Conception et évaluation du jeu sérieux Pyrates : agencement du milieu didactique pour la transition Scratch-Python

Matthieu Branthôme

Conception et évaluation du jeu sérieux *Pyrates :* agencement du milieu didactique pour la transition Scratch-Python

Matthieu BRANTHÔME[1]

[1] Université de Bretagne Occidentale, CREAD - EA 3875, Brest, France
matthieu.branthome@univ-brest.fr

Résumé. Cette communication présente la conception et l'évaluation de l'application en ligne *Pyrates*, un jeu sérieux qui vise l'introduction de la programmation Python en classe de seconde. Nous y exposons plus particulièrement l'aménagement du milieu didactique avec pour objectif de s'approcher des facilités offertes par les environnements de programmation basés sur les blocs. Afin d'évaluer nos choix, nous avons testé l'application sur le terrain auprès de 240 élèves de seconde et ainsi récolté environ 70.000 traces d'activités générées automatiquement. Ces traces sont complétées par un questionnaire en ligne. Nous montrons que certains choix de conception ont les effets attendus. Ainsi, la création d'un « mémo programmation » permet la découverte des notions algorithmiques tout en offrant un support de référence pour la syntaxe Python. La facilitation des copiés-collés depuis ce mémo limite la saisie au clavier et soutient la tâche de structuration des programmes. L'intégration d'un analyseur syntaxique conçu pour les débutants confère une grande autonomie aux élèves dans le traitement des erreurs. Cependant, d'autres choix ont des impacts plutôt néfastes. La création d'un panneau de commande pour l'exécution des programmes s'avère être entièrement au service d'une démarche de programmation par essais-erreurs ou de stratégies de « contournement didactique ».

Mots-clés : Apprentissage programmation, transition Scratch-Python, milieu didactique, jeu sérieux, EIAH, learning analytics.

1 Introduction

Au fil des ans, en France comme à l'international, la programmation par blocs semble être devenue une des modalités préférentielles d'introduction de la programmation informatique auprès des plus jeunes. La recherche a démontré les bénéfices de cette approche comparativement à l'introduction classique à l'aide de langages textuels (Armoni et al., 2015 ; Price et Barnes, 2015 ; Weintrop et Wilensky, 2017). Dans le même temps, la programmation basée sur du texte reste très majoritairement utilisée au lycée par les élèves plus âgés et par les étudiants à l'université. La plupart des apprenants ayant débuté la programmation sous la forme de blocs devront donc changer de modalité en s'initiant à la programmation textuelle. Comment les aider dans cette transition ?

C'est une des questions vives qui occupent le champ de recherche qui porte sur l'introduction à la programmation (Weintrop, 2019).

Afin d'accompagner les élèves dans ce changement, nous avons développé un jeu sérieux (Alvarez, 2007) visant l'introduction du langage de programmation Python à des élèves de seconde. Ainsi, l'application en ligne *Pyrates* [1,2] prend la forme d'un jeu de plateforme permettant le contrôle d'un avatar en Python. Nous présentons dans cette contribution la conception du jeu et en particulier l'agencement du milieu didactique (Brousseau, 1998). Nous évaluons ensuite nos choix de conception en analysant l'appropriation du jeu par les élèves dans les classes.

Nous exposons d'abord notre cadre théorique en synthétisant quelques concepts de la Théorie des situations didactiques de Brousseau (1998). Ensuite, nous présentons notre revue de travaux en lien avec la transition bloc-texte, avant de détailler notre méthodologie et les résultats en découlant. Finalement, nous concluons et formulons les perspectives et les prolongements de ce travail.

2 Cadre théorique et problématique

Nous nous plaçons dans le cadre de la Théorie des situations didactiques élaborée par Brousseau (1998). Nous exposons ci-dessous les concepts que nous allons mobiliser.

Brousseau définit une **situation** comme : « *une situation problème qui nécessite une adaptation, une réponse de l'élève.* » (1981, p. 112). Il établit le **milieu didactique** comme étant « c*onstitué des objets (physique, culturels, sociaux, humains) avec lesquels le sujet interagit dans une situation [...]. C'est le système antagoniste de l'actant [...] tout ce qui agit sur l'élève et ce sur quoi l'élève agit.* » (2010, p. 2). Cet auteur considère que les activités proposées aux élèves doivent tendre vers « *une sorte d'idéal vers lequel il s'agit de converger* » (1986, p. 50) : la **situation adidactique**. Ce type de problème doit permettre à l'élève d'agir de son propre mouvement, guidé uniquement par la logique interne de la situation. Autrement dit, sans s'appuyer sur les intentions didactiques de l'enseignant qui se refuse à intervenir comme proposeur de la connaissance qu'il cible. Le milieu didactique doit être en mesure de fournir des **rétroactions** à l'élève en réponse à ses actes. Le milieu doit ainsi pouvoir lui dispenser des sanctions positives ou négatives lui permettant d'ajuster son action. (Bessot, 2003).

Chacun des huit niveaux du jeu *Pyrates* est conçu sur le modèle de la situation adidactique : une situation ludique qui doit amener les élèves à écrire un programme mettant en œuvre certaines notions algorithmiques. Nous ne développons pas cet aspect de la conception dans cette communication bien qu'il soit nécessaire de le garder en tête pour comprendre la suite. Nous nous concentrons ici sur l'agencement du milieu didactique des situations. Ainsi, les questions de recherche adressées par ce travail sont :

- QR1 : quels sont les avantages des environnements de programmation basés sur les blocs en comparaison de ceux basés sur du texte ?
- QR2 : comment concevoir un environnement d'apprentissage de Python qui comporte certaines des caractéristiques avantageuses des éditeurs basés sur les blocs ?
- QR3 : lors des expérimentations sur le terrain, les élèves se saisissent-ils de ces caractéristiques ? Si oui, quel usage en font-ils ?

