Enseignement explicite du débogage systématique et transfert de compétences

Mémoire de Master

Didactique de l'informatique

Juin 2024

Auteur : Nicolas Tuor

Sous la direction de : Engin Bumbacher

Membre du jury : Morgane Chevalier

Remerciements

La réalisation de ce mémoire a été possible grâce au concours de plusieurs personnes à qui je voudrais témoigner toute ma gratitude.

Premièrement, je remercie mon directeur de mémoire, Engin Bumbacher pour sa bienveillance dans son accompagnement, ses conseils, et la confiance qu'il m'a témoignée en acceptant d'encadrer ce travail.

Je remercie chaleureusement les enseignants qui m'ont accueilli dans leur classe ainsi que leurs élèves pour tout le temps qu'ils m'ont accordé.

Merci également à Laila El-Hamamsy pour l'encadrement, les relectures, remarques et conseils.

Enfin, un grand merci à ma famille pour leur soutien durant toutes ces années d'études.

Résumé

Ce travail avait pour objectif d'analyser l'effet de l'enseignement explicite des techniques de débogage sur le transfert des compétences des élèves de 5H (8-9 ans) en programmation et en débogage. Deux hypothèses ont été formulées : 1) l'enseignement explicite du débogage améliore les compétences en programmation, et 2) les capacités de débogage s'améliorent dans le langage enseigné (Scratch) et sont transférables à d'autres langages.

Une séquence pédagogique structurée a été mise en place, comprenant une introduction à la programmation avec Scratch, suivie d'une introduction à VPL (Thymio) et d'un enseignement explicite du débogage systématique. Des tests de programmation et de débogage ont été administrés pour mesurer la progression des élèves.

Les résultats ont montré une amélioration significative des scores de programmation entre le pré-test et le post-test ($p=0,009$), avec un effet modéré ($d=0,43$), validant partiellement la première hypothèse. Cependant, l'amélioration entre le mid-test et le post-test n'était pas significative ($p=0,131$), suggérant que l'intervention sur le débogage avait un impact inférieur à l'introduction à la programmation.

En ce qui concerne le débogage, les scores se sont significativement améliorés ($p<0,001$) avec un effet important ($r>0,5$), à la fois dans Scratch et dans d'autres langages comme VPL (Thymio), confirmant la deuxième hypothèse sur la transférabilité des compétences.

Mots clés

Débogage, didactique de l'informatique, programmation, débogage systématique

Dans ce travail, l'emploi du masculin pour désigner des personnes n'a d'autres fins que celles d'alléger le texte et de renforcer l'anonymat des élèves.

Table des matières

Table des figures

Index des tableaux

1. Introduction

L'enseignement de la programmation implique non seulement l'acquisition de connaissances théoriques, mais aussi le développement de compétences pratiques essentielles. Elle implique de créer, de tester et de corriger du code. Ce dernier processus, appelé débogage, est souvent difficile (Lahtinen et al., 2005) et frustrant (Michaeli & Romeike, 2019a) pour les élèves qui apprennent à programmer (Fitzgerald et al., 2008). Pourtant, le débogage est une compétence essentielle qui nécessite un enseignement explicite et adapté (Gao & Hew, 2023).

Dans ce contexte, la présente étude qui s'inscrit dans le domaine de la didactique de l'informatique vise à analyser la transférabilité des compétences de débogage avec des élèves de 5ème année Harmos (8-9 ans) sous trois angles (détails section 3) : celui de la programmation, du débogage au sein du langage d'apprentissage, et vers d'autres langages (« généralisation »). En mettant en place une séquence pédagogique structurée, nous avons introduit les élèves aux langages de programmation Scratch (séquentiel, https://scratch.mit.edu/) et VPL (robot éducatif Thymio, événementiel, https://www.thymio.org/) puis les avons entraînés à appliquer une méthode de débogage systématique uniquement avec le langage Scratch.

Grâce à des tests de programmation et de débogage que les élèves ont passé à des moments clés, nous avons cherché à mettre en lumière l'importance d'intégrer l'enseignement explicite du débogage dans les programmes scolaires d'informatique, afin de préparer les élèves aux défis technologiques actuels.

2. Cadre théorique

Dans cette partie, nous définirons quelques concepts clés (2.1) et présenterons l'état actuel de la recherche dans le domaine de l'enseignement du débogage (2.2), en particulier les études que nous avons adaptées à notre contexte pour développer le dispositif d'enseignement (2.3 et 2.4) afin de poser le contexte dans lequel s'inscrit ce travail. Nous détaillerons ces adaptations en section 4.

2.1 Concepts clés

Le débogage est généralement défini comme l'ensemble des actions entreprises par les programmeurs pour identifier et corriger les erreurs (bugs) dans leur code (Michaeli & Romeike, 2019 a et b). Il s'agit d'un processus itératif, qui consiste à diagnostiquer puis réparer les erreurs dans un programme défectueux de manière systématique. Comme il est une compétence distincte parmi celles utilisées en programmation, il nécessite donc une attention particulière pour éviter une surcharge cognitive (Michaeli & Romeike, 2019a, Gao & Hew, 2023). Ces auteurs recommandent par ailleurs de l'enseigner explicitement, un programmeur compétent n'étant pas forcément bon en débogage, mais un bon débogueur sera bon en programmation (Romero et al., 2007, Fitzgerald et al., 2008, Ahmadzadeh et al., 2005, cités dans Michaeli & Romeike, 2019b).

Les techniques de débogage sont des méthodes ou des outils qui facilitent le débogage. Les techniques de débogage peuvent consister à relire le code, le tester par parties, le commenter, le partager avec d'autres, ou à utiliser des fonctionnalités spécifiques de l'environnement de programmation (par exemple la fonction « print » ou des modules d'assistance) (Gao & Hew, 2023 ; Michaeli & Romeike, 2019a). Gao & Hew (2023), en partant des résultats de Vessey (1985, dans Gao & Hew, 2023), Carver & Risinger (1987, dans Gao & Hew, 2023) et Yoon & Garcia (1998, dans Gao & Hew) proposent quatre étapes principales : « (1) l'identification et la représentation des problèmes (par exemple, la comparaison des objectifs et des résultats du programme), (2) la localisation des bogues (par exemple, la compréhension de la structure du programme), (3) la correction des bogues et (4) l'évaluation des solutions » (traduit de Gao & Hew, 2023, p.1066).

L'enseignement explicite des techniques de débogage à l'aide de différentes méthodes (pédagogiques et didactiques) est bénéfique. Voici celles que nous avons sélectionnées dans la littérature :

- Afficher un modèle de débogage systématique en classe après l'avoir expliqué pour que les élèves puissent l'utiliser (Michaeli & Romeike, 2019b).
- Identifier les différents types d'erreurs avec les élèves pour qu'ils les traitent différemment car on n'aborde pas une erreur de syntaxe comme on le ferait avec une erreur sémantique (Michaeli & Romeike, 2019a).
- L'approche inversée, qui consiste à faire apprendre aux élèves les connaissances de base avant le cours, et à leur faire pratiquer des activités de débogage pendant le cours (Gao & Hew, 2023).
- Le processus de débogage systématique (PDS, voir figure 5, Gao & Hew, 2023, Michaeli & Romeike, 2019b), qui propose une démarche en quatre étapes pour identifier et résoudre les erreurs de programmation (identification et représentation du problème, localisation du bogue, correction du bogue et évaluation de la solution, Gao & Hew, 2023).
- La méthode de modélisation, qui consiste à montrer aux élèves le processus de débogage d'un expert, en expliquant les stratégies et les raisonnements utilisés (Gao & Hew, 2023).

La spécificité de notre travail par rapport aux études existantes (section 2.2) est de chercher à observer si un transfert des compétences en débogage est possible du langage de programmation dans lequel elles sont acquises vers d'autres langages. *Le transfert des apprentissages* est la capacité à utiliser les connaissances et les compétences acquises dans un contexte donné dans un autre contexte différent et nouveau. Selon Perkins et Salomon (1992), le transfert peut être de deux types : le transfert proche, qui se produit lorsque les contextes sont similaires, et le transfert lointain, qui se produit lorsque les contextes sont différents. La limite entre les deux est assez floue, la notion de proximité entre des connaissances n'ayant pas d'unité de mesure précise. Pour favoriser le transfert des apprentissages, il faut enseigner les concepts et les processus de manière explicite, variée et réflexive, en aidant les apprenants à faire des liens entre les situations sources et cibles (Samson, 2012). Notre méthode d'enseignement explicite du débogage vise à répondre à ces critères.

2.2 État actuel de la recherche

La thématique de recherche abordée dans ce travail rassemble plusieurs aspects principaux. D'un côté, elle s'intéresse au débogage et son enseignement, et de l'autre aux effets de l'apprentissage du débogage sur la programmation et le débogage. Compte tenu des spécificités de notre contexte et de notre sujet, seul un nombre très restreint de recherches restent pertinentes (pour le manque de recherches sur l'enseignement du débogage, Michaeli & Romeike, 2019b).

En effet, nous nous intéressons à un public plutôt jeune d'élèves d'école primaire (8-9 ans), ce qui implique d'utiliser un langage accessible (les élèves n'ont notamment pas de connaissances en anglais). Notre choix s'est porté sur Scratch pour le nombre de ressources à disposition et sa simplicité d'utilisation. De fait, les recherches utilisables se limitent à celles menées avec des enfants, en utilisant un langage de programmation par blocs (pour ne pas se limiter à Scratch), et avec un focus sur le débogage. La plupart de celles que nous avons pu trouver sont effectuées avec des adolescents ou des adultes (Ahmadzadeh et al., 2005, Chiu & Huang, 2015), en utilisant des langages de programmation textuels (Ahmadzadeh et al., 2005, Lee et al., 2014) ou des objets physiques comme des robots ou des tablettes (Heikkilä & Mannila, 2018, Neutens & Wyffels, 2020) voire se concentrent sur d'autres aspects influençant l'apprentissage (Zúñiga Muñoz et al., 2023). Nous les avons toutefois consultées pour nous fournir des pistes de réflexion sur les approches (dispositifs), les définitions, les types de bugs proposés, les formats des exercices et des tests utilisés.

Nous nous concentrerons donc sur les études qui correspondent le plus à notre contexte pour présenter un aperçu des différents dispositifs testés dans la recherche, afin d'en tirer des éléments pertinents pour la justification de notre protocole d'enseignement explicite du débogage, ciblant des élèves de ou proche de l'âge de 8-9 ans, sans compétences préalables en programmation et utilisant un langage de programmation par blocs.

2.3 Enseignement du débogage

Dans « Current Status and Perspectives of Debugging in the K12 Classroom : A Qualitative Study. », Michaeli et Romeike (2019a) se sont intéressés à la place actuelle du débogage dans l'enseignement, au K12 (école obligatoire pour nous). Ils y font également état du manque de littérature sur le sujet : « when it comes to teaching debugging, there are surprisingly few studies and results ; this is true in both academic (university) settings and K12 » (p.2).

