Tester son programme lors d'activités de conception de jeux vidéo à l'école

Jérémy Le Du[1], Julian Alvarez[2], Daniel Schmitt[1]

[1] Université de Lorraine
Jeremy.Le-Du@hrsk.fi
daniel.schmitt@univ-lorraine.fr
[2] Université de Lille
Julian.alvarez@univ-lille.fr

Résumé. Cette étude explore le rôle du test de programme dans le cadre d'activités de conception de jeux vidéo en milieu scolaire. Elle a été conduite auprès de 373 élèves âgés de 9 à 13 ans, dans le cadre du projet Educational Game & Play Design Project (EGPDP). S'appuyant sur une approche mixte combinant l'analyse de vidéos avec suivi oculaire et des entretiens en rappel stimulé (méthode Remind), ce travail montre qu'une majorité des élèves utilisent des tests pour valider un élément du jeu ou du programme nouvellement ajouté, tandis qu'une minorité les mobilise pour déboguer ou se divertir. Une forte corrélation positive (96 %) est observée entre la fréquence des tests et les scores post-test des élèves en matière de concepts de pensée informatique. Ces résultats semblent conforter la valeur pédagogique des pratiques de test et de débogage systématiques dans les activités de conception de jeux vidéo.

Mots-clés : pensée informatique, conception de jeux vidéo, test de progrmame, technologie éducative, recherche mixte, suivi oculaire, méthode Remind.

Abstract. This study explores the role of program testing within the framework of video game design activities in a school setting. It was conducted with 373 students aged 9 to 13 as part of the Educational Game & Play Design Project (EGPDP). Using a mixed-methods approach that combines video analysis with eye tracking and stimulated recall interviews (Remind method), this study shows that the majority of students use testing primarily to validate newly added game or program elements, while a minority use them for debugging or entertainment. A strong positive correlation (96%) was observed between test frequency and students' post-test scores in computational thinking concepts. These findings seem to support the educational value of systematic testing and debugging practices in video game design activities.

Keywords: Computational Thinking, Video Game Design, Program Testing, Educational Technology, Mixed-Methods Research, Eye Tracking, Remind Method.

1 Introduction

Aujourd'hui encore, la pensée informatique ne fait pas l'objet d'une définition consensuelle. Certains la considèrent comme une compétence du XXI^{e} siècle en tant que telle [1], d'autres comme un processus de réflexion [2], un ensemble de compétences [3] ou un processus de résolution de problèmes [4], multidisciplinaire donc fondamental pour chaque enfant [5].

Un moyen d'opérationnaliser la pensée informatique est proposé par Brennan et Resnick [6], qui distinguent les concepts, les pratiques et les perspectives de la pensée informatique. Cet article se focalise sur l'une de ces pratiques, le test et le débogage, largement liée à d'autres puisqu'elle implique une réflexion logique et analytique pour isoler le problème et se concentrer sur l'erreur, et qu'elle fait partie intégrante des stratégies de développement incrémental et de décomposition des problèmes [7].

Une métasynthèse portant sur 55 études empiriques [8] montre que les tests et le débogage constituent la pratique informatique la plus fréquemment observée dans le cadre d'activités de programmation avec le logiciel Scratch. Les élèves semblent pratiquer les tests et le débogage de manière récurrente [9], à mesure que les programmes gagnent en complexité. Cette pratique se développe à un rythme lent, en particulier la capacité à déboguer un programme, processus complexe qui nécessite de la persévérance et une approche systématique [10], et qui se manifeste selon différentes stratégies [11]. Elle semble favorisée par la collaboration entre élèves [12], et contribue à l'amélioration de la capacité à analyser le code [10] et à comprendre le fonctionnement d'un programme [13].

Cette pratique informatique intègre l'identification de la source du problème, la lecture et la réécriture des scripts, la recherche d'exemples de scripts fonctionnels, toutes associées au débogage ; l'expérimentation des scripts, associée au test [6]. Les études consacrées demeurent rares dans des environnements de programmation par blocs [14; 11], voire inexistantes s'agissant spécifiquement de la pratique de test sur laquelle se focalise la présente étude. Celle-ci examine la capacité d'élèves de 9 à 13 ans à tester le programme informatique conçu. Les questions de recherche sont les suivantes :

- QR1 : À quelle fréquence les élèves testent-ils leur programme lorsqu'ils conçoivent un jeu vidéo ?
- QR2 : Quelles sont les attentes des élèves lorsqu'ils testent le programme créé ?
- QR3 : La pratique de test rend-elle compte de la maîtrise des concepts de la pensée informatique ?