3 État de l'art

Notre revue de travaux est constituée de deux parties. Nous présentons d'abord des applications existantes conçues dans le but d'accompagner la transition des blocs vers le texte. Nous exposons ensuite les résultats de travaux analysant les différences intrinsèques entre ces deux types d'environnements.

3.1 Applications existantes

Afin de soutenir la transition des blocs vers le texte, plusieurs pistes s'appuyant sur des environnements numériques ont été explorées. Parmi ces dispositifs, nous distinguons trois types d'environnements : composés unidirectionnels, composés bidirectionnels et hybrides.

Les environnements composés unidirectionnels comprennent deux vues. L'une permet l'édition des programmes à l'aide de blocs, ces programmes étant convertis automatiquement dans un langage textuel cible dans l'autre vue. Ce langage cible n'est pas directement modifiable, il peut uniquement être consulté et éventuellement exécuté par les utilisateurs. C'est par exemple le cas de l'application *Block2Py* (Declercq et Nény, 2020) dont les blocs miment la syntaxe de Python. L'environnement *Patch* (Robinson, 2016) présente un fonctionnement similaire se basant sur Scratch.

Les environnements composés bidirectionnels sont structurés de la même manière que ceux qualifiés d'unidirectionnels. Ce à quoi s'ajoute la possibilité de créer ou de modifier les programmes directement dans la vue textuelle. Cela entraîne automatiquement la traduction ou la mise à jour du programme dans la vue blocs. Parmi les implémentations existantes, nous pouvons citer *PencilCode* (Bau et al., 2015) qui vise l'apprentissage de Javascript et plus récemment de Python (Andrews et al., 2021), et *BlocEditor* (Matsuzawa et al., 2015) qui propose une approche similaire pour Java. *Blockpy* (Bart et al., 2017) permet également de programmer en Python.

Enfin, les environnements hybrides sont basés sur des cadres (« *framed-based* ») qui combinent blocs et textes en une seule vue. Les structures de haut niveau (boucles, conditionnelles, etc.) peuvent être insérées par glisser-déposer ou au clavier en utilisant des raccourcis. Le code de niveau expression est introduit par édition de texte traditionnelle soutenue par l'auto-complétion. *GP* est une mise en œuvre issue d'une étude exploratoire (Monig et al., 2015). *Stride* met à disposition des enseignants une implémentation opérationnelle pour le langage Java (Kölling et al., 2015, 2017). Depuis peu, *Strype* (Kyfonidis et al., 2021) offre un environnement hybride dédié à l'édition Python.

Certains de ces environnements ont été évalués par des études empiriques. Même s'ils semblent prometteurs (Alrubaye et al., 2019 ; Blanchard et al., 2020), les effets positifs sur les apprentissages ne sont pas toujours démontrés (Brown et al., 2021). Ces logiciels intègrent une traduction automatique vers du texte depuis un environnement d'édition par blocs, ou proposent un environnement mixte blocs-texte. Nous proposons d'explorer une troisième voie en concevant un environnement d'édition purement textuel intégrant certaines caractéristiques avantageuses des éditeurs de blocs. Cette modalité, complémentaire des deux autres, pourra trouver sa place dans une étape plus avancée de la transition blocs-texte.

3.2 Avantages des environnements basés sur les blocs

Plusieurs auteurs (Bau et al., 2017 ; Kolling et al. 2015 ; Weintrop, 2019) ont analysé les différences intrinsèques et les avantages des environnements de programmation par blocs comparés à ceux basés sur du texte. Nous résumons ci-dessous les résultats qui en ressortent. Cela permet de répondre à notre question de recherche QR1.

- **R1.1 : présence d'un catalogue de commande.** Les environnements de programmation par blocs présentent à l'utilisateur un panneau recensant tous les blocs existants organisés de manière thématique et conceptuelle. Les utilisateurs novices peuvent ainsi découvrir de nouveaux concepts ou se remettre en mémoire ceux précédemment acquis. Dans les environnements textuels, l'existence et la syntaxe des structures de code doivent être connues par avance des programmeurs.
- **R1.2 : nombre réduit d'éléments significatifs**. Les langages de programmation textuels sont constitués de nombreux éléments significatifs (mots-clés, signes typographiques, etc.). Cette notation dense constitue un obstacle pour les novices car elle peut surcharger leur mémoire de travail. Les programmeurs expérimentés ont appris au fil du temps à interpréter le code en plus gros morceaux (« *chunks* »). Les blocs permettent de réduire la charge cognitive des programmeurs débutants en leur permettant d'appréhender les commandes en morceaux plus importants.
- **R1.3 : aide à la structuration des programmes**. Les programmes basés sur du texte sont structurés à l'aide de parenthèses, d'accolades ou de l'indentation des lignes. La mécanique d'agencement et de maintien de cette structure est un défi pour les débutants et entraine très souvent des erreurs syntaxiques ou sémantiques. Ces contraintes sont moins présentes dans les langages de blocs dans la mesure ou la structuration des programmes est guidée et maintenue par la forme des blocs.
- **R1.4 : peu de saisies au clavier**. La composition de programmes par glisser-déposer des blocs limite la difficulté de la saisie et de la recherche des signes typographiques sur le clavier. L'acte purement mécanique de taper le texte du programme peut constituer un obstacle cognitif et moteur pour les jeunes apprenants. La nécessité de le faire ajoute une charge supplémentaire et des distractions cognitives lorsqu'il faut corriger les inévitables erreurs de frappe.
- **R1.5 : absence d'erreurs syntaxiques.** Les systèmes basés sur des blocs permettent d'éviter la plupart des erreurs de syntaxe à la faveur d'une manipulation globale et contrainte des structures. Dans les systèmes textuels, ces erreurs sont nombreuses et les messages d'erreur sont généralement vagues dans leur formulation. L'interprétation de ces messages est une compétence non triviale que les novices mettent beaucoup de temps à maîtriser.
- **R1.6 : contrôle et visibilité de l'exécution**. Les environnements basés sur les blocs facilitent le contrôle et améliorent la visibilité de l'exécution des programmes. Ils permettent de mettre en évidence le bloc en cours d'exécution afin de rendre visible la correspondance entre le code et l'action, d'offrir un mode pas-à-pas (régler la vitesse, arrêter et reprendre l'exécution) ou de rendre visible l'état courant des variables. Ces fonctionnalités, que l'on ne rencontre pas nécessairement dans les environnements basés sur du texte, permettent aux débutants de mieux comprendre l'exécution des programmes.