2.3.1 Points clés

L'article relève la distinction entre les compétences de débogage et les compétences générales de programmation, le débogage comportant son propre sous-ensemble de compétences, l'importance d'un processus de débogage systématique, la nécessité d'enseigner les techniques de débogage et le manque d'enseignement explicite des compétences de débogage à la fois dans les établissements d'enseignement supérieur et dans les écoles primaires et secondaires.

Les freins identifiés à un bon enseignement des techniques de débogage sont le manque de temps, de concepts et de matériel pour les enseigner et le fait qu'il ne s'agisse pas d'un contenu explicite du programme.

Les auteurs concluent que les stratégies de débogage et les compétences en lien doivent faire l'objet d'un enseignement à part, car elles sont souvent enseignées de manière décousue (lorsqu'elles le sont), sans structure claire. Cette démarche profiterait aux élèves de niveaux faible à moyen qui favorisent généralement des approches par essai-erreur, peu efficace, et peuvent ainsi accaparer beaucoup de temps à l'enseignant qui se retrouve à devoir passer d'un ordinateur à l'autre pour aider les élèves à résoudre leurs problèmes.

2.3.2 Première méthode de débogage

Dans leur seconde recherche, Michaeli et Romeike (2019b) se sont ainsi intéressé à l'enseignement explicite d'un processus de débogage systématique.

Dans deux classes du secondaire (élèves de 15-16 ans), les auteurs ont mis en place une séquence d'enseignement consistant en un pré-test (exercices de débogage et questionnaire), une leçon (exercices de débogage pour un groupe, intervention expliquant la méthode systématique pour l'autre) et un post-test sur le même modèle que le pré-test (Fig. 1). Les langages BlueJ et Java ont été utilisés, les élèves les connaissaient déjà étant donné que la recherche a été menée en cours d'année, avec des classes au même état d'avancement du programme.

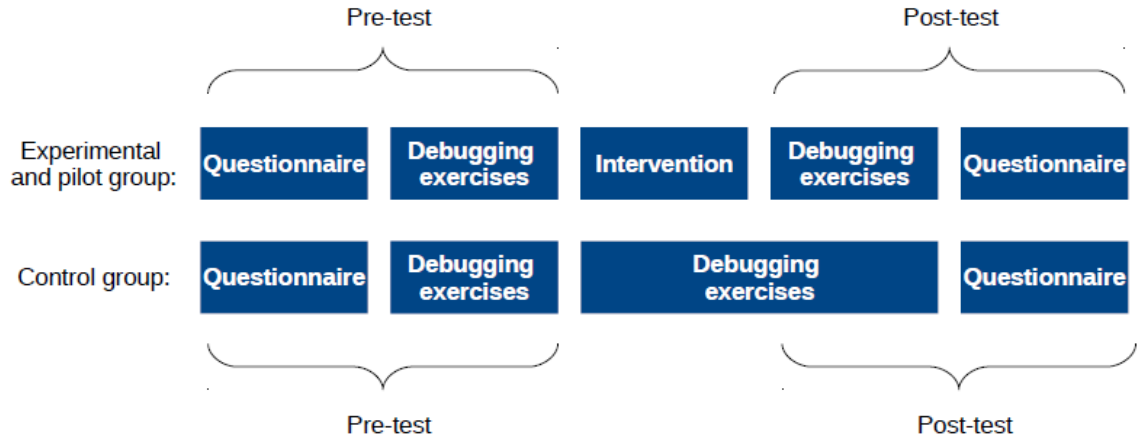


Figure 1: Schéma de l'étude, Michaeli & Romeike, 2019b

Leur intervention consistait en une explication des différents types de bugs et une présentation d'un schéma (Fig. 2) pour aborder les types de bugs de manière stratégique.

La technique introduite est celle du débogage systématique, que nous avons adaptée pour notre situation. Elle consiste à utiliser une stratégie de débogage pour éviter l'approche à tâtons que les novices peuvent instinctivement privilégier.

Les élèves ayant suivi l'intervention ont réussi à corriger plus d'erreurs dans les exercices de débogage entre le pré et le post-test que les élèves du groupe contrôle. Nous nous sommes ainsi intéressés aux parties nous concernant, c'est-à-dire le test utilisé et les problèmes potentiels auxquels nous pourrions être confrontés dans notre étude.

La méthode en détails

Les auteurs ont schématisé le processus (voir Fig. 2) et l'ont affiché en classe en tout temps pour que les élèves se rappellent de l'utiliser activement. Ils sont ainsi invités à commencer par tester si leur code se compile, vérifier que la structure est respectée et lisible par l'interpréteur (étape 1 « Compile » sur le schéma). S'il y a une erreur à cette étape, ils lisent le message d'erreur, essaient de le comprendre pour ajuster leur code et réessaient jusqu'à ce que le programme se compile. Si le code se compile, ils passent à l'étape 2, exécuter le code (« Run »). S'il y a une erreur, ils doivent à nouveau lire le message et le comprendre. Ici, deux étapes supplémentaires s'ajoutent, d'abord trouver la cause du bug en élaborant une hypothèse (par exemple dans un jeu de plateforme, le personnage ne saute pas, le problème est peut-être localisé dans l'attribution de la touche sensée le faire sauter). Dans un second temps, trouver là où les lignes de codes correspondant à leur hypothèse. Enfin, appliquer un correctif et tester. Si le problème est toujours présent, on recommence, s'il est réparé, on passe à l'étape suivante.

La dernière étape (étape 3, « Compare ») consiste à comparer le résultat produit et le résultat attendu. Si les versions divergent, on formule à nouveau une hypothèse, on recherche la partie du code potentiellement responsable, et on applique le changement avant de tester. Si ça marche, le problème est résolu, sinon on refait une boucle.

Chaque étape a deux éléments communs : on teste après chaque modification apportée et si elle n'a pas marché, on revient en arrière notamment pour éviter l'accumulation de bugs.

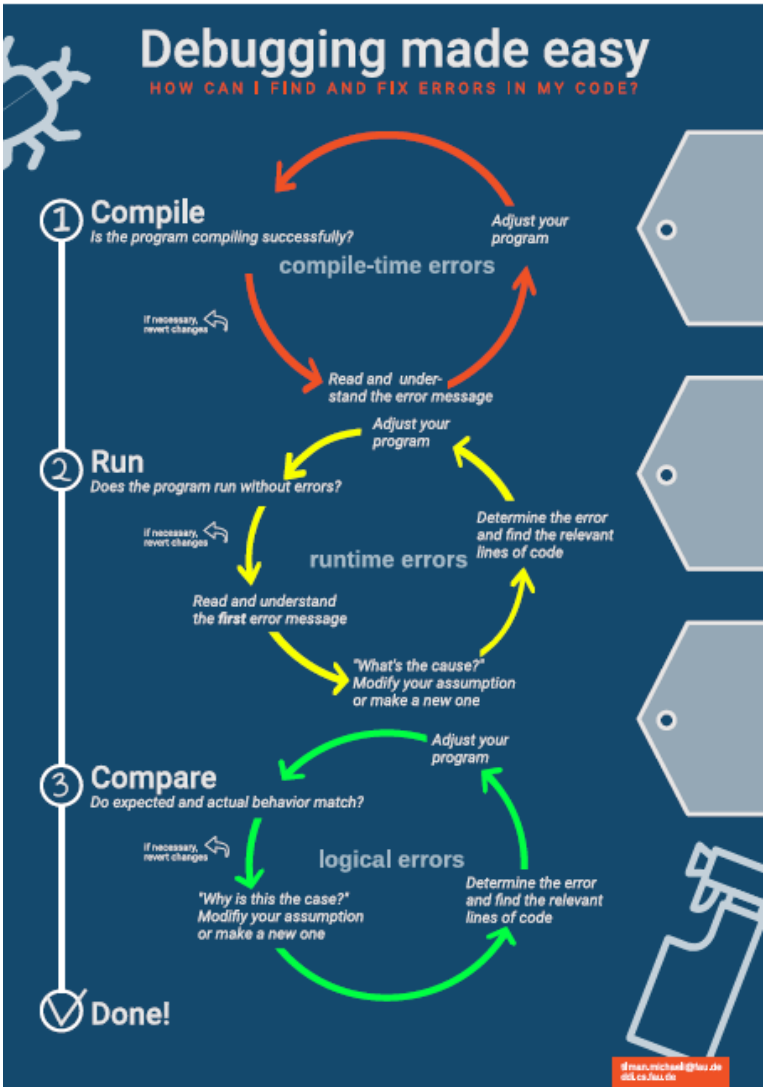


Figure 2: Schéma de débogage systématique, Michaeli & Romeike, 2019b

Dans le dispositif de test, les programmes à déboguer étaient repris d'exercices vu en classe afin de limiter la quantité de code inconnu, ce dernier représentant un défi d'adaptation pour des élèves novices. Les bugs étaient apparents et leur nombre donné à l'avance, l'objet observé étant les compétences de débogage les auteurs voulaient que les élèves puissent directement

commencer à déboguer. Les bugs étaient toutefois de natures variées, nécessitant des approches différentes.

Concepts pris en compte dans notre adaptation

Les obstacles les plus souvent rencontrés par les programmeurs novices relevés dans cet article nous ont aidés durant la conception de notre séquence, car les élèves testés dans notre travail étaient des débutants en programmation. Ils peuvent ainsi se retrouver facilement bloqués, ce qui, combiné au concept de « learned helplessness » (2019b, p. 6) peut les pousser à ne rien faire en attendant l'aide de l'enseignant. Ceci est toutefois évitable à l'aide du concept de « self-reliance » qui permet aux élèves de continuer leur tâche car ils se savent outillés pour y arriver. L'enseignement explicite d'une méthode de débogage systématique est donc très important car elle donne aux élèves une technique, une marche à suivre à laquelle se référer en cas de blocage. Les programmeurs sont ainsi invités à réfléchir activement aux problèmes rencontrés, leur permettant de gagner en autonomie.

Lorsqu'un novice débogue un programme, il risque d'introduire de nouvelles erreurs s'il ne procède pas systématiquement en testant ses correctifs avant de continue. La méthode enseignée vise également à pallier ce risque et prévenir la technique de l'essai-erreur, notamment en apprenant aux élèves à faire la distinction entre les différents types de bugs leur permettant d'utiliser des stratégies de résolution adaptées.

Les auteurs concluent également que leur approche n'est pas dépendante d'un langage de programmation, car elle a donné des résultats positifs en étant testée en Java, BlueJ, Stride et Greenfoot « the positive results in both cases indicate that this approach is independent of tools and (text-based) programming languages » (p.6). Il s'agissait toutefois d'enseigner et tester dans un langage, là où nous enseignerons dans un langage et testerons dans un autre.