2 Méthode

Une approche mixte, fondée sur le paradigme du pragmatisme [15] et qui combine les approches quantitatives et qualitatives pour se concentrer sur le produit final de la recherche [16], est suivie.

2.1 Contexte et participants

Cette étude est menée dans le cadre du projet Educational Game & Play Design (EGPDP), un programme de conception de jeux vidéo conçu pour être adapté à des élèves du CE2 à la 5e, et proposé à des établissements scolaires francophones de Finlande et de France depuis l'année scolaire 2022-23. Au total, 373 élèves sont inclus dans cette étude.

2.2 Educational Game & Play Design Project

La figure 1 résume les principales caractéristiques du programme EGPDP, définies à partir d'une étude pilote et d'une analyse critique des expériences antérieures de conception de jeux vidéo [17].

Les élèves conçoivent des jeux vidéo éducatifs destinés à leurs pairs avec le logiciel Scratch (6 sur figure 1). Ils programment par pair (4), dispositif qui a montré son efficacité [18]. Quinze séances de 45 minutes sont organisées (2), dont deux dédiées aux évaluations pré-test et post-test (9). Le module est divisé en deux phases différentes (1).

La première est consacrée à l'apprentissage de la programmation et des stratégies de conception de jeux. Elles s'appuient sur 10 modèles de jeu de difficulté variable (8). Les séances sont divisées en temps d'apprentissage de quinze minutes, organisées autour d'une vidéo tutoriel de trois minutes : des vidéos courtes, limitées à un seul objectif, basées sur des exemples concrets, suivies d'une mise en œuvre immédiate [19].

La seconde phase laisse place à la conception libre du jeu vidéo, un temps jugé suffisant pour stimuler la créativité des élèves en s'éloignant des modèles étudiés durant les séances précédentes. À deux reprises, des sessions de tests sont organisées.

Les enseignants disposent d'un ensemble de ressources pédagogiques à leur disposition : des séances de formation en ligne (5, 7) et divers outils pédagogiques. Leur rôle principal est de guider les élèves dans le processus de développement des jeux et dans l'appropriation des ressources.

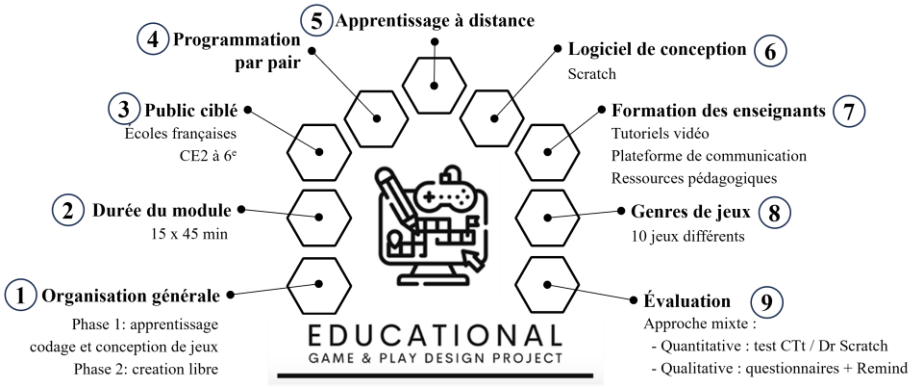

Fig. 1. Protocole expérimental suivi dans le EGPDP [17]

2.3 Mesure

Questionnaire d'auto-évaluation
Le premier outil de mesure repose sur une question posée aux élèves lors du post-test, leur demandant à quelle fréquence ils testent chaque nouvel élément ajouté à leur programme. Une échelle de Likert à 4 points est utilisée : Aucun ; Un peu ; Beaucoup ; Énormément.

Analyse de vidéos des séances
18 groupes d'élèves de niveau hétérogène en programmation ont été observés à l'aide de lunettes caméras avec suivi oculaire lors de la phase de conception. Les vidéos des séances ont ensuite été analysées afin d'identifier et mesurer la durée des phases de test, facilement discernables dans Scratch.

Méthode Remind
En complément, des entretiens en rappel stimulé avec perspective subjective située de 20/25 minutes ont été menés auprès des élèves précédents en suivant le protocole Remind [20]. La combinaison du suivi oculaire avec des entretiens en rappel stimulé caractérisant cette approche permet de comprendre une part significative du raisonnement et des processus de pensée complexes [21], en l'occurrence ici comment l'élève utilise la pratique de test.

