145/3231711
15. Van der Kleij, F.M., Feskens, R.C.W., Eggen, T.J.H.M.: Effects of Feedback in a Computer-Based Learning Environment on Students' Learning Outcomes: A Meta-Analysis. Review of Educational Research **85**(4), 475–511 (Dec 2015). https://doi.org/10.3102/0034654314564881, http://journals.sagepub.com/doi/10.3102/0034654314564881
16. Lopez, M.I., Luna, J.M., Romero, C., Ventura, S.: Classification via clustering for predicting final marks based on student participation in forums. International Educational Data Mining Society (2012)
17. Millán, E., Loboda, T., Pérez-De-La-Cruz, J.L.: Bayesian networks for student model engineering. Computers & Education **55**(4), 1663–1683 (2010)
18. Narciss, S.: Designing and Evaluating Tutoring Feedback Strategies for digital learning environments on the basis of the Interactive Tutoring Feedback Model. Digital Education Review **23** (2013)
19. Nogry, S., Jean-Daubias, S., Ollagnier-Beldame, M.: Évaluation des EIAH : une nécessaire diversité des méthodes. In: Acte du colloque "Technologies de l'Information et de la Connaissance dans l'Enseignement Supérieur et l'Industrie". pp. 265–271. Compiègne, France (2004), https://edutice.archives-ouvertes.fr/edutice-00000729/document
20. Pankiewicz, M., Baker, R.: Large language models (gpt) for automating feedback on programming assignments. In: 31st International Conference on Computers in Education (12 2023)
21. Qian, Y., Lehman, J.: Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. ACM Transactions on Computing Education **18**(1), 1–24 (Dec 2017). https://doi.org/10.1145/3077618, https://dl.acm.org/doi/10.1145/3077618
22. Quintana, C., Reiser, B.J., Davis, E.A., Krajcik, J., Fretz, E., Duncan, R.G., Kyza, E., Edelson, D., Soloway, E.: A Scaffolding Design Framework for Software to Support Science Inquiry. Journal of the Learning Sciences **13**(3), 337–386 (Jul 2004). https://doi.org/10.1207/s15327809jls1303_4, http://www.tandfonline.com/doi/abs/10.1207/s15327809jls1303_4
23. Shute, V.J.: Focus on Formative Feedback. Review of Educational Research **78**(1), 153–189 (Mar 2008). https://doi.org/10.3102/0034654307313795
24. Singh, R., Gulwani, S., Solar-Lezama, A.: Automated feedback generation for introductory programming assignments. In: Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 15–26. ACM, Seattle Washington USA (Jun 2013). https://doi.org/10.1145/2491956.2462195
25. Small, M., Lin, A.: Instructional feedback in mathematics. In: Lipnevich, A., Smith, J. (eds.) The Cambridge handbook of instructional feedback, pp. 169–190. Cambridge University Press (2018)
26. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)

27. Vandewaetere, M., Desmet, P., Clarebout, G.: The contribution of learner characteristics in the development of computer-based adaptive learning environments. Computers in Human Behavior **27**(1), 118–130 (2011)
28. Vie, J.J., Popineau, F., Bruillard, É., Bourda, Y.: A review of recent advances in adaptive assessment. Learning analytics: Fundaments, applications, and trends: A view of the current state of the art to enhance e-learning pp. 113–142 (2017)
29. Walkington, C.A.: Using adaptive learning technologies to personalize instruction to student interests: The impact of relevant contexts on performance and learning outcomes. Journal of educational psychology **105**(4), 932 (2013)
30. Wisniewski, B., Zierer, K., Hattie, J.: The power of feedback revisited: A meta-analysis of educational feedback research. Frontiers in Psychology **10**, 3087 (2020). https://doi.org/https://doi.org/10.3389/fpsyg.2019.03087, publisher: Frontiers
31. Yudelson, M.V., Koedinger, K.R., Gordon, G.J.: Individualized bayesian knowledge tracing models. In: Artificial Intelligence in Education: 16th International Conference, AIED 2013, Memphis, TN, USA, July 9-13, 2013. Proceedings 16. pp. 171–180. Springer (2013)
32. Zhang, K., Shasha, D.: Simple fast algorithms for the editing distance between trees and related problems. SIAM journal on computing **18**(6), 1245–1262 (1989), publisher: SIAM