2.3.3 Seconde méthode de débogage

Une autre étude a réutilisé cette approche, il s'agit d'« A Flipped Systematic Debugging Approach to Enhance Elementary Students' Program Debugging Performance and Optimize Cognitive Load " de Gao, X., & Hew, K. F. (2023).

Ils ont appliqué une version adaptée de celle de Michaeli et Romeike (2019b), en y amenant un focus sur la charge cognitive et l'utilisation d'un modèle de classe inversée et d'enseignement explicite du débogage modifiés, appliqués au groupe témoin. Dans le modèle de la classe inversée, les élèves ont dû apprendre les concepts de base avant les cours à l'aide de vidéos et de quiz en ligne. L'étude a été menée sur quatre classes d'élèves de 10-11 ans (7H), pour un

total de 158 élèves séparés en un groupe contrôle et un groupe expérimental. L'intervention a duré 4 semaines, avec une leçon par semaine plus les contenus à distance (Fig. 3 et 4). Ils ont utilisé le langage Scratch, comme pour notre situation.

La méthode en détails

Au début des séances en classe, l'enseignant a commencé par un bref rappel des contenus vus avant le cours pour les réactiver. Les questions du quiz (fait avant le cours) présentant un taux d'erreur élevé ont ensuite été abordées pour éviter les malentendus. L'enseignant a continué avec une présentation du processus de la méthode de débogage systématique sur des exemples concrets. Enfin, les élèves ont pu s'entraîner à appliquer la méthode eux-mêmes sur des exercices.

Les auteurs ont relevé une amélioration des capacités de débogage ainsi qu'une réduction des charges cognitives intrinsèque et extrinsèque et une augmentation de l'investissement des élèves dans la charge essentielle (« germane », contribuant directement à l'apprentissage).

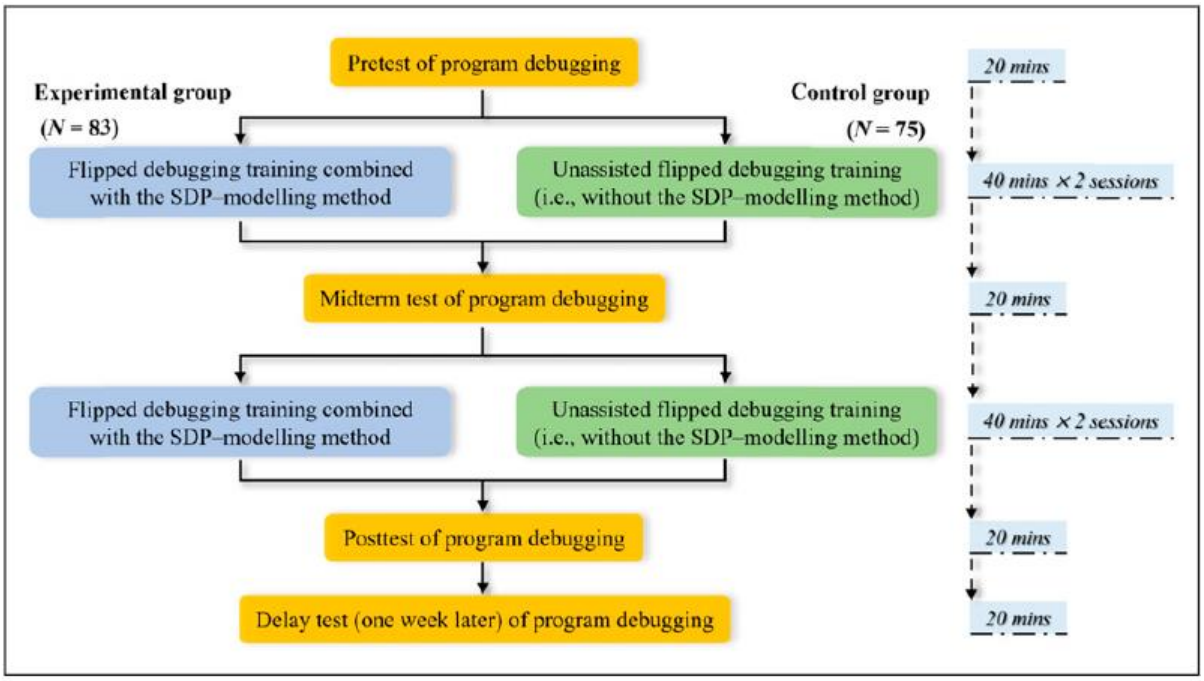
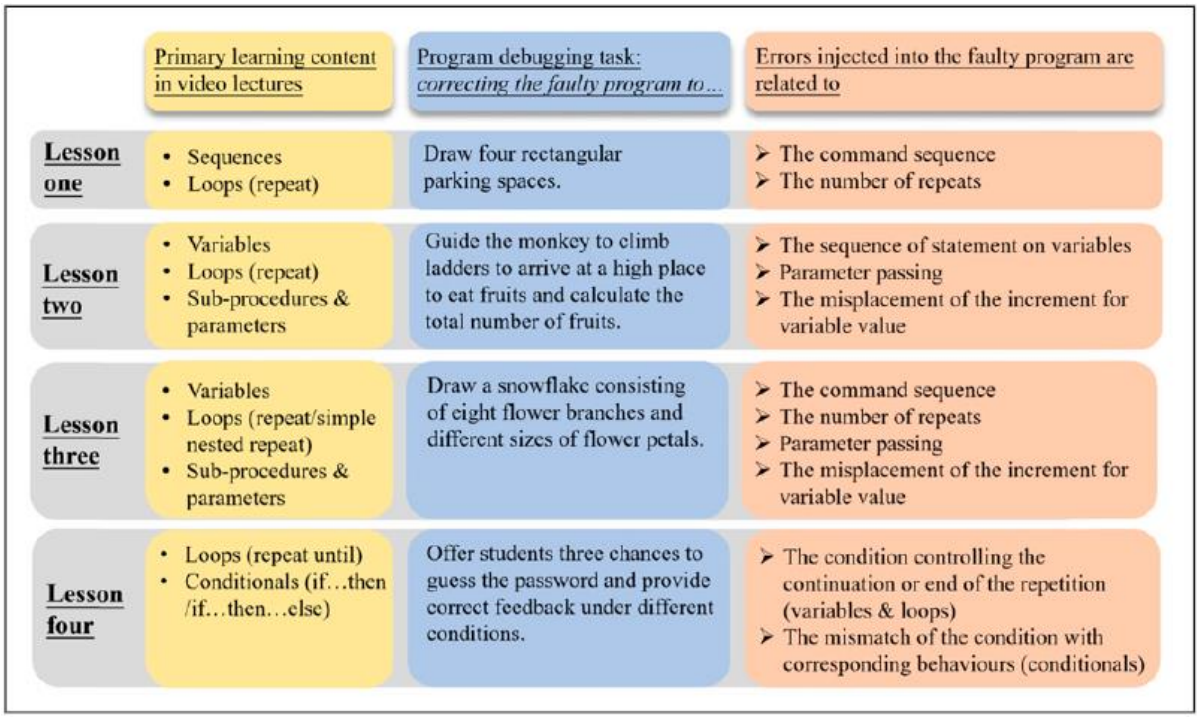


Figure 3: Forme de l'intervention (Gao & Hew, 2023)



	Primary learning content in video lectures	Program debugging task: *correcting the faulty program to…*	Errors injected into the faulty program are related to
Lesson one	• Sequences • Loops (repeat)	Draw four rectangular parking spaces.	➢ The command sequence ➢ The number of repeats
Lesson two	• Variables • Loops (repeat) • Sub-procedures & parameters	Guide the monkey to climb ladders to arrive at a high place to eat fruits and calculate the total number of fruits.	➢ The sequence of statement on variables ➢ Parameter passing ➢ The misplacement of the increment for variable value
Lesson three	• Variables • Loops (repeat/simple nested repeat) • Sub-procedures & parameters	Draw a snowflake consisting of eight flower branches and different sizes of flower petals.	➢ The command sequence ➢ The number of repeats ➢ Parameter passing ➢ The misplacement of the increment for variable value
Lesson four	• Loops (repeat until) • Conditionals (if…then /if…then…else)	Offer students three chances to guess the password and provide correct feedback under different conditions.	➢ The condition controlling the continuation or end of the repetition (variables & loops) ➢ The mismatch of the condition with corresponding behaviours (conditionals)

Figure 4: Contenus abordés en cours (Gao & Hew, 2023)

Les contenus des leçons ont été centrés sur des concepts de programmation (Fig. 4), pour amener les connaissances nécessaires à la résolution des différents types de bugs relatifs à ces concepts au moment où ils sont étudiés. De cette manière, les élèves se concentrent sur un aspect à la fois et s'entraînent à utiliser des approches différentes dans un contexte efficace. Les concepts sont régulièrement repris pour faire des rappels ; le tout est conçu pour optimiser les charges cognitives.

Les élèves ont passé quatre tests de débogage pour évaluer leurs compétences en débogage. Le premier test a été effectué avant les leçons (prétest), le deuxième test entre la deuxième et la troisième leçon (test intermédiaire), le troisième test après les leçons (posttest) et le quatrième test une semaine après les leçons (test différé).

Concepts pris en compte dans notre adaptation

Chaque test comportait 11 tâches de débogage avec 27 erreurs logiques à corriger, les élèves devaient corriger le plus d'erreurs possible en 20 minutes. Comme chez Michaeli et Romeike (2019b), les programmes à déboguer réutilisaient du code vu en classe.

Le schéma utilisé pour le débogage systématique varie quelque peu (Fig. 5 page suivante). Il s'agit d'un seul cycle en quatre étapes. (1) Identification et représentation du problème : comparer les objectifs et les résultats du programme, et formuler des hypothèses sur les erreurs. (2) Localisation des erreurs : comprendre la structure et la logique du programme, et supposer

les emplacements des erreurs. (3) Correction des erreurs : proposer et appliquer des solutions pour réparer les erreurs. (4) Évaluation des solutions : vérifier si les solutions ont résolu les erreurs, et recommencer le cycle de débogage si nécessaire.