Les entretiens ont ensuite été analysés à partir d'un langage de description basé sur l'utilisation de signes hexadiques permettant de raconter la dynamique d'une expérience vécue [22], couplé au modèle #5c21 [23] qui décompose cinq compétences du 21^{e} siècle en composantes mesurables [24]. Dans le cadre de cette étude, seule la composante « s'engager dans le processus d'évaluation et d'amélioration itérative d'un programme informatique » associée à la pensée informatique est considérée.

Mesure des concepts de pensée informatique
L'évaluation des concepts de la pensée informatique s'est faite à l'aide d'une méthode quantitative, sous la forme d'un pré-test et d'un post-test appariés. Le CTt [25] ciblant les élèves du CM2 à la 3e, a démontré à la fois une validité de critère [26], une validité interculturelle [27], tout en disposant de la variante cCTt [28] adaptée aux classes de CE2/CM1. Ces deux tests fournissent une évaluation décontextualisée basée sur respectivement 28 et 25 questions à choix multiples. Dans cette étude, toutes les notes sont ramenées en pourcentage.

3 Résultats

3.1 Fréquence de test et développement des concepts informatiques

La figure 2 représente la distribution des élèves en fonction de leur fréquence de test. Pour mesurer une éventuelle différence dans les bénéfices des activités de conception

de jeux vidéo avec la fréquence de test, les scores moyens pré- et post-test des quatre groupes d'élèves considérés sont représentés. Une corrélation de 96 % est observée entre la fréquence de test et la note moyenne obtenue au post-test, corrélation indépendante de l'âge des élèves.

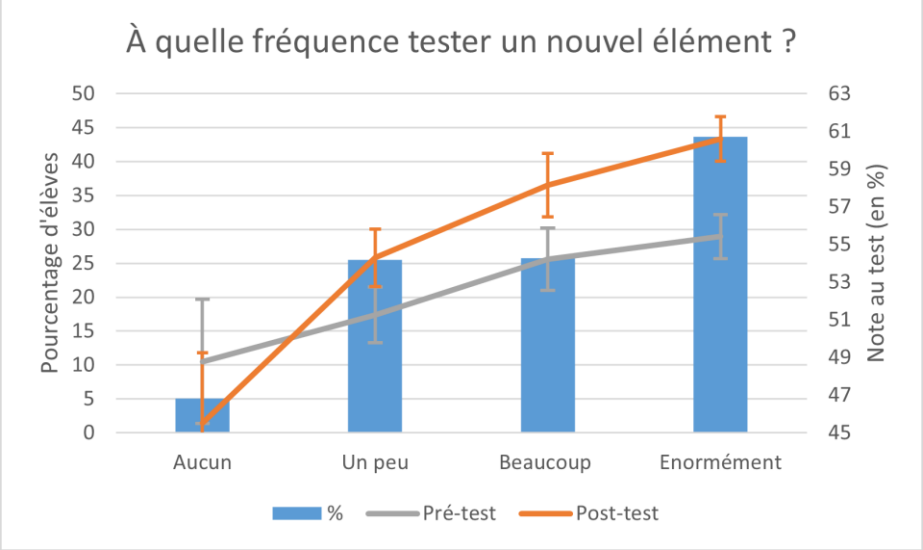

Fig. 2. Distribution des élèves en fonction de la fréquence de test (bleu), et score moyen obtenu au pré-test (gris) et au post-test (orange).

La figure 3 montre la variation de la fréquence de test en fonction de l'âge des élèves.