Ces comparaisons s'appuient sur des éditeurs de code basiques. Notons que certains environnements de développement pédagogiques utilisés en seconde, tel que *EduPython* [3], proposent des fonctionnalités facilitantes comme la coloration syntaxique, la complétion automatique ou la vérification de la syntaxe pendant la saisie qui peuvent aider à la structuration des programmes (R1.3) et à limiter la saisie au clavier (R1.4).

4 Méthodologie

Dans cette section, nous décrivons la méthodologie nous permettant de répondre aux questions de recherche relatives à l'agencement du milieu didactique (QR2) et à l'évaluation de nos choix de conception (QR3).

4.1 Un milieu didactique inspiré des environnements basés sur les blocs

Nous avons aménagé le milieu didactique des situations du jeu (QR2) en nous appuyant sur les résultats de la question de recherche QR1. Ainsi, nous avons incorporé dans le milieu les caractéristiques des environnements de programmation par blocs en espérant pouvoir profiter de leurs avantages.

4.2 Évaluation de la conception : traces d'activités et questionnaire

La méthodologie liée à l'évaluation de notre conception (QR3) se base d'abord sur l'analyse des traces de l'activité des utilisateurs. Ces traces sont générées automatiquement au format standardisé *xAPI* (Kevan & Ryan, 2016). Elles témoignent des interactions des élèves avec le milieu didactique : consultations des contenus, copiés-collés, erreurs dans les programmes, aides apportées par l'enseignant, manipulation du panneau de contrôle, etc. Ces traces sont complétées par une enquête en ligne renseignée par les élèves en fin d'expérimentation. Ce questionnaire a pour but de recueillir le point de vue qualitatif des élèves sur l'application.

Nous avons pu expérimenter notre application dans huit classes de seconde auprès de 240 élèves débutants en Python sur deux ou trois séances de 55 minutes chacune. Après, une rapide présentation, il était attendu des élèves qu'ils utilisent le jeu de manière autonome. L'enseignant avait pour consigne de n'intervenir qu'à leur demande. Afin de garder trace de ces interactions, l'enseignant devait renseigner dans l'interface le contenu de l'aide apportée en cliquant sur des boutons qui lui sont réservés (voir Fig. 1-f). Notons que nous avons, lors de toutes les séances, prêté main forte à l'enseignant dans cette tâche.

Notre corpus de données est ainsi constitué de 69 701 traces d'activités et de 224 réponses au questionnaire en ligne (certains étudiants n'ont pas pu répondre pour des raisons techniques). Il a été analysé de façon automatisée au moyen de programmes Python. La manipulation et le traitement des données exploitent la bibliothèque *Pandas*, les graphiques sont générés par les librairies *Matplotlib* et *Seaborn*.

5 Résultats

5.1 Aménagement du milieu didactique

Nous relatons maintenant la manière dont nous avons aménagé le milieu didactique. Notre exposé s'appuie sur la figure Fig. 1 qui reproduit l'interface graphique et les différentes zones de l'application *Pyrates.*

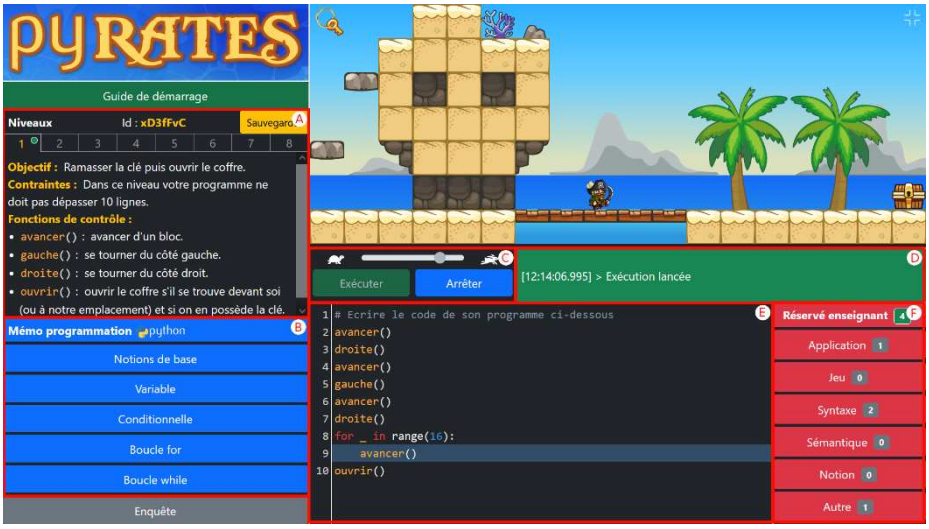

Fig. 1. Différentes zones de l'interface graphique de l'application *Pyrates*.