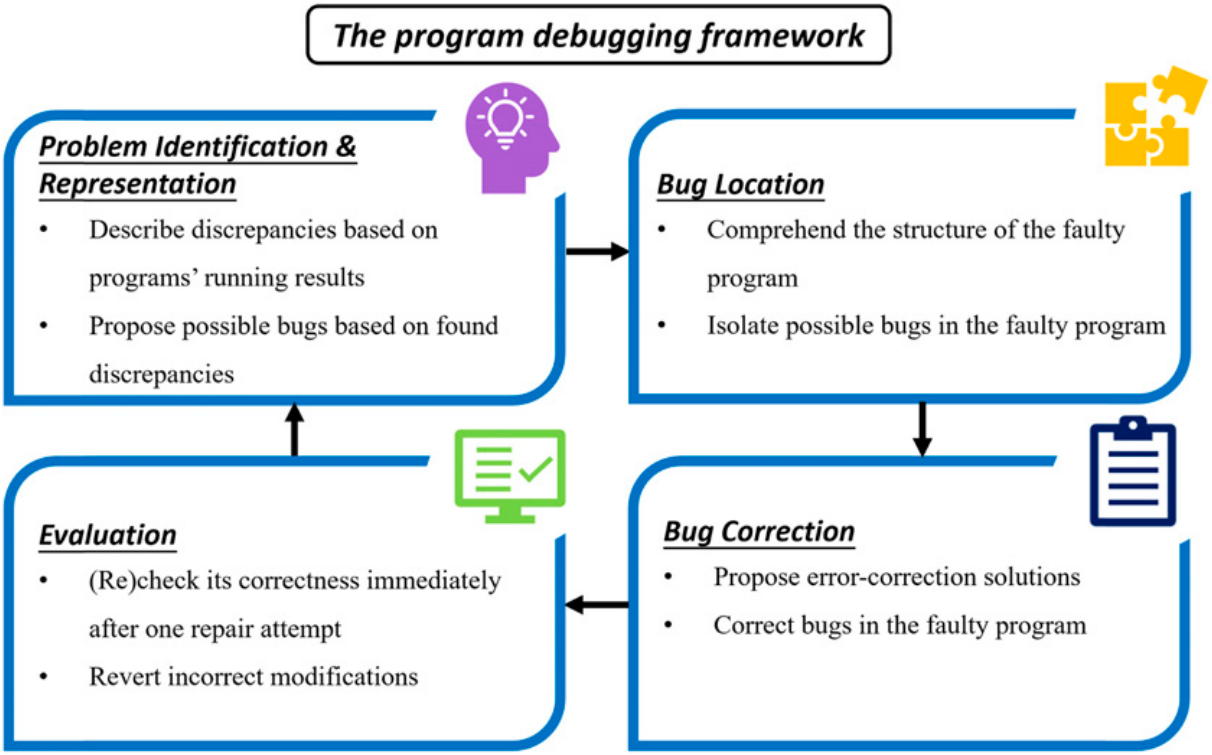


Figure 5: Schéma du processus de débogage systématique, Gao & Hew, 2023

D'autres études sur le sujet du débogage reprennent également l'idée qu'utiliser une méthode systématique de débogage et l'enseigner explicitement est bénéfique (Neutens, & Wyffels, (2021), McCauley et al., 2008, Li et al., 2019, Falloon, 2016, Zhang & Nouri, 2019) en modifiant légèrement les types de questions à se poser lors du débogage ou la forme des exercices (donner le nombre de bugs à trouver, leur type, donner des outils supplémentaires).

2.4 Types de bugs

Comme nous l'avons vu avec la méthode de débogage systématique, la classification des types de bugs permet de définir l'approche utilisée pour identifier la source du problème et la solution à appliquer. En effet, un bug d'événement déclencheur (le code ne s'exécute pas) s'abordera de manière différente qu'un problème dans l'exécution (par exemple un personnage qui ne va pas dans la bonne direction).

2.4.1 Dans la recherche

La littérature sur les types de bugs est généralement vaste. Dans notre travail, nous avons choisi de nous concentrer spécifiquement sur les recherches portant sur les types de bugs rencontrés dans Scratch et dans les études utilisant la méthode de débogage systématique. D'un langage de programmation à l'autre, ces problèmes peuvent varier considérablement. On peut les comparer aux langues classiques : bien qu'il puisse y avoir des similitudes entre les problèmes de syntaxe ou de sémantique en français et en chinois, les approches pour les identifier et les résoudre peuvent différer grandement.

De manière générale les bugs sont classifiés en erreurs syntaxiques (fautes d'orthographe ou de ponctuation), erreurs sémantiques (erreurs concernant la signification du code) et logiques (résultant d'une approche erronée ou d'une mauvaise interprétation des spécifications) et en erreurs de compilation et d'exécution. Certaines études font la différence entre les erreurs liées à la construction (dans le sens des constructions spécifiques au langage de programmation) et les erreurs non liées à la construction.

Michaeli et Romeike (2019a et b) ont utilisé des langages textuels pour leur recherche, ils relèvent que pour les novices, les bogues rencontrés étaient de nature syntaxique (par exemple, des accolades ou des types de données manquants), les erreurs d'exécution (par exemple, une initialisation manquante qui entraîne des exceptions d'exécution) ainsi que les erreurs logiques (par exemple, des appels de méthode manquants ou des directions de mouvement interchangées).

Gao et Hew (2023) ont examiné les erreurs logiques rencontrées par les étudiants dans le langage de programmation par blocs Scratch. Ils ont sélectionné différents types d'erreurs basées sur les bugs rencontrés par les étudiants lors de leurs leçons de programmation précédentes, ainsi que sur les idées fausses couramment rapportées dans la littérature. La classification suivante a été retenue : les erreurs de séquences, de boucles, de passage de

paramètres, de variables et de structures conditionnelles (en lien avec les concepts de programmation éponymes).

Une étude s'est particulièrement intéressée au sujet des types de bugs dans Scratch. Il s'agit de « Common Bugs in Scratch Programs » de Frädrich et al. (2020).

Ils relèvent trois grandes catégories de bugs dans Scratch : les bugs de syntaxe, qui s'appliquent à des langages textuels, les bugs généraux, qui peuvent se retrouver dans différents langages et les bugs spécifiques à Scratch.

Les bugs de syntaxe concernent principalement les blocs personnalisés (Scratch donne la possibilité d'éditer des blocs pour les utilisateurs les plus avancés), ils ne nous concernent donc pas étant donné que nous avons uniquement utilisé les blocs préremplis présents de base dans le programme ; les seuls blocs dans lesquels les élèves étaient invités à remplir des champs contenaient des valeurs numériques, donc non sujets à des problèmes de syntaxe, ou des textes s'affichant dans des bulles, soit sans incidence sur l'exécution du programme. Le seul cas qui pourrait potentiellement se présenter concerne le sous-type « missing termination condition » (Frädrich et al., 2020), qui survient dans le cas d'une boucle infinie qui empêche l'exécution de la suite du programme. Dans notre classification, nous l'avons rassemblée avec les erreurs de boucles.

Les bugs spécifiques à Scratch, contrairement à ce que leur nom peut suggérer, étaient également peu présents dans notre séquence et notre test. Il s'agit de bugs très spécifiques, sur les cinq sous-catégories, 3 concernent l'utilisation du « stylo » (utilisé pour tracer des lignes), extension uniquement présente dans un exercice de prolongement pour les élèves avancés et dont les bugs décrits concernent des oublis de blocs (bloc effaçant les précédents tracés, bloc définissant le début du tracé ou la fin), qui étaient dans tous les cas présents dans le programme à déboguer conçu par nos soins. Une autre sous-catégorie était spécifique aux programmes initiés par des décors qui n'apparaissent jamais, mais les élèves n'y ont jamais été confrontés dans les exercices. La dernière sous-catégorie concerne les mouvements du « sprite » (avatar utilisé), qui peuvent être saccadés en réponses aux touches du clavier (lorsqu'elles sont assignées à un mouvement), mais ce cas affecte principalement l'esthétique et l'utilisabilité, en plus d'être peu commun.

Les bugs généraux sont les plus fréquents, notamment ceux concernant les boucles (boucle infinie à l'intérieur d'une boucle), le mauvais placement de la condition de vérification ou événement dans une boucle (vérifié qu'une seule fois au lieu d'à chaque occurrence), un

mauvais rattachement de blocs rendant le code inutile. Les autres sous-types de cette catégorie faisaient soit référence à des blocs personnalisés, soit à des situations que les élèves n'allaient pas rencontrer dans notre situation.

2.4.2 Les types de bugs retenus

En procédant par élimination, nous avons écartés les bugs de compilation et d'exécution (Scratch permet d'éviter ces problèmes), de paramètres et de variables (ceux-ci étant présents dans les langages textuels) ou les blocs personnalisés pour utilisateurs avancés. Les catégories retenues des différents articles sont donc les bugs syntaxiques, sémantiques, logiques et liés/non liés à la construction (spécifique ou non au langage de programmation). Ces catégories étant assez larges, nous les avons affinées pour qu'elles correspondent aux concepts précis rencontrés dans Scratch. Nous nous sommes ainsi rapprochés des dénominations de Gao et Hew (2023). Nos types incluent les bugs de séquences, de boucles, de conditions et d'événements. En effet, dans Scratch ce qui touche à l'exécution ou non d'un programme est le bug initiant une séquence de blocs, l'événement. Le concept de séquence dans Scratch, la manière dont les blocs s'enchaînent, rejoint et inclus les erreurs sémantiques ou logiques. Enfin, les blocs et concepts restants et potentiellement problématiques sont ceux de conditions et de boucles. Swidan et al. (2018) ont analysé les idées fausses et difficultés rencontrées sur Scratch auprès de 145 participants d'âges divers (7 à 17 ans). Si l'on regarde les résultats pour nos types de bugs, il en ressort que 56,2% avaient des idées fausses sur le concept de séquence, 33,3% sur les boucles, et que les conditions étaient mieux comprises mais qu'environ 50% des participants y répondaient de manière incorrecte pour diverses raisons. L'étude incluaient également diverses erreurs de variables, mais comme nous n'avons pas couvert ce sujet avec les élèves elles ont été écartées de notre choix.

3. Problématique, hypothèses et question de recherche

La programmation informatique est une discipline complexe qui requiert non seulement la maîtrise de concepts théoriques, mais également le développement de compétences pratiques essentielles. Parmi ces compétences, le débogage occupe une place centrale, car il permet aux programmeurs d'identifier et de corriger les erreurs présentes dans leur code. Cependant, l'enseignement du débogage est souvent négligé dans les programmes scolaires, les élèves étant généralement laissés à eux-mêmes pour acquérir cette compétence.

Cette lacune pédagogique peut avoir des conséquences négatives sur l'apprentissage de la programmation, car les élèves se retrouvent confrontés à des obstacles récurrents sans disposer des outils adéquats pour les surmonter. En effet, comme nous l'avons vu le débogage nécessite une approche structurée et systématique, qui doit être enseignée de manière explicite pour permettre aux élèves de développer des stratégies efficaces de résolution de problèmes spécifiques à la programmation.

Dans le cadre de ce travail, nous formulons deux hypothèses. La première est qu'un enseignement explicite des techniques de débogage avec des élèves de 5H (8-9 ans) améliore les compétences en programmation. Cette hypothèse repose sur l'idée que l'enseignement ciblé et détaillé des méthodes de débogage peut améliorer la capacité des élèves à développer des compétences dans des contextes variés, au-delà des situations d'apprentissage initiales.