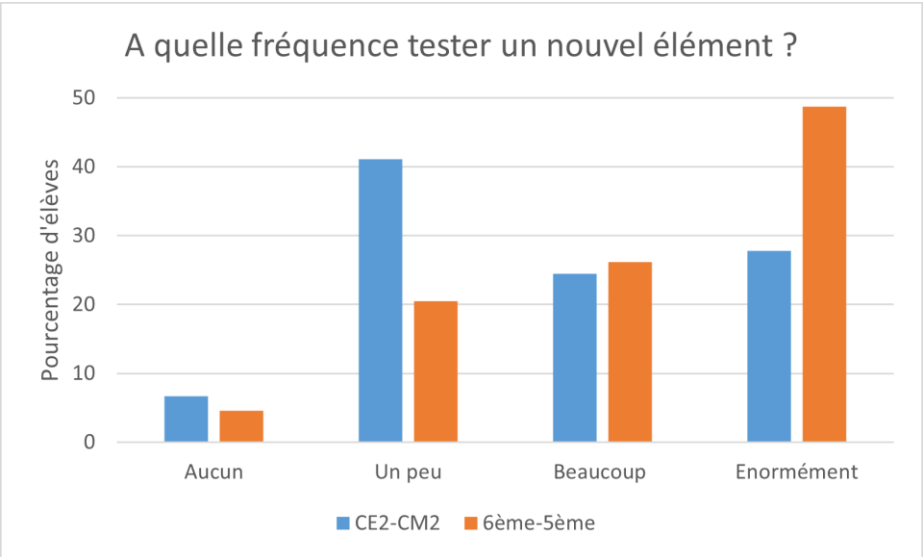

Fig. 3. Distribution des 90 élèves de CE2-CM2 (bleu) et 283 élèves de 6ème-5ème (orange) en fonction de la fréquence de test.

3.2 Analyse des séances

Analyse de vidéos des séances

L'analyse des vidéos en vue subjective des 18 groupes montre qu'en moyenne, les élèves consacrent 12,2 % du temps d'une séance à tester leur programme. Cela représente 11,6 séquences différentes pendant la séance, soit une moyenne de 30,2 secondes consacrée à chaque test, pour un écart type de 15,1s.

Les entretiens Remind des 18 groupes montrent qu'en moyenne, les élèves consacrent 6,9 % du temps de l'entretien à expliciter des actions liées au test de leur programme. Cela représente 5,7 moments différents pendant l'entretien, soit une moyenne de 11,4 secondes consacrée à chacun.

Concernant l'utilisation des tests faite par les élèves, sur les 97 verbalisations des élèves indiquant qu'ils sont en train de tester leur programme, 69 le font pour vérifier si le nouvel élément ajouté fonctionne conformément à leurs attentes, 13 mentionnent explicitement avoir découvert un nouveau problème suite à ce test, 14 testent spécifiquement le jeu en vue de résoudre un problème de codage qu'ils sont en train de traiter, et 1 élève indique tester le jeu simplement pour se divertir.

Fréquence de test et développement des concepts informatiques

La figure 4 représente les notes des élèves obtenues au pré-test et au post-test en fonction du pourcentage de temps consacré à tester le programme, estimé à partir de l'analyse des vidéos.

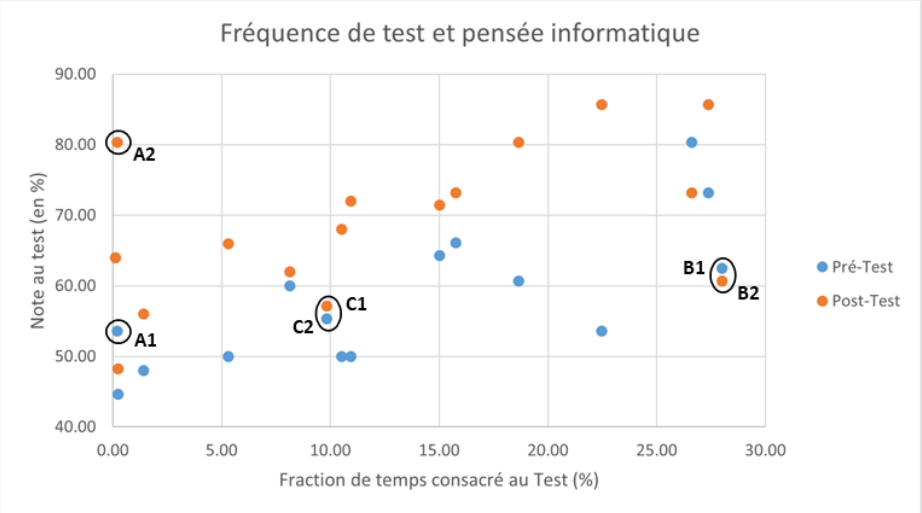

Fig. 4. Notes des élèves obtenues au pré-test et au post-test en fonction du pourcentage de temps de test.

4 Discussion

4.1 Temps de test du programme

La pratique de « tester » le programme conçu est largement mobilisée par les apprentis concepteurs. Lorsqu'ils ajoutent un nouvel élément à leur programme, ils sont une large majorité à le tester systématiquement ou presque, ce qui confirme des études antérieures [8, 9]. Seule une très faible proportion ne teste jamais le jeu conçu.