Nous avons d'abord créé sur la partie gauche de l'interface un bandeau fixe comportant un « Mémo programmation » (Fig. 1-b). Cette zone s'inspire du catalogue de commande présent dans les environnements basés sur les blocs (R1.1). Les contenus y sont classés par notions (notions de base, variable, conditionnelle, boucle for, boucle while) et sont accessibles en cliquant sur les différents boutons bleus. Les concepts présentés ont été choisis en cohérence avec les programmes scolaires du cycle 4 et de la seconde. Notre dispositif visant la transition collège-lycée, nous ne présentons pas la notion de fonction informatique qui est un objectif pour l'année de seconde. De plus, les notions liées aux expressions et aux calculs ne sont pas réinvesties car le contexte du jeu ne s'y prête peu. Afin de guider au mieux les élèves dans l'exploration du milieu didactique, le fait de survoler un bouton à la souris change son intitulé en donnant un aperçu de l'utilité de la notion. Par exemple « variable » devient « garder des informations en mémoire ». Le clic sur un bouton entraine l'apparition d'un panneau latéral qui détaille la notion en sous-notions (voir Fig. 2). Ce découpage a été guidé par la constitution des blocs Scratch. Ainsi, la distinction « Répétition simple » et « Répétition avec compteur » (Fig. 2-a) n'a, par exemple, pas de sens du point de vue de la syntaxe Python. Cependant, les compteurs de boucle n'existant pas en Scratch, il nous semble utile de faire cette différence dans un contexte de transition.

(a) Extrait du panneau relatif à la notion de "Boucle for"

(b) Extrait du panneau relatif à la notion de "Conditionnelle"

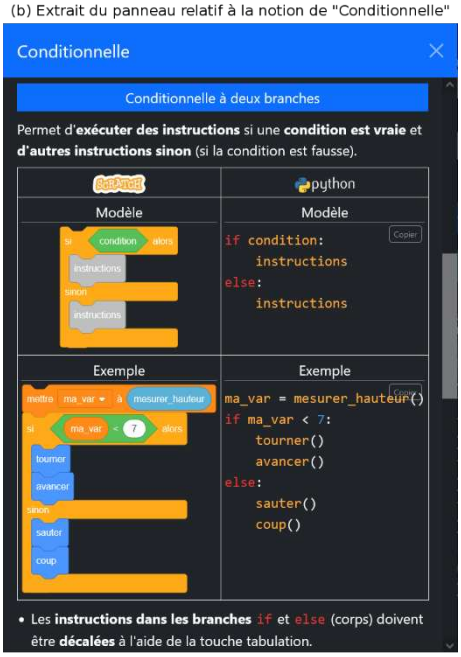

Fig. 2. Exemples d'extraits des panneaux latéraux

Chaque sous-notion est présentée et accompagnée de deux programmes Python : un modèle générique et un exemple dans le contexte du jeu. Nous fournissons de surcroit la traduction de ces éléments dans le langage Scratch. La présence du modèle générique et de l'équivalent Scratch des programmes a pour but d'aider les apprenants à réduire le nombre d'éléments significatifs. Il s'agit d'appréhender le programme Python par morceaux (« *chunk* ») et non élément par élément (R1.2). Par exemple, dans le cas de la répétition simple (Fig. 2-a), les élèves doivent se focaliser sur le nombre entre parenthèses et considérer le reste du code comme un même agrégat. Afin de limiter la saisie au clavier (R1.4), chaque pièce de code Python est accompagnée d'un bouton « Copier » . L'objectif est ici d'encourager la pratique du copier-coller vers l'éditeur de texte. Cet usage doit, d'une certaine manière, prendre le relais du glisser-déposer caractéristique des environnements basés sur les blocs. Cela peut aussi, dans une moindre mesure, soutenir la structuration des programmes (R1.3). L'utilisateur devra cependant veiller au maintien de cette structure au cours de la rédaction de son programme.

En dépit des efforts de conception qui viennent d'être décrits, il semble présomptueux d'envisager la disparition des erreurs syntaxiques. L'interprétation des messages d'erreur constituant un frein pour les programmeurs novices (R1.5), nous avons décidé d'enrichir le milieu didactique d'un analyseur syntaxique conçu pour les débutants (Kohn, 2017). Ce module analyse le code Python saisi par les utilisateurs avant qu'il ne soit exécuté par l'interpréteur. Il a la particularité de formuler des messages d'erreur en français, mais surtout dans un registre pratique et compréhensible des novices. Nous avons également effectué un travail de reformulation des messages afin d'adapter leur

terminologie à celle du mémo programmation. Ainsi, lors de la survenue d'une erreur syntaxique, le message s'affiche dans la zone console de l'interface (Fig. 1-d) et la ligne en cause est surlignée en rouge dans la zone d'édition du code (Fig. 1-e). L'absence d'erreurs détectées par l'analyseur syntaxique ne signifie pas que le code est interprétable. Des erreurs sémantiques (liées par exemple au typage) peuvent toujours apparaitre lors de l'interprétation.

Enfin, nous avons créé un panneau de contrôle (Fig. 1-c) dans le but d'améliorer la maîtrise de l'exécution des programmes (R1.6). Outre le fait de pouvoir lancer l'exécution de leur programme, les utilisateurs ont la capacité d'arrêter l'exécution en cours et de régler sa vitesse à l'aide d'un curseur. Ce curseur modifie la vitesse d'action des personnages en agissant sur un coefficient multiplicateur positionné à 1 (tortue) au lancement du jeu et pouvant aller jusqu'à 3 (lièvre). La visualisation et le suivi de l'exécution (R1.6) sont assurés par la mise en valeur (surlignage) de la ligne en cours d'exécution dans la zone d'édition du code (Fig. 1-e). Ainsi, la correspondance entre le code et l'action en cours est apparente.