La seconde hypothèse est que les capacités de débogage des élèves dans le langage Scratch s'amélioreront, mais aussi dans d'autres langages comme VPL (Thymio), permettant d'acquérir des compétences transférables à d'autres contexte que celui dans lequel elles ont été travaillées.

La question de recherche qui découle de nos hypothèses est la suivante :

Quel est l'effet de l'enseignement explicite des techniques de débogage sur le transfert des compétences des élèves de 5H en programmation et en débogage ?

Cette question vise à explorer la relation entre l'enseignement des techniques de débogage et l'efficacité avec laquelle les élèves peuvent appliquer ces compétences dans différents contextes de programmation, ce qui est essentiel pour leur développement en tant que programmeurs compétents. Notre objectif est de mesurer l'ampleur de ces transferts.

4. Adaptation des concepts au contexte

Lors de l'élaboration du dispositif d'enseignement, nous avons été confrontés à plusieurs défis. Tout d'abord, trouver des recherches traitant de l'enseignement du débogage. Comme cité précédemment (Michaeli & Romeike, 2019b), le sujet est encore jeune et il n'existe donc pas de consensus sur la manière de s'y prendre. Ensuite, même dans les études existantes (section 2), les contextes étaient différents du notre, soit par l'âge des élèves (enfants plus âgés ou adultes), soit par le langage de programmation utilisé (BlueJ, Java, etc.), soit par le format de l'enseignement (travail à la maison, durée de plusieurs mois, utilisation de programmes personnalisé).

Malgré ces contraintes, nous avons tout de même pu adapter de nombreux contenus à notre contexte, et les études qui n'ont pas été retenues ont pu nous aider à diriger nos réflexions et questionnements.

Pour rappel, nous avons utilisé les langages de programmation par blocs Scratch ainsi que VPL (Thymio), avec deux classes d'élèves de 5H (8-9 ans), sur une période de trois semaines (14x45min.).

4.1 Les outils retenus

En nous appuyant sur les études de Gao et Hew (2023) et Michaeli et Romeike (2019 a et b), nous sommes partis d'une base impliquant un enseignement explicite du débogage comme corps de leçon, et de l'utilisation d'une technique de débogage systématique comme approche permettant aux élèves d'appliquer leurs connaissances. Les contenus, les types de bugs que l'on peut rencontrer, ont été adaptés des différentes recherches citées précédemment (section 2).

4.2 La séquence d'introduction aux concepts fondamentaux de programmation

Pour nous assurer que le niveau des élèves serait similaire, nous avons sélectionné deux classes avec des élèves n'ayant aucune expérience au préalable en programmation. Ceci impliquait qu'il fallait commencer par l'introduire aux élèves.

La première semaine, nous avons enseigné les concepts de base de la programmation sur Scratch aux deux classes en 4 leçons d'introduction de 45 minutes chacune. Les contenus sont inspirés des « computational concepts » du cadre de Brennan et Resnick (2012), concepts communs à de nombreux langages de programmation : séquences, boucles, événements, parallélisme, conditionnels et opérateurs. Étant donné le nombre de leçons limité, nous ne

pouvions pas couvrir tous les sujets en profondeur. Nous avons donc sélectionné les principaux concepts que les élèves utiliseront le plus souvent, en ajoutant ceux enseignés dans les séquences des études citées. Aucun consensus clair n'étant actuellement disponible concernant l'ordre d'enseignement des concepts sur Scratch, nous les avons organisés dans celui suggéré par Hijón-Neira et al. (2021).

La première leçon portait ainsi sur le concept de séquences avec une introduction aux événements (nécessaires pour avoir une initialisation de séquence dans Scratch), la seconde sur les conditions, la troisième sur les boucles et la dernière sur les événements. Il s'agissait ici uniquement de s'assurer que tout le monde avait des bases équivalentes et suffisantes pour pouvoir déboguer des programmes simples.

4.3 La séquence de débogage

Pour les quatre leçons (4 fois 45 minutes) suivant l'introduction à Scratch et portant sur l'enseignement explicite du débogage et des techniques systématiques, nous nous sommes concentrés sur les types de bugs afin de focaliser chaque leçon sur un concept lié à un type de bug et une technique en particulier.

Comme les types retenus concordent avec les contenus d'enseignement, nous avons adapté les formats de leçons de Gao et Hew (2023) en enlevant le travail fait hors des heures de cours (entraînement et apports théoriques) et en l'incluant dans les leçons. Nous sommes ainsi partis sur une structure similaire à l'introduction, avec le même enchaînement de concepts, mais en abordant leurs bugs relatifs.

La première leçon introduisait le concept de débogage systématique ainsi que la procédure à suivre (section 4.5). Chaque leçon avait ensuite le même format, un rappel du concept de la leçon, par exemple les conditions, de la méthode de débogage systématique, un exemple de bug en lien avec le concept (de condition dans notre exemple) ainsi que la manière de l'approcher, un exemple de bug résolu par les élèves en plenum et une série d'exercices de débogage (pour permettre de différencier) sur le concept du jour en duos.

4.4 Le processus de débogage systématique

Pour l'adaptation du modèle de débogage systématique nous avons principalement gardé ceux proposés par Gao et Hew (2023) et Michaeli et Romeike (2019). Les autres articles mentionnant le débogage systématique apportaient des réflexions sur la formulation des questionnements à

se poser durant le processus mais pas de schématisation claire et utilisable en tant que telle pour notre situation.

L'approche systématique consiste à suivre un plan structuré pour aborder les bugs. La plupart des modèles systématiques suivent une structure commune, comparable à la méthode scientifique : on se représente le résultat attendu, puis après avoir testé le programme et pris conscience des erreurs, des hypothèses sont formulées, vérifiées par des expériences et, si nécessaire, affinées jusqu'à ce que la cause de l'erreur soit trouvée. Voici les étapes retenues que nous avons adaptées par la suite :

1. **Tester et comprendre le but du programme visuellement** (Michaeli & Romeike, 2019b, Gao& Hew, 2023) :
    - Avant de plonger dans le code, il est essentiel de tester le programme et de comprendre son objectif global.
    - Cette étape permet d'identifier les problèmes évidents sans entrer dans les détails du code.
    - Dans un deuxième temps, faire des parallèles entre ce qui est observé dans le code (structure générale, sans lire toutes les lignes)
2. **Identification et représentation du problème** (Gao & Hew, 2023) :
    - Sélectionner un problème à la fois.
    - Formuler des hypothèses sur la cause du problème sélectionné.
3. **Localisation des bugs** (Gao & Hew, 2023) :
    - Identifier les parties spécifiques du code où se trouvent potentiellement les erreurs.
4. **Définition d'une stratégie de résolution** (Gao & Hew, 2023) :
    - Sur la base des hypothèses formulées, élaborer une stratégie pour résoudre le problème.
5. **Correction des bugs** (Michaeli & Romeike, 2019b, Gao& Hew, 2023) :
    - Modifier le code pour corriger l'erreur.
6. **Évaluation des solutions** (Michaeli & Romeike, 2019b, Gao& Hew, 2023) :

- Après avoir apporté des modifications, tester à nouveau le programme.
- Vérifier si les bugs ont été résolus.

7. **Répéter le processus** (Michaeli & Romeike, 2019b, Gao& Hew, 2023) :
    - Si nécessaire, revenir aux hypothèses initiales et itérer jusqu'à ce que tous les problèmes soient résolus.

Les élèves étant jeunes, nous avons simplifié les étapes pour en garder quatre :

1. Tester et observer le comportement du programme : qu'est-ce qu'il fait et qu'est-ce qu'on aimerait qu'il fasse ?
2. Observer le code (identifier un problème) : on essaie de comprendre en gros ce que les différentes parties font pour savoir dans quelle direction aller lorsqu'on aura défini le bug sur lequel on va travailler.
3. Formuler des hypothèses sur ce qui ne joue pas et essayer de corriger la partie concernée en prenant un bug à la fois
4. On évalue : est-ce que ça a marché ? Si oui on continue si non on remet le code dans son état initial et on recommence.

Ces étapes sont schématisées sous forme développée dans la figure 6 :

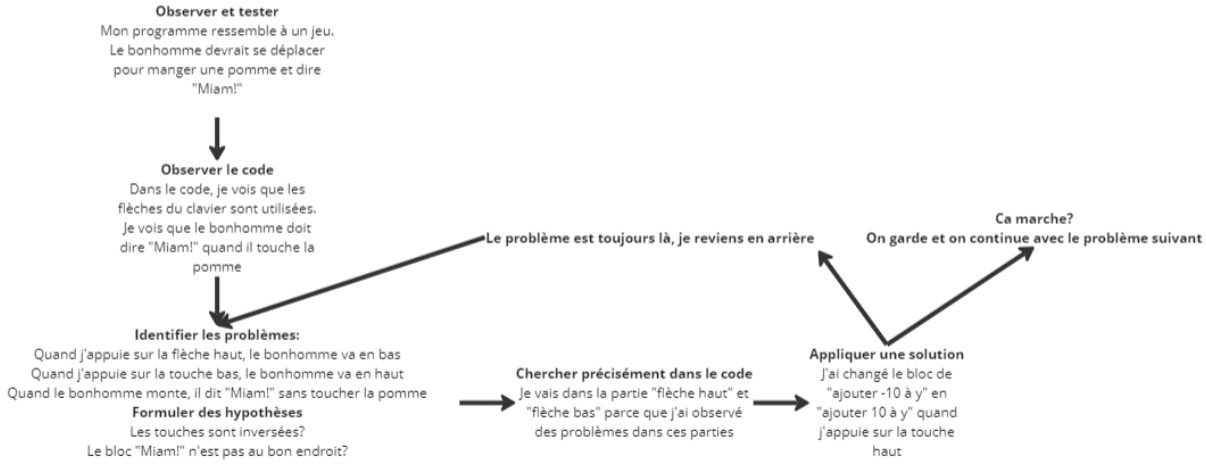


Figure 6: Schématisation développée des étapes

Comme on peut le constater, il est difficile de se limiter strictement à des étapes définies par peu de mots, car chacune d'entre elles englobe des sous-processus nécessaires à leur accomplissement. Il est essentiel d'analyser le contexte d'application pour adapter l'approche de débogage, d'où les descriptions sous forme de questionnements.

4.5 Les exercices à déboguer

Les exercices de débogage étaient à chaque fois disponible sur l'espace de classe du site Scratch ; nous les avons produits en amont pour que les élèves se concentrent uniquement sur le débogage, tout en ayant une « récompense » à la fin, un programme fonctionnel qui parfois était un jeu. Chaque exercice se déclinait en trois niveaux, le premier étant le plus simple que tous les groupes devaient résoudre, les suivants plus compliqués. La difficulté des prolongements augmentait légèrement d'une leçon à l'autre, mais l'exercice de base était à chaque fois prévu pour être accessible à tous. Nous avons suivi le même format que dans les études de Gao et Hew et Michaeli et Romeike, en limitant le nombre de bugs.