Ce résultat s'explique par la complexité de la tâche de conception d'un jeu vidéo, qui oblige à décomposer le jeu pensé en différents éléments à programmer, et pousse ainsi à adopter une stratégie de développement incrémental [7]. Il vient aussi confirmer l'efficacité de la phase 1 du programme, qui a su communiquer l'importance de cette pratique auprès d'une large majorité d'élèves novices en programmation. Plusieurs raisons semblent expliquer les 19 élèves n'ayant jamais testé leur programme : manque de motivation, forte confiance en leur capacité à programmer sans aide, forte imitation des jeux modèles.

L'analyse des vidéos en vue subjective confirme l'enquête menée auprès des 373 participants puisque les élèves consacrent en moyenne 12,2 % du temps à tester leur programme. Un constat similaire est opéré avec les entretiens Remind, la moindre fraction du temps consacré s'expliquant par le fait que cette pratique ne nécessite guère d'explication pour être comprise. Si la durée des entretiens avait été fixée à celle des séances, les écarts auraient sans doute été inexistants.

À partir de l'analyse des vidéos et les entretiens Remind, nous expliquons les écarts constatés entre les groupes sur le nombre total de tests opérés et le temps moyen d'un test de trois façons : l'âge des élèves, la séance observée et le type de jeu programmé.

L'âge des élèves impacte la propension des élèves à tester leur jeu. La figure 3 indique que les élèves les plus âgés testent davantage leur jeu. Les élèves de CE2 ont davantage tendance à s'inspirer du programme des jeux modèles, avant de le modifier et se l'approprier une fois leur propre jeu fonctionnel. Le test de chaque nouveau bloc ajouté ne constitue alors plus un impératif puisqu'ils basent leur « création » sur un programme fonctionnel.

La séance pendant laquelle les élèves sont observés est le second facteur expliquant les différences entre les groupes. Les 18 groupes ont été observés à différents niveaux d'avancement de leur jeu. Ainsi, les élèves du groupe 10 (A1/A2 sur la figure 4) en étaient à la séance 10, dans des phases créatives intenses, consacrant peu de temps à programmer des scripts. L'absence de test opéré n'implique pas qu'ils ne testent jamais leur programme, mais simplement qu'ils n'ont pas eu besoin de le faire durant cette séance.

Le type de jeu programmé impacte largement la durée d'un test associé. À titre d'exemple, les élèves du groupe 17 (C1/C2 sur la figure 4), qui ont créé un jeu de plateforme, ont multiplié les tests rapides pendant la séance, au contraire des élèves du groupe 1 (B1/B2 sur la figure 4) qui ont programmé un jeu narratif. Doit-on pour autant favoriser des « jeux courts » qui diminuent les temps de test ? Ce serait négliger l'aspect motivationnel, un facteur important pour justifier le potentiel éducatif des activités de

conception de jeux vidéo à l'école [29]. Les filles seraient ainsi plus motivées lorsque la programmation est intégrée à une activité narrative [30].

Plutôt que d'éliminer certains types de jeux, un renforcement des stratégies de test dans la première phase du programme EGPDP semblerait davantage profitable. La pratique de test et de débogage devrait faire l'objet d'un enseignement spécifique qui pourrait s'inspirer de certaines initiatives récentes [31 ; 32] et faire l'objet d'activités de débogage tangibles [33] à partir d'exemples concrets [34] et de mises en situation liées à la programmation de jeux vidéo.

4.2 Le rôle du test

Les entretiens Remind permettent de comprendre les raisons qui poussent les élèves à tester le jeu vidéo qu'ils programment.

Dans de nombreux cas relevés, test et débogage forment des pratiques associées. Lorsque les élèves testent leur programme, cela conduit parfois à la découverte d'un nouveau problème, qui les amène à tenter de le résoudre, puis de tester la correction apportée. Dans ce cas, le test évalue la solution proposée au problème de codage identifié et traité.

Néanmoins, cette entrée dans un cycle de test-débogage jusqu'à résolution du problème ne semble pas refléter la majorité. Celle-ci voit les tests plus souvent utilisés pour s'assurer du bon fonctionnement des nouveaux éléments ajoutés à leur jeu, comme de leur intérêt et leur difficulté pour le jeu lui-même. On se situe ici plus dans une dimension ludique qu'informatique, la création du jeu étant étroitement liée à la pratique du jeu [9]. Le test prend une dimension plus générale et intègre pleinement l'évaluation et l'amélioration itérative du jeu en tant que tel. Il fait office de validation pour l'élève, que celle-ci porte sur le code ou la conception du jeu, et s'accompagne souvent d'une émotion positive ressentie, voire d'une frustration lorsque le résultat ne correspond pas aux attentes.