5.2 Évaluation des choix de conception

Afin d'évaluer les choix de conception que nous venons d'exposer, nous allons analyser les traces d'utilisations générées automatiquement par l'application. Dans le cadre de cette étude, nous prenons en compte les traces concernant : la consultation du mémo, les copiés-collés du mémo vers l'éditeur de code, les erreurs détectées par l'analyseur syntaxique et par l'interpréteur, les aides syntaxiques et sémantiques renseignées par les enseignants lors de leurs interventions, la manipulation du curseur de vitesse et la vitesse choisie lors de l'exécution des programmes.

Commençons par examiner l'utilisation du mémo programmation. Comme nous pouvons le voir dans la figure Fig. 3-a, ce mémo est fréquemment consulté par les élèves. Nous remarquons également que, à l'instar du catalogue des environnements basés sur les blocs, il est le support de la découverte des notions. Ainsi, chaque fois qu'une nouvelle notion est mise en jeu dans un niveau (niv.1, niv.3, niv4. et niv.8), nous retrouvons une certaine variété dans les notions consultées. C'est en effet la manifestation d'une démarche de recherche. Lorsque les notions ont déjà été utilisées (niv.2, niv.5 et niv.6), la consultation semble plus ciblée sur les notions en jeu. Nous pouvons faire l'hypothèse qu'il s'agit, dans ce cas, pour les élèves de se rappeler de la syntaxe d'implémentation des notions. Nous retrouvons ici la fonction de remémoration du catalogue des environnements basés sur les blocs. En analysant la figure Fig. 3-b, nous pouvons affirmer que les élèves s'emparent quasi-systématiquement du copier-coller lors de l'implémentation d'une notion. En effet, chaque fois qu'une notion est mise en jeu dans un niveau, nous retrouvons, en moyenne, au moins un usage du copier-coller qui lui est associé. Exception faite de la notion de variable qui dispose d'une syntaxe d'implémentation beaucoup plus simple que les autres notions. Cette pratique se rapprochant du glisser-déposer des blocs, elle est en mesure de limiter la saisie au clavier et d'aider l'établissement des structures de code.

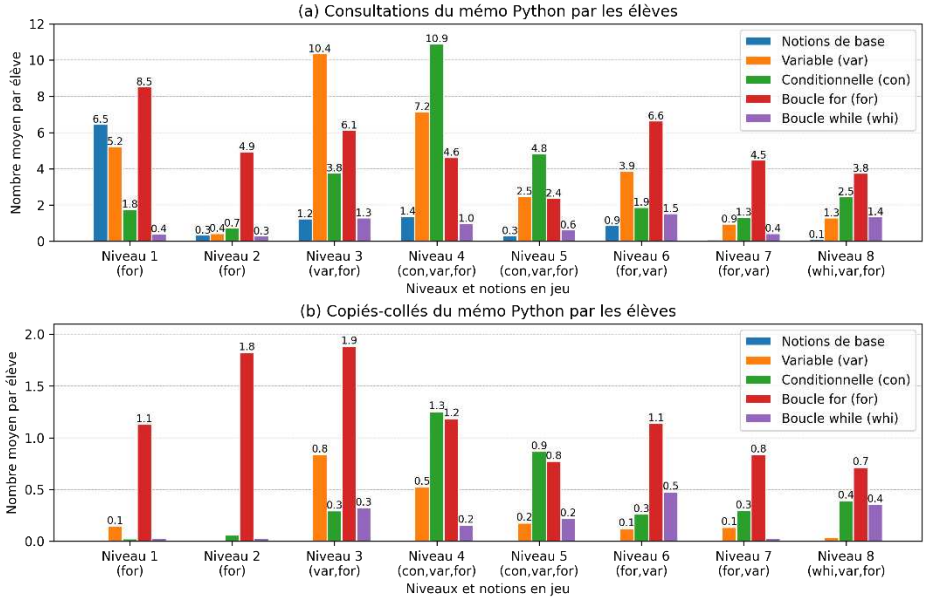

Fig. 3. Consultations et copiés-collés du mémo Python par les élèves par niveau.

Exposons désormais nos analyses se rapportant aux erreurs syntaxiques (issues de l'analyseur syntaxique). L'examen de la figure Fig. 4-a permet d'affirmer qu'elles sont présentes en nombre et dans une proportion bien supérieure à celles portant sur la sémantique (issues de l'interpréteur). En s'intéressant aux aides apportées par les enseignants (Fig. 4-b), il est remarquable de constater que les interventions en rapport avec la syntaxe des programmes sont très peu fréquentes. Cette rareté s'apprécie en comparaison du nombre d'erreurs. Ainsi, on compte une intervention pour trente à quarante erreurs syntaxiques dans les quatre premiers niveaux. Les élèves donc sont en mesure d'ajuster leurs procédures grâce aux rétroactions du milieu, sans solliciter l'enseignant.