En amont, nous avons développé pour chaque type de bug les obstacles que les élèves pourraient rencontrer en y étant confrontés, les ajustements que l'on pouvait apporter au niveau des contenus enseignés, de la forme des exercices ou de la formulation des consignes pour prévenir ces obstacles ainsi que les compétences nécessaires à leur résolution (tableau récapitulatif en annexe 1).

Pour chaque exercice, nous avons ajouté des notes donnant le nombre de bugs à trouver, et dans les cas compliqués des indices sur leur emplacement (page). Le but du jeu était résumé ainsi que les contrôles de base et autres indications nécessaires pour le tester, principalement pour rendre les élèves le plus autonomes possible. La figure 7 illustre un exemple d'exercice :

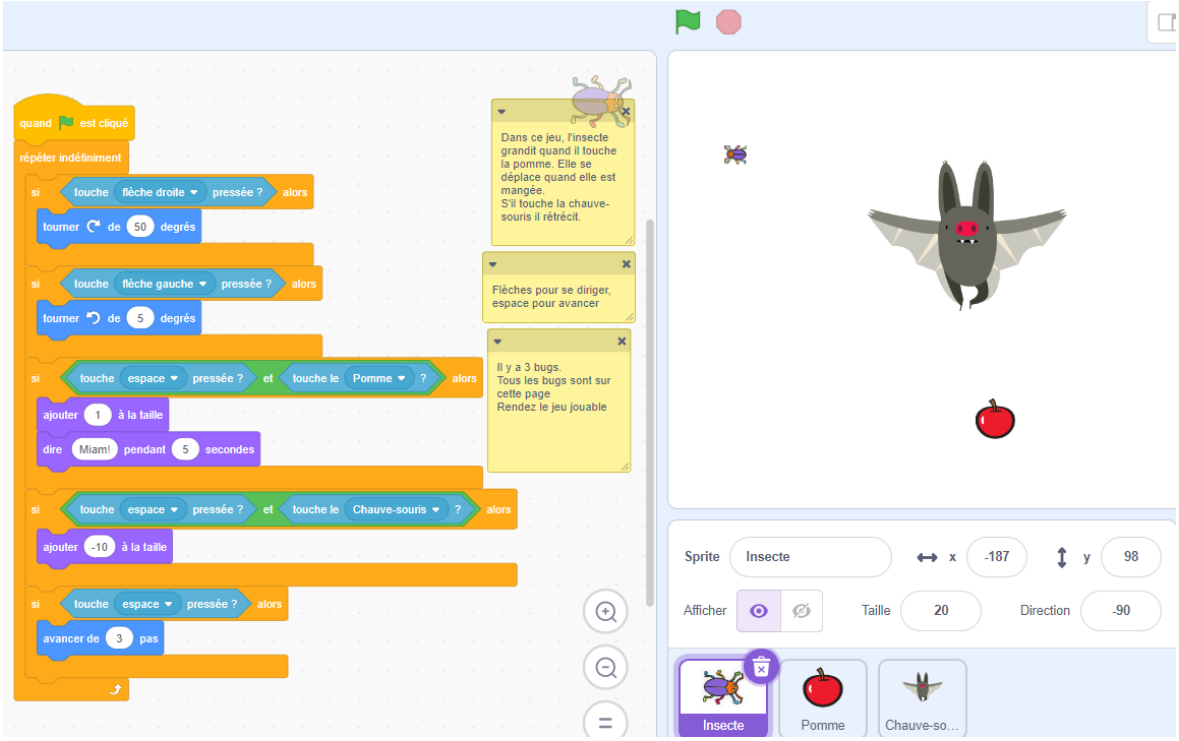


Figure 7: Exemple d'exercice à déboguer

4.6 L'introduction à Thymio VPL

Étant donné que nous voulions tester les compétences de débogage des élèves dans un autre langage, il nous fallait également l'introduire. Nous avons sélectionné un langage par blocs (VPL), suffisamment éloigné de Scratch. Thymio VPL remplit ces conditions étant donné qu'il permet de contrôler un robot, avec un langage événementiel. Comme il était difficile de se procurer et de contrôler un grand nombre de robots, nous avons opté pour la version « simulateur », permettant de se passer du robot physique.

Nous avons repris les concepts de base enseignés sur Scratch, mais comme ils étaient représentés et utilisés différemment sur VPL nous avons poussé les élèves à relever ces différences pour éviter les confusions. Nous avons également insisté sur la pratique, à l'aide d'exercices guidés avec indices pour qu'ils puissent suffisamment manipuler le langage. Le nombre de manipulations et scénarios simples possibles sur le simulateur étant limité (3 leçons de 45 min.), et le but que les élèves puissent s'en sortir lors du test de débogage, nous avons créé des exercices reprenant ceux qui seraient vus dans le test. Chaque leçon comportait une première partie théorique et de comparaison, un exercice central permettant de tester des fonctionnalités du robot et le reste du temps à disposition permettait aux élèves d'explorer librement. La première leçon était une introduction au robot et à l'interface du simulateur, avec un exercice de manipulation simple, la seconde portait sur le suivi de ligne (capteurs de sol, contrôles moteurs) et le dernier sur le suivi d'un objet (capteurs de proximité, contrôles moteurs). La figure 8 est un exemple de consigne donnée aux élèves pour a carte « circuit » disponible sur le simulateur (Fig. 12) :

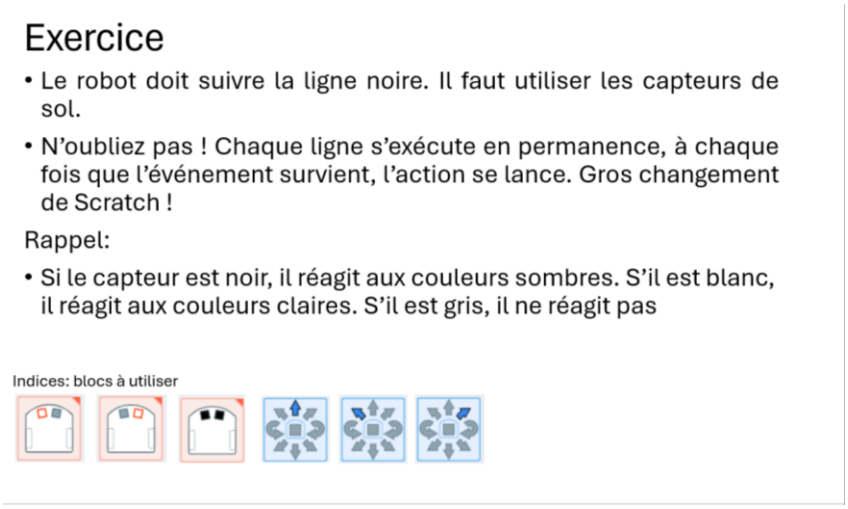


Figure 8: Exemple d'exercice sur Thymio, avec indices

5. Méthodologie

Notre recherche s'appuie sur une séquence d'enseignement structurée, spécifiquement conçue pour faciliter l'acquisition de compétences de débogage chez les élèves. Pour évaluer l'efficacité de notre intervention, nous avons utilisé divers outils pédagogiques et mis en place des mesures d'évaluation précises.

Cette section détaille la méthodologie que nous avons adoptée pour répondre à notre question de recherche selon laquelle un enseignement explicite du débogage améliore les compétences en programmation et en débogage des élèves. Notre protocole expérimental inclut des leçons interactives et des exercices pratiques (section 4.5), dont l'impact a été mesuré à l'aide de tests qui seront présentés en section 5.4.

5.1 Population et échantillon

L'étude a été menée dans deux classes de 5H (8-9 ans) du canton de Fribourg, sélectionnées car elles n'avaient pas d'expérience en programmation. L'une est composée de 16 élèves et l'autre de 20 élèves. Comme il n'était pas possible de faire un groupe contrôle et un groupe test, nous avons décidé d'élargir notre échantillon en testant notre protocole expérimental sur les deux classes. Comparer les résultats entre un petit nombre d'élèves ayant suivi l'intervention et un petit nombre d'élèves constituant le groupe témoin ne nous aurait pas permis de réaliser des comparaisons adéquates. En effet, les différences intrinsèques entre les deux classes, telles que le niveau des élèves ou le milieu, auraient pu biaiser nos résultats.

5.2 Dispositif expérimental

L'objectif du dispositif était d'entraîner les élèves à appliquer une méthode de débogage systématique, après les avoir introduits aux langages de programmation Scratch et VPL (Visual Programming Language sur Thymio, un robot éducatif).

Les élèves ont passé un test de programmation au début de la séquence (pré-test) afin d'évaluer leur niveau initial, puis des versions modifiées dans leur forme après la partie d'introduction aux langages (mid-test) et à la toute fin (post-test). De plus, des tests de débogage ont été réalisés au milieu et à la fin de la séquence. La figure 9 (page suivante) résume les étapes du protocole.

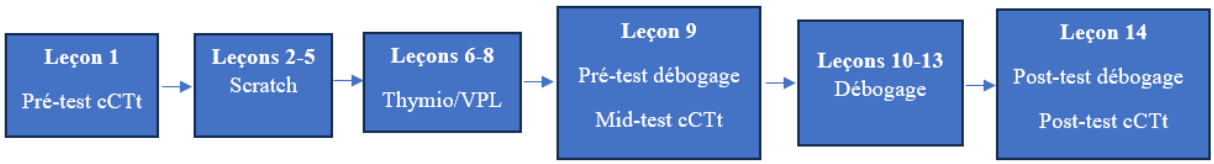


Figure 9 : Schéma du dispositif

5.3 Outils de collecte de données et procédure

Pour mesurer le niveau initial des élèves en programmation ainsi que leur progression en fonction des différentes parties, nous avons fait passer différents tests aux élèves à des moments clés.

5.3.1 Le test de programmation

Le test principal est le « competent CT test » (cCTt) développé pour mesurer les concepts de basse de la programmation auprès des élèves de 7 à 9 ans (El-Hamamsy et al., 2022). Il a été passé en pré-test, pour évaluer le niveau de départ des élèves, en mid-test, pour évaluer leur progression après les leçons d'introduction à Scratch (4 leçons) ainsi que pour connaître le niveau avant l'intervention, et en post-test pour mesurer l'évolution après les leçons de débogage (4 leçons) et comparer les résultats avec le niveau de départ des élèves (pré-test).

Leur article aborde le développement et la validation d'un test de pensée informatique déconnecté, ciblant les élèves de 7 à 9 ans. Le test présente une bonne validité apparente, de construction et de contenu, ce qui en fait un outil de choix pour ce travail.