Le dernier cas de figure rencontré concerne l'utilisation du test dans la seule optique de divertissement. L'échantillon d'élèves observé nous suggère que cette pratique est mineure. Le jumelage des élèves avec leurs amis semble lié à une interaction hors-jeu les éloignant de la tâche selon certaines études [18]. Nous n'observons pas cet effet à travers une hypothétique multiplication des tests à visée purement ludique.

Les tests sont ainsi le plus souvent utilisés pour évaluer le nouvel élément de jeu programmé, notamment son aspect ludique. En cela, les séances de test collectives offrent des retours d'expérience contribuant à l'équilibrage du jeu. Nous recommandons leur instauration lors de la mise en place d'activités de ce type à l'école.

4.3 Lien entre pratique de test et concepts informatiques

Les questionnaires d'auto-évaluation comme l'analyse des 18 groupes d'élèves observés montrent que les élèves qui testent le plus leur programme sont ceux qui obtiennent les meilleurs résultats au post-test, mais aussi ceux qui progressent le plus. L'abondance de tests opérés ne témoignerait donc pas d'un manque de compétences en programmation, amenant ces élèves à procéder par essais-erreurs pour corriger des bugs

par manque d'approche stratégique, niveau élémentaire des stratégies de débogage [11].

En rentrant dans une démarche de résolution de problèmes de codage et de développement incrémental, les phases de test constituent un maillon essentiel du prototypage itératif de création d'une solution à une situation-problème, associée ici à un élément du jeu vidéo conçu [35]. Les élèves ne l'ayant pas assimilé sont ainsi moins confrontés à des situations leur permettant d'assimiler les concepts nécessaires à la résolution d'un programme, encore moins à progresser dans un test mesurant leur maîtrise dans des situations décontextualisées. Au contraire, les élèves les plus à l'aise, pourtant les plus à même de se passer d'une phase validant l'ajout de scripts élémentaires, ont tendance à systématiquement tester chaque bloc, une pratique qui tend à se réduire à mesure qu'ils améliorent leur capacité à simuler mentalement l'exécution d'un programme et à utiliser des algorithmes [11].

Nous voyons ici comment concepts et pratiques informatiques sont liées. Il est donc essentiel que les programmes intègrent ces deux aspects dans leur plan d'apprentissage. La conception de jeux vidéo constitue un bon moyen le permettant, en particulier par rapport à des exercices de codage plus classiques [36], puisque les élèves sont contraints d'apprendre à décomposer leur jeu en différents éléments, à les programmer de manière itérative en faisant preuve d'adaptation, à isoler un problème pour le tester et le déboguer, à réutiliser des parties de scripts fonctionnels issus d'autres jeux, autant de pratiques renforçant la pensée informatique de l'élève selon toutes ses dimensions.

5 Conclusion

L'objectif de cet article est d'examiner comment, lors d'activités de conception de jeux vidéo, se manifeste la pratique de test. L'étude menée auprès de 373 élèves de 9 à 13 ans ayant participé au programme EGPDP nous permet de mesurer l'importance de cette composante de la pensée informatique, largement mobilisée par les participants.

Le test est majoritairement utilisé par les élèves pour évaluer un élément du jeu ou du programme, et prend une place essentielle dans le processus d'évaluation et d'amélioration itérative du programme créé. Dans certains cas, minoritaires, le test s'accompagne de la découverte d'un problème et de l'entrée dans un cycle de test-débogage. Le test s'intègre alors pleinement dans le processus de résolution du problème puisqu'il valide ou invalide la solution proposée.

La fréquence de test est positivement corrélée aux résultats du test mesurant la maîtrise des concepts de la pensée informatique. Cela renforce l'idée d'un développement conjoint des concepts et des pratiques de la pensée informatique dans les programmes d'enseignement. Ces dernières ne sont pas naturellement maîtrisées par les apprenants et nécessitent un apprentissage spécifique.

Cette étude pourrait être développée. Un échantillon plus important d'élèves nous permettrait de davantage comprendre les différentes utilisations du test en fonction du profil des élèves, et comment le type de jeu conçu impacte celles-ci. L'influence de l'enseignant dans le développement de la pratique de test pourrait aussi être étudiée.