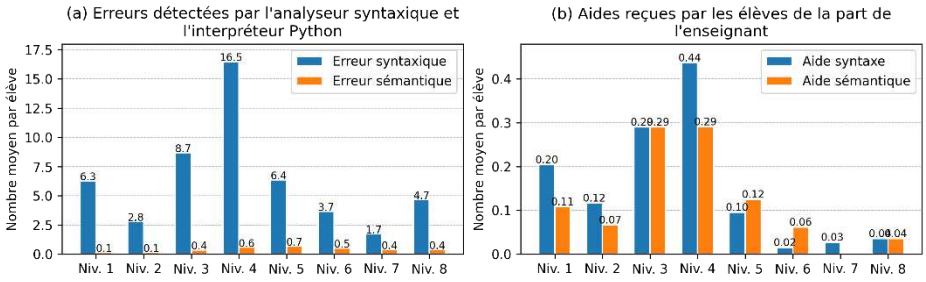

Fig. 4. Erreurs détectées par l'application et aides de l'enseignant reçues par les élèves.

Les traces générées par l'application nous donnent des indications quantitatives relatives à l'utilisation du mémo et à l'interprétation des messages d'erreur. Ces analyses peuvent néanmoins être complétées qualitativement par le questionnaire que nous

avons fait passer aux élèves en fin d'expérimentation. Cette enquête comportait des questions en lien avec le mémo Python et les messages d'erreur. Il s'agissait pour les élèves d'évaluer plusieurs aspects de l'application en plaçant des curseurs entre deux extrêmes (« Pas clair » – « Très clair », « Pas utile » – « Très utile »), ce qui a pour effet de générer un score entre 0 et 100. Nous présentons dans la figure Fig. 5 la répartition (densité) et la médiane des scores concernant les questions qui nous intéressent.

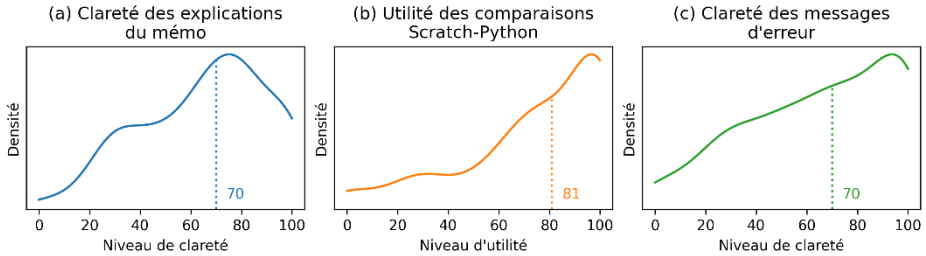

Fig. 5. Extrait des résultats de l'enquête élèves (répartition des réponses et médiane).

En plus d'être beaucoup consultées par les élèves, les explications contenues dans le mémo sont jugées comme étant claires par la majorité des élèves (Fig. 5-a). Nous distinguons néanmoins un groupe d'élèves autour du score de 30 pour qui ces contenus sont plus obscurs. Les comparaisons avec Scratch (Fig. 5-b) sont estimées comme étant utiles, voire très utiles par la grande majorité des élèves. Enfin, les messages d'erreur, dont nous avons montré l'utilité dans l'autonomie des élèves, sont également estimés comme étant clairs par la majorité des répondants.

Évaluons désormais l'utilisation des fonctionnalités de contrôle des programmes. En nous penchant sur la figure Fig. 6-a, nous constatons un très grand nombre de programmes lancés en moyenne par élève. Parmi ces programmes, beaucoup sont erronés (erreur syntaxique ou sémantique, trop de lignes), ce qui semble indiquer l'adoption par les élèves d'une démarche de programmation par essais-erreurs relativement à la correction des programmes. De nombreux programmes corrects sont également lancés (erreur liée au jeu, perte d'un niveau, progression dans un niveau, gain d'un niveau), cela montre que les élèves progressent dans les niveaux de façon incrémentale, par étapes intermédiaires. Les arrêts de programmes sont peu fréquents. Il est cependant possible de distinguer deux types de comportements en fonction du mode de génération du parcours des niveaux. Pour un premier ensemble de niveaux ayant des parcours fixes non aléatoires (niv.1, niv.2, niv.6 et niv.7), les élèves utilisent en moyenne entre quinze et vingt lancements et pratiquement aucun arrêt. Dans les niveaux ayant un parcours aléatoire changeant à chaque exécution (niv.3, niv.4, niv.5 et niv.8), les élèves ont tendance à avoir recours à davantage d'exécutions et à en stopper un certain nombre. Il est probable qu'une partie d'entre eux adoptent pour ces niveaux, de façon transitoire, un mode opératoire consistant à enchaîner les « lancer-arrêter » jusqu'à obtenir une configuration du parcours favorable à leur programme. Cette stratégie que nous qualifions de « contournement didactique » permet de réussir ces niveaux sans mettre œuvre les structures basées sur des tests (conditionnelle, while). Cette procédure a très peu de chances d'aboutir en raison du nombre important de configurations aléatoires

différentes. Ces élèves qui restent « à tout prix » dans le domaine ludique ne souhaitent ou ne peuvent pas rentrer dans l'apprentissage notionnel en explorant le milieu à la recherche d'une notion qui pourrait leur permettre de terminer le niveau.