Le test se compose de 25 questions, divisées en 7 catégories ; les 4 premières sur le concept de séquence, les 4 suivantes sur les boucles simples, 7 sur les boucles complexes, 4 sur les conditions, 4 sur les boucles « while » (tant que), et les deux dernières sur une combinaison des précédents concepts. Les exercices sont présentés sous forme de question à choix multiples, les élèves devant guider un poussin vers un objectif à l'aide d'instructions en suivant une consigne (par exemple, amener le poussin chez sa mère en ramassant une fleur sur le passage). Les instructions sont présentées sous forme de pictogrammes (Fig. 10).

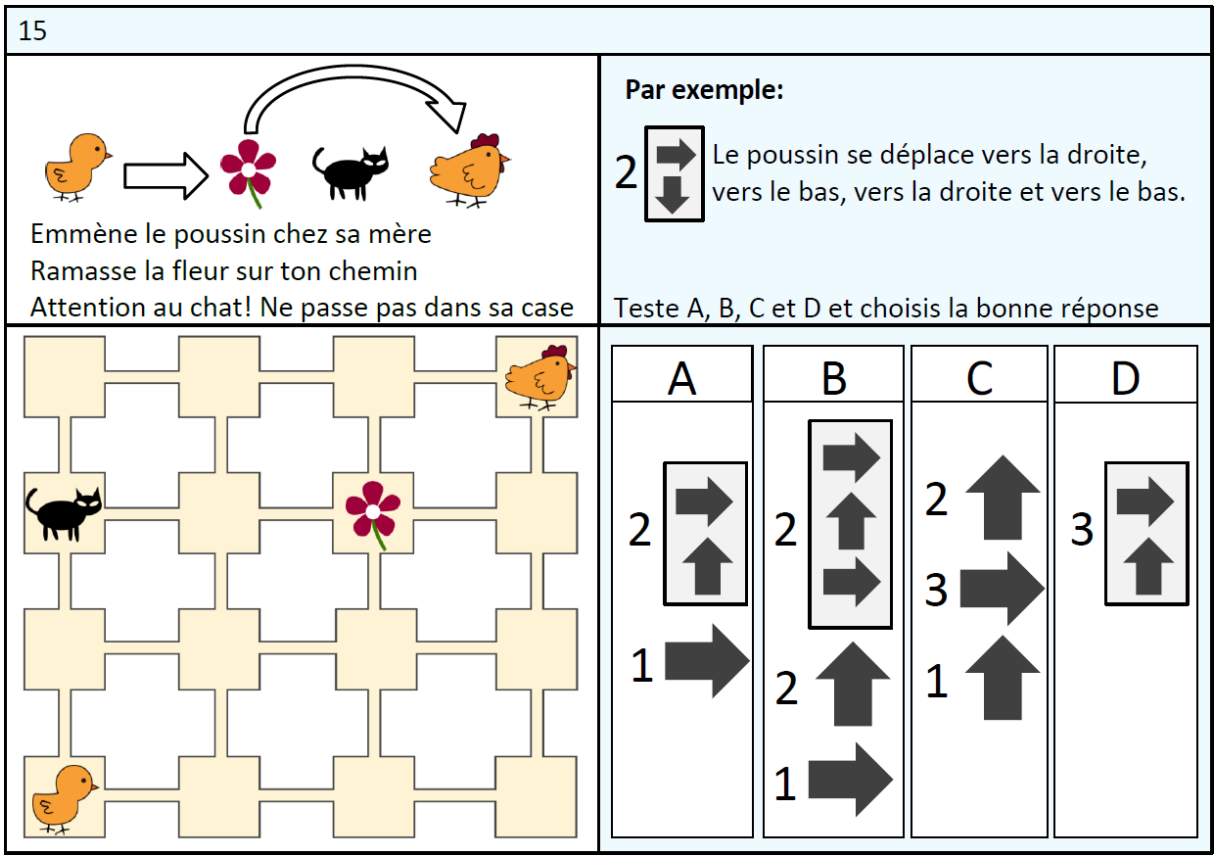


Figure 10 : Exemple de question (boucles) du cCTt (El-Hamamsy et al., 2022)

Comme les élèves devaient passer le même test trois fois, nous y avons apporté quelques modifications visuelles pour ne pas influer sur les mesures tout en rendant les questions méconnaissables. Il s'agissait d'éviter qu'ils les reconnaissent et se souviennent de réponses en se basant sur ce qu'ils ont répondu précédemment, les résultats étant influencés par leur mémoire et non par les contenus de cours

En effet, selon les travaux de Parker et al. (2022) des modifications de type cosmétique n'ont pas d'effet sur les propriétés du test et peuvent donc être utilisés pour créer des variantes isomorphes de tests.

Le premier test fourni aux élèves était l'original, pour le second nous avons remplacé les images d'illustration (un poussin, une poule, une fleur et un chat) par des images équivalentes (une abeille, une ruche, une fleur et un ours). Pour le troisième et dernier test, nous avons gardé les images de base mais avons changé l'ordre des réponses. Ainsi, les contenus des propositions A, B, C et D étaient inversées en D, C, B et A (mais en gardant l'en-tête original). Un exemple de chaque est disponible en annexe 10.

5.3.2 Les tests de débogage

En complément de ce test de programmation, nous avons voulu produire un test de débogage, pour comparer les performances des élèves avant et après la séquence éponyme. Pour ce faire, deux versions ont été produites, à nouveau en changeant les images et l'ordre des questions, des modifications n'affectant pas le degré de difficulté.

Ce test expérimental (validité psychométrique à déterminer) a été administré pour analyser la progression des élèves en débogage et évaluer si un transfert était possible hors du langage dans lequel il a été enseigné. Il se composait de questions sur Scratch, pour le transfert proche, dans lesquelles un programme était illustré par un schéma. Nous avons également projeté le programme dans sa version fonctionnelle et boguée, pour que les élèves puissent se représenter ce qui ne marche pas. Le programme contenant un bug était affiché en dessous de l'illustration, et plusieurs parties de code étaient réunies par des accolades, les élèves devant cocher celles contenant le bug. Sur une autre page, ils étaient ensuite invités à justifier leur choix. Un exemple de question est visible en figure 11 :

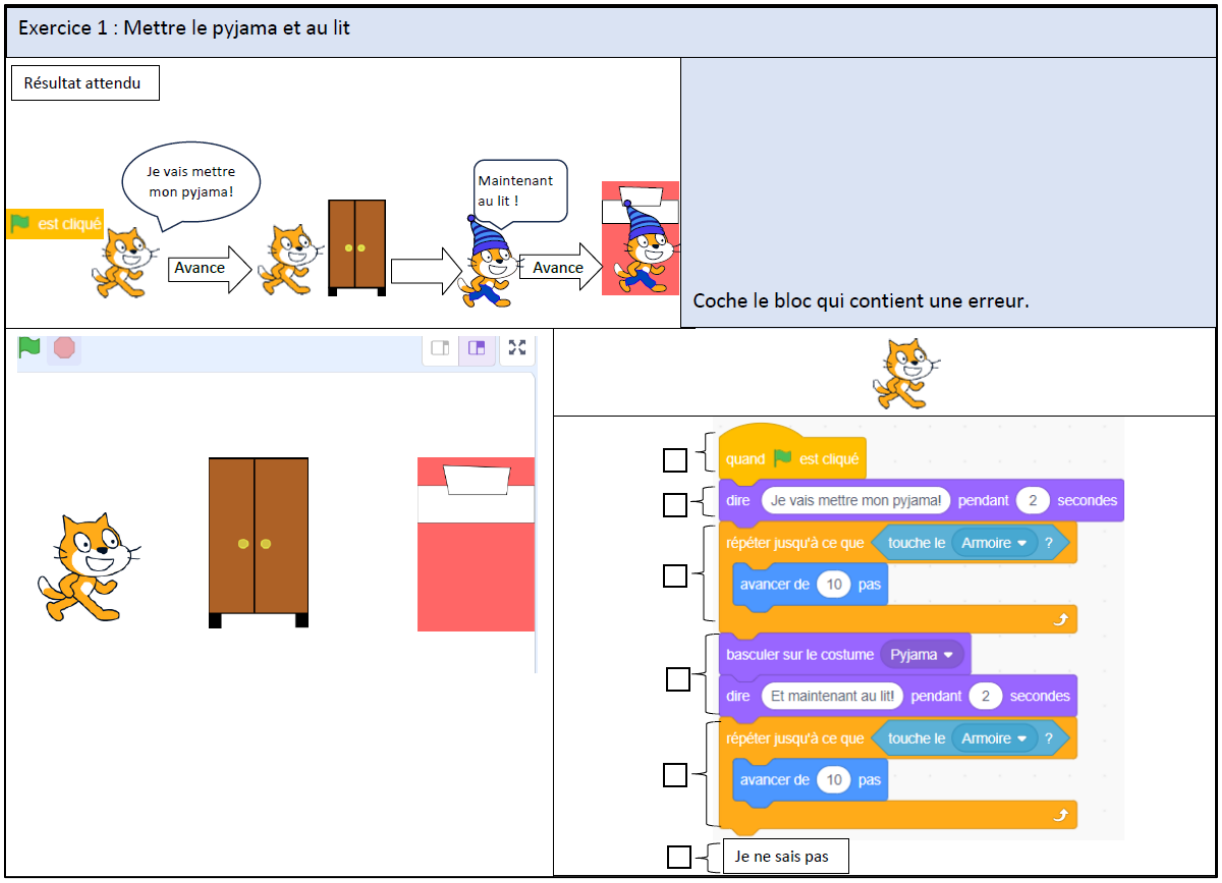


Figure 11 : Exemple d'exercice issu du test de débogage

Cette première partie du test correspond aux types d'exercices sur lesquels les élèves se sont entraînés en classe durant la séquence de débogage sur Scratch. Afin de voir s'il y avait également un transfert et une amélioration de leurs performances sur des langages plus généraux même s'ils n'y avaient pas été entraînés, nous avons ajouté une seconde partie au test. Celle-ci portait sur deux exercices similaires au test Scratch, mais avec un robot Thymio (Fig. 12) et le langage VPL.