References

1. Voogt, Joke, Petra Fisser, Jon Good, Punya Mishra, et Aman Yadav. 2015. « Computational Thinking in Compulsory Education: Towards an Agenda for Research and Practice ». *Education and Information Technologies* 20(4):715-28. doi: 10.1007/s10639-015-9412-6.
2. Wing, JM. 2011. « A definition of computational thinking from Jeannette Wing ». *Computing educational research blog*.
3. Fagerlund, Janne, Päivi Häkkinen, Mikko Vesisenaho, et Jouni Viiri. 2021. « Computational Thinking in Programming with Scratch in Primary Schools: A Systematic Review ». *Computer Applications in Engineering Education* 29(1):12-28. doi: 10.1002/cae.22255.
4. Standl, Bernhard. 2017. « Solving Everyday Challenges in a Computational Way of Thinking ». P. 180-91 in *Informatics in Schools: Focus on Learning Programming*. Vol. 10696, édité par V. Dagienė et A. Hellas. Cham: Springer International Publishing.
5. Denning, Peter. 2017. « Computational Thinking in Science ». *American Scientist* 105:13-17.
6. Brennan, K., et M. Resnick. 2012. « New Frameworks for Studying and Assessing the Development of Computational Thinking ». P. 25 in *2012 Annual Meeting of the American Educational Research Association*. Vol. 1. Vancouver.
7. Grover, Shuchi, et Roy Pea. 2017. « Computational Thinking: A Competency Whose Time Has Come ». P. 20-38 in *Computer Science Education: Perspectives on Teaching and Learning in School*. Bloomsbury Publishing.
8. Zhang, LeChen, et Jalal Nouri. 2019. « A systematic review of learning computational thinking through Scratch in K-9 ». *Computers & Education* 141:103607. doi: https://doi.org/10.1016/j.compedu.2019.103607.
9. Kafai, Yasmin, et Veena Vasudevan. 2015. « Constructionist Gaming Beyond the Screen: Middle School Students' Crafting and Computing of Touchpads, Board Games, and Controllers ». P. 49-54 in.
10. Falloon, G. 2016. « An analysis of young students' thinking when completing basic coding tasks using Scratch Jnr. On the iPad: General thinking and computational work ». *Journal of Computer Assisted Learning* 32. doi: 10.1111/jcal.12155.
11. Yan, Wei, Feiya Luo, Maya Israel, Ruohan Liu, et Latoya Chandler. 2025. « How Do Elementary Students Apply Debugging Strategies in a Block-Based Programming Environment? » *Education Sciences* 15:292. doi: 10.3390/educsci15030292.
12. Cafarella, Lorien, et Lucas Vasconcelos. 2024. « Computational thinking with game design: An action research study with middle school students ». *Education and Information Technologies* 30:5589-5633. doi: 10.1007/s10639-024-13010-5.
13. Webb, Heidi, et Mary Beth Rosson. 2013. « Using scaffolded examples to teach computational thinking concepts ». P. 95-100 in.
14. Kim, ChanMin, Jiangmei Yuan, Lucas Vasconcelos, et Minyoung Shin. 2018. « Debugging during block-based programming ». *Instructional Science* 46. doi: 10.1007/s11251-018-9453-5.
15. Tashakkori, Abbas, et Charles Teddlie. 2010. *SAGE Handbook of Mixed Methods in Social & Behavioral Research*. 2455 Teller Road, Thousand Oaks California 91320 United States: SAGE Publications, Inc.
16. Biesta, Gert. 2010. « Pragmatism and the philosophical foundations of mixed methods research ».
17. Le Du, Jérémy, Julian Alvarez, et Daniel Schmitt. 2024. « Développer la pensée informatique à travers la conception de jeux vidéo éducatifs Educational Game & Play Design