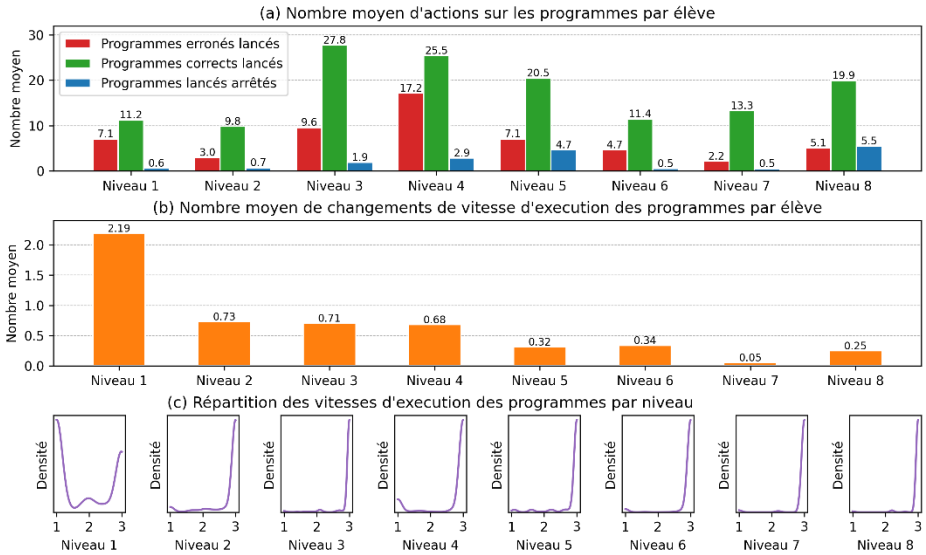

Fig. 6. Données concernant le contrôle de l'exécution des programmes par niveau.

Pour finir, portons notre attention sur le curseur de changement de vitesse. Il est en moyenne peu utilisé (Fig. 6-b) et de manière décroissante dans le temps. La figure Fig. 6-c présente la répartition (densité) des vitesses d'exécution des programmes lancés pour chaque niveau. Elle permet de constater que, dès le niveau 2, les programmes sont presque tous lancés à la vitesse maximale (coefficient multiplicateur 3). La démarche de programmation par essais-erreurs et par étapes incrémentales que nous avons décrit précédemment va de pair avec cette vitesse d'exécution élevée. Trois élèves font d'ailleurs remarquer dans le champ libre du questionnaire que « *le bonhomme n'avance pas assez vite* ». Nous constatons néanmoins une pratique utilisée à la marge dans des niveaux plus avancés (niv.4 et niv.5) qui consiste à revenir à des vitesses d'exécution plus lentes. Nos observations durant les expérimentations indiquent qu'il s'agissait pour certains élèves de pouvoir, ponctuellement, suivre plus facilement les lignes exécutées dans un mode d'action se rapprochant du pas-à-pas.

6 Conclusion et perspectives

En conclusion de cette contribution, nous allons d'abord en rappeler les principaux résultats en revenant à nos questions de recherche. Notre revue de travaux nous a permis d'établir les différences et les avantages des environnements de programmation basés sur les blocs en comparaison de ceux basés sur du texte (QR1). Nous avons ensuite aménagé le milieu didactique des situations de l'application *Pyrates* en y intégrant

certains éléments, supposés profitables pour les élèves, des environnements basés sur les blocs (QR2). Enfin, nous avons évalué notre conception en analysant les traces d'utilisations de l'application et les réponses au questionnaire (QR3). Certains choix d'agencement ont des conséquences positives :

- le mémo programmation est très fréquemment consulté par les élèves, il est le support de la découverte et de la remémoration des notions ;
- les comparaisons avec Scratch qu'il comporte sont considérées comme utiles par une très large majorité d'élèves, elle doivent aider l'appréhension par morceaux des structures Python ;
- le copier-coller depuis le mémo programmation est largement pratiqué, cela a pour effet de limiter la saisie au clavier et, dans une moindre mesure, de soutenir la tâche de structuration des programmes ;
- les rétroactions fournies par l'analyseur syntaxique sous la forme de messages d'erreur jugés « clairs » par les élèves rend possible la correction des programmes en sollicitant très peu l'enseignant.

Le panneau de contrôle devait permettre aux élèves de mieux comprendre l'exécution des programmes. Nous constatons, de façon très marginale, un réduction de la vitesse du personnage afin de suivre les exécutions sur le mode du pas-à-pas. Cependant, de manière générale, il ne produit pas les résultats escomptés :

- le bouton de lancement des programmes est fréquemment utilisé et le curseur de réglage de la vitesse d'exécution est très rapidement positionné au maximum dans le but d'adopter une démarche de programmation par essais-erreurs peu propice à la réflexion ;
- le bouton permettant de stopper les exécutions est peu utilisé, quand il l'est, c'est surtout pour tenter de réussir des niveaux basés sur des parcours aléatoires par « contournement didactique ».

Ces résultats doivent être considérés au regard des limites de notre méthodologie. Ainsi, les élèves étant en contexte écologique, il fut difficile de maintenir des conditions expérimentales totalement similaires entre nos différents groupes, notamment concernant l'activité de l'enseignant et la distance temporelle entre les séances. D'autre part, le raisonnement sur des moyennes permet de dégager des tendances, mais masque les disparités de pratique entre les élèves dont nous avons été témoin dans les classes.

Évoquons pour finir quelques perspectives permettant de prolonger ce travail. Edwards (2004), affirme que les débutants en informatique réussissent mieux à apprendre s'ils passent d'une approche par essais-erreurs à une pratique de « *réflexion en action* ». Il serait ainsi avantageux de modifier les possibilités de contrôle de l'exécution dans notre application de manière à contraindre les élèves à moins d'action et à plus de réflexion. Un moyen pourrait être de limiter le nombre d'exécutions à l'aide de pénalités. Par ailleurs, il serait intéressant d'exploiter nos traces d'activités à l'aide d'algorithme de Data Mining afin de mettre en évidence les différentes stratégies de résolution mises en œuvre. Des algorithmes de clustering pourraient également permettre de faire émerger différents profils d'élèves.

Remerciements

Ce travail a bénéficié du soutien financier de la Région Bretagne et du projet ANR IE-CARE.