Project : élaboration de la méthode ». *Distance et Médiation des savoirs* (48). doi: https://doi.org/10.4000/12xon.
18. Campe, Shannon, Jill Denner, Emily Green, et David Torres. 2020. « Pair Programming in Middle School: Variations in Interactions and Behaviors ». *Computer Science Education* 30(1):22-46. doi: 10.1080/08993408.2019.1648119.
19. Mayer, Richard. 2018. « Thirty Years of Research on Online Learning ». *Applied Cognitive Psychology* 33. doi: 10.1002/acp.3482.
20. Schmitt, Daniel, et Olivier Aubert. 2017. « REMIND : une méthode pour comprendre la micro-dynamique de l'expérience des visiteurs de musées ». *RIHM, Revue des Interactions Humaines Médiatisées* 17:43-70.
21. Schindler, Maike, et Achim Lilienthal. 2019. « Students' Creative Process in Mathematics: Insights from Eye-Tracking-Stimulated Recall Interview on Students' Work on Multiple Solution Tasks ». *International Journal of Science and Mathematics Education*. doi: 10.1007/s10763-019-10033-0.
22. Theureau, Jacques. 2006. *Le cours d'action: Méthode développée*. Octares Editions.
23. Romero, Margarida. 2017. « Les compétences pour le XXI e siècle ». in *Usages créatifs du numérique pour l'apprentissage au XXIe siècle*.
24. Le Du, Jérémy, Daniel Schmitt, et Julian Alvarez. Reviewed. « Assessing 21st Century Skills with Stimulated Recall Interviews with Situated Subjective Perspective: The Remind Method ».
25. Román-González, Marcos. 2015. « Computational Thinking Test: Design Guidelines and Content Validation ». P. 2436-44 in *7th International Conference on Education and New Learning Technologies*. Barcelona, Spain.
26. Román-González, Marcos, Juan-Carlos Pérez-González, et Carmen Jiménez-Fernández. 2017. « Which Cognitive Abilities Underlie Computational Thinking? Criterion Validity of the Computational Thinking Test ». *Computers in Human Behavior* 72:678-91. doi: 10.1016/j.chb.2016.08.047.
27. Wiebe, Eric, Jennifer London, Osman Aksit, Bradford W. Mott, Kristy Elizabeth Boyer, et James C. Lester. 2019. « Development of a Lean Computational Thinking Abilities Assessment for Middle Grades Students ». P. 456-61 in *SIGCSE '19: The 50th ACM Technical Symposium on Computer Science Education*. Minneapolis MN USA: ACM.
28. El-Hamamsy, Laila, María Zapata-Cáceres, Estefanía Martín Barroso, Francesco Mondada, Jessica Dehler Zufferey, et Barbara Bruno. 2022. « The Competent Computational Thinking Test: Development and Validation of an Unplugged Computational Thinking Test for Upper Primary School ». *Journal of Educational Computing Research* 60(7):1818-66. doi: 10.1177/07356331221081753.
29. Repenning, Alexander, David C. Webb, Kyu Han Koh, Hilarie Nickerson, Susan B. Miller, Catharine Brand, Ian Her Many Horses, Ashok Basawapatna, Fred Gluck, Ryan Grover, Kris Gutierrez, et Nadia Repenning. 2015. « Scalable Game Design: A Strategy to Bring Systemic Computer Science Education to Schools through Game Design and Simulation Creation ». *ACM Trans. Comput. Educ.* 15(2). doi: 10.1145/2700517.
30. Howland, Kate, et Judith Good. 2015. « Learning to communicate computationally with Flip: A bi-modal programming language for game creation ». *Computers & Education* 80:224-40. doi: 10.1016/j.compedu.2014.08.014.
31. Kim, ChanMin, Lucas Vasconcelos, Brian Belland, Duygu Umutlu, et Cory Gleasman. 2022. « Debugging behaviors of early childhood teacher candidates with or without scaffolding ». *International Journal of Educational Technology in Higher Education* 19. doi: 10.1186/s41239-022-00319-9.

32. Gao, Xuemin, et Khe Hew. 2023. « A Flipped Systematic Debugging Approach to Enhance Elementary Students' Program Debugging Performance and Optimize Cognitive Load ». *Journal of Educational Computing Research* 61:1-32. doi: 10.1177/07356331221133560.
33. Socratous, Chrysanthos, et Andri Ioannou. 2020. « Common Errors, Successful Debugging, and Engagement During Block-Based Programming Using Educational Robotics in Elementary Education ».
34. Bofferding, Laura, Sezai Kocabas, Mahtob Aqazade, et Lizhen Chen. 2022. « The Effect of Play and Worked Examples on First and Third Graders' Creating and Debugging of Programming Algorithms ».
35. Romero, Margarida, Alexandre Lepage, et Benjamin Lille. 2017. « Computational thinking development through creative programming in higher education ». *International Journal of Educational Technology in Higher Education* 14. doi: 10.1186/s41239-017-0080-z.
36. Kafai, Yasmin B., et Quinn Burke. 2015. « Constructionist Gaming: Understanding the Benefits of Making Games for Learning. » *Educational Psychologist* 50(4):313-34. doi: 10.1080/00461520.2015.1124022.