Références

1. Alrubaye, H., Ludi, S., & Mkaouer, M. W. (2019). Comparison of block-based and hybrid-based environments in transferring programming skills to text-based environments. In T. Pakfetrat, G.-V. Jourdan, K. Kontogiannis R. Enenkel (dir.), *Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering* (p.100-109). IBM Corp.
2. Alvarez, J. (2007). *Du jeu vidéo au serious game: approches culturelle, pragmatique et formelle* [Thèse de doctorat]. Université de Toulouse 2.
3. Andrews, E., Bau, D., & Blanchard, J. (2021). From Droplet to Lilypad: Present and Future of Dual-Modality Environments. In *2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (p. 1-2). IEEE.
4. Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). From scratch to "real" programming. *ACM Transactions on Computing Education (TOCE)*, 14(4), 1-15.
5. Bart, A. C., Tibau, J., Tilevich, E., Shaffer, C. A., & Kafura, D. (2017). Blockpy: An open access data-science environment for introductory programmers. *Computer*, 50(5), 18-26.
6. Bau, D., Bau, D. A., Dawson, M., & Pickens, C. S. (2015). Pencil code: block code for a text world. In *Proceedings of the 14th International Conference on Interaction Design and Children* (p. 445-448).
7. Bau, D., Gray, J., Kelleher, C., Sheldon, J., & Turbak, F. (2017). Learnable programming: blocks and beyond. *Communications of the ACM*, 60(6), 72-80.
8. Bessot, A. (2003). Une introduction à la théorie des situation didactiques. *Les cahiers du laboratoire Leibniz*, 91, 1-28.
9. Blanchard, J., Gardner-McCune, C., & Anthony, L. (2020, February). Dual-Modality Instruction and Learning: A Case Study in CS1. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (p. 818-824).
10. Brousseau, G. (1981). Problèmes de didactiques des décimaux. *Recherches en Didactique des Mathématiques*, 2(1), 37-125.
11. Brousseau, G. (1986). Fondements et méthodes de la didactique des mathématiques. *Recherches En Didactique Des Mathématiques*, 7(2), 33–115.
12. Brousseau, G. (1998). *Théorie des situations didactiques : Didactique des mathématiques 1970-1990.* La Pensée Sauvage.
13. Brousseau, G. (2010). *Glossaire de quelques concepts de la théorie des situations didactiques en mathématiques.* Site personnel. http://guy-brousseau.com/wp-content/uploads/2010/09/Glossaire_V5.pdf
14. Brown, N., Kyfonidis, C., Weill-Tessier, P., Becker, B., Dillane, J., & Kölling, M. (2021). A Frame of Mind: Frame-based vs. Text-based Editing. In *Proceedings of the 2021 Conference on United Kingdom and Ireland Computing Education Research* (p. 1-7). Association for Computing Machinery.
15. Declercq, C., & Nény, F. (2020). *Block2Py, un éditeur de blocs pour l'apprentissage du langage Python.* Atelier présenté à Didapro 8 – DidaSTIC , Lille, France

16. Edwards, S. H. (2004). Using software testing to move students from trial-and-error to reflection-in-action. In *Proceedings of the 35th SIGCSE technical symposium on Computer science education* (p. 26-30).
17. Kyfonidis, C., Weill-Tessier, P., & Brown, N. (2021). Strype: Frame-Based Editing tool for programming the micro:bit through Python. In *The 16th Workshop in Primary and Secondary Computing Education*. (p. 1–2).
18. Kevan, J. M., & Ryan, P. R. (2016). Experience API: Flexible, decentralized and activity-centric data collection. *Technology, knowledge and learning*, 21(1), 143-149.
19. Kohn, T. (2017). *Teaching Python programming to novices : Addressing misconceptions and creating a development environment* [Thèse de doctorat]. ETH Zurich.
20. Kölling, M., Brown, N. C., & Altadmri, A. (2015). Frame-based editing: Easing the transition from blocks to text-based programming. In *Proceedings of the Workshop in Primary and Secondary Computing Education* (p. 29-38).
21. Kölling, M., Brown, N. C., & Altadmri, A. (2017). Frame-Based Editing. *Journal of Visual Languages and Sentient Systems*, 3, 40-67.
22. Price, T. W., & Barnes, T. (2015). Comparing textual and block interfaces in a novice programming environment. In *Proceedings of the eleventh annual international conference on International Computing Education Research* (p. 91–99). Association for Computing Machinery.
23. Matsuzawa, Y., Ohata, T., Sugiura, M., & Sakai, S. (2015). Language migration in non-cs introductory programming through mutual language translation environment. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (p. 185-190)
24. Monig, J., Ohshima, Y., & Maloney, J. (2015). Blocks at your fingertips: Blurring the line between blocks and text in GP. In *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)* (p. 51-53). IEEE.
25. Robinson, W. (2016). From scratch to patch: Easing the blocks-text transition. In *Proceedings of the 11th Workshop in Primary and Secondary Computing Education* (p. 96-99).
26. Weintrop, D., & Wilensky, U. (2017). Comparing block-based and text-based programming in high school computer science classrooms. *ACM Transactions on Computing Education (TOCE)*, 18(1), 1-25.
27. Weintrop, D. (2019). Block-based programming in computer science education. *Communications of the ACM*, 62(8), 22-25.

Pages Internet

1. Page d'accueil de l'application Pyrates, https://py-rates.fr, dernière consultation 15/12/21.
2. Guide pédagogique de l'application Pyrates, https://py-rates.fr/guide/FR/, dernière consultation 15/12/21.
3. Page d'accueil du site EduPython, https://edupython.tuxfamily.org/, dernière consultation 15/12/21.