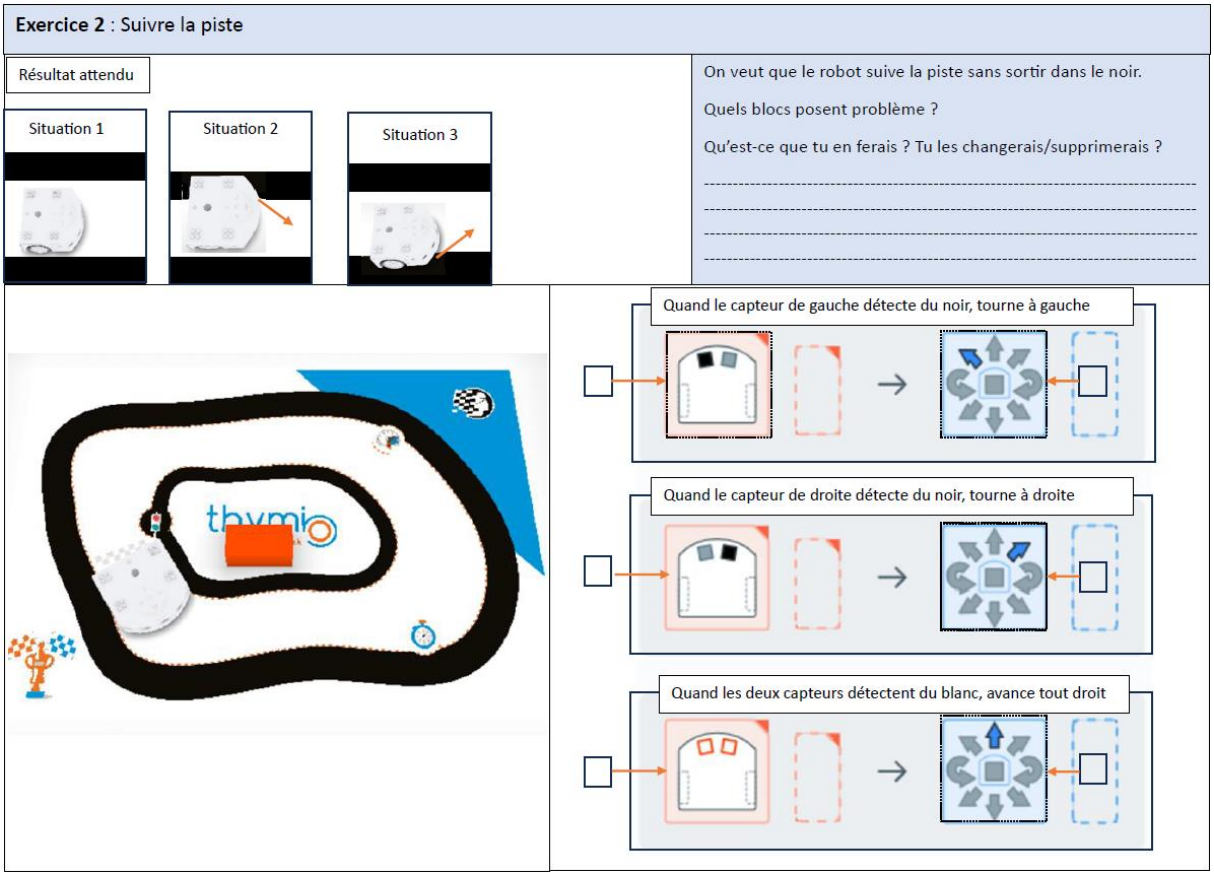


Figure 12 : Exemple d'exercice Thymio

Nous y avons ajouté trois questions modifiées du test de programmation, pour les concepts de séquence, boucle et condition (Fig. 13) ; comme le langage employé est très général, et que les élèves étaient déjà entraînés il nous a paru intéressant d'avoir un retour sur cet outil.

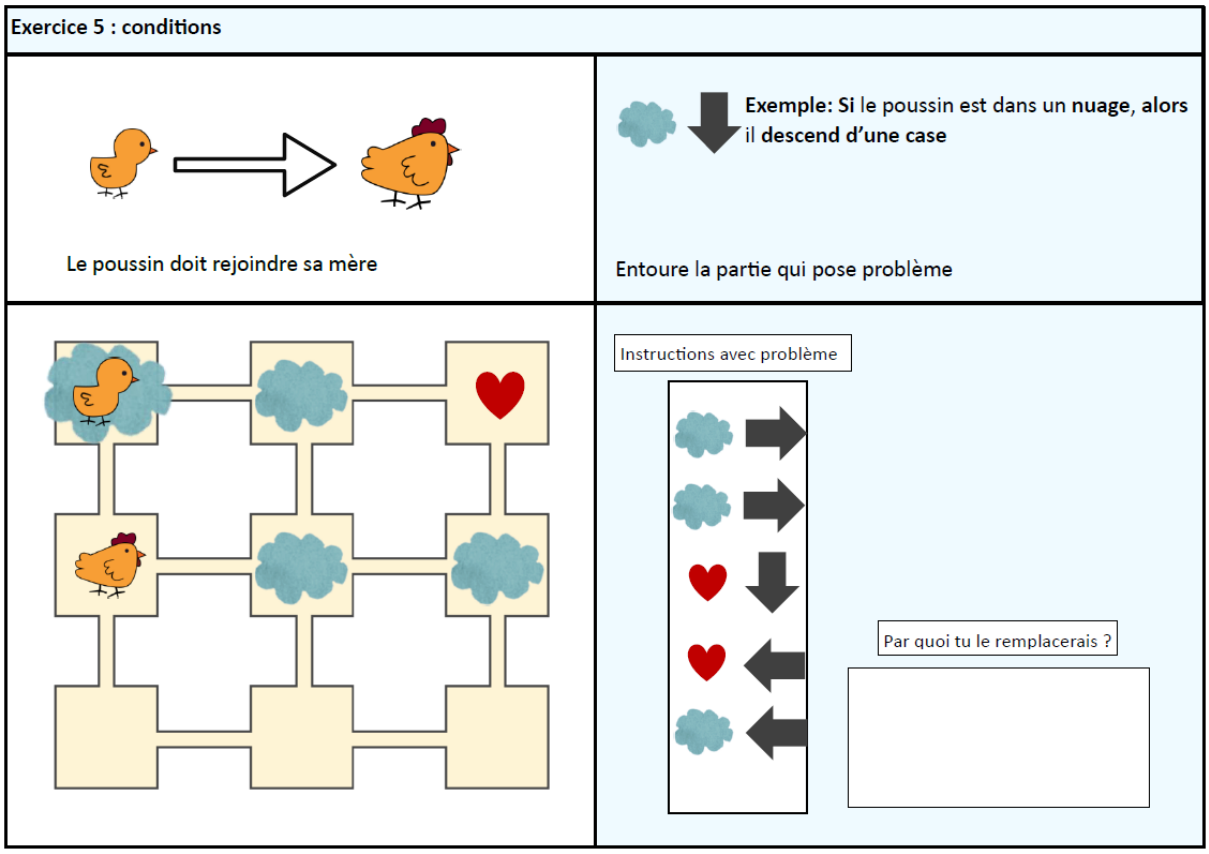


Figure 13 : Exercice adapté du test de programmation

Nous avons choisi le format débranché pour des questions d'organisation et d'accessibilité des résultats. En effet, nous ne disposions pas de suffisamment d'ordinateurs pour permettre à chaque élève de passer le test en ligne, de plus il semblait compliqué de demander aux élèves de justifier leur choix par écrit sur une feuille, ce qui leur demanderait de devoir jongler entre les deux. La gestion du matériel et les potentiels problèmes que les élèves auraient pu rencontrer, indépendants du test, nous auraient également surchargé. Devoir passer d'un ordinateur à l'autre pour s'assurer que chacun est bien connecté, qu'il reste sur la bonne fenêtre, qu'il sait sauvegarder son travail et qu'il a bien répondu à toutes les questions sont autant d'éléments qui nous auraient distraits du but principal du test et aurait potentiellement perturbé les élèves. En somme, effectuer les exercices sur des ordinateurs dans ces conditions n'aurait pas amené de bénéfices surpassant les inconvénients dans notre contexte.

5.4 Analyse des données

Les tests de programmation ont été corrigés à l'aide de la grille de correction fournie dans l'étude de El-Hamamsy et al. (2022). Chaque question était notée 1 pour juste et 0 pour faux. La grille fournit des informations supplémentaires sur les types d'erreurs commises par question, nous nous sommes toutefois concentrés sur les scores (1 ou 0) et sur les catégories de questions dans notre analyse, étant donné que nous nous intéressons sur l'évolution des performances en programmation d'une manière générale et par concept et non les types d'erreurs commises. Certaines questions n'ayant pas de réponses ont été comptées comme fausses mais représentent moins d'une dizaine de résultats sur les 3 tests de 25 questions pour les 33 élèves, et sont équitablement répartis entre les séries et les élèves, il a donc été conclu qu'ils ne pouvaient pas influer significativement sur les résultats.

Après correction des tests, il est apparu que l'un des élèves de la classe 2, qui avait un score plutôt stable (score de 11 au pré-test et 10 au mid-test), avait répondu aléatoirement au post-test, probablement pour terminer rapidement (il ne semblait pas motivé ce jour-là). Dans la classe 1, deux élèves étaient absents au pré-test. Nous avons donc retiré ces élèves des statistiques.

Pour les tests expérimentaux de débogage, le choix a été fait d'adopter une notation sur deux points. Pour les parties contenant des cases à cocher, si l'élève n'a rien répondu ou qu'il a fait un choix erroné, 0 point lui ont été attribués. Si la réponse était bonne, 2 points, et 1 point si la zone du bug ou la réflexion étaient correctes mais le choix faux, par exemple dans le cas d'une explication pertinente montrant que l'élève a bien compris mais qu'il a pu se tromper dans la lecture, ou qu'il n'a sélectionné qu'une case quand il fallait en cocher deux. Ce score intermédiaire visait à ne pas pénaliser les élèves sur la lecture/écriture, étant donné que la période de test était longue, que ceux-ci arrivaient à la fin (après environ 30 minutes de concentration sur le tes de programmation) et que les élèves étaient encore débutants en lecture/écriture. L'octroi de ce point intermédiaire se faisait également dans le contexte des questions, par exemple si un élève a visiblement oublié de cocher une case et a répondu correctement en incluant les deux cases dans sa justification (la plupart des questions des tests ne demandaient qu'une seule réponde et non deux).

La justification était notée de la même manière, mais avec une plus grande marge d'interprétation. Après avoir travaillé trois semaines avec les élèves, il était plus facile d'attribuer les scores intermédiaires, un correcteur externe ne donnerait peut-être pas les

mêmes. La partie Scratch comportait trois questions, le score maximal était 12. Pour le test Thymio, les deux premières questions portaient sur VPL (Thymio) et les trois suivantes sur les exercices de programmation adaptés. La notation était la même et le score maximal possible 20. Tous les élèves étaient présents au pré et post-test et aucune anomalie n'a été détectée, nous les avons donc tous gardés.

Nous avons veillé à anonymiser toutes les données et documents utilisés, conformément aux exigences éthiques et académiques. Nous nous référerons aux classes et aux élèves par des numéros.

Nous avons ensuite effectué plusieurs tests en fonction des échantillons comparés :

1. **Tests de différence significative (p-valeur) :**
    - En comparant les scores des élèves entre les trois séries de tests, nous pouvons déterminer si les différences observées sont statistiquement significatives. Si la p-value est inférieure à un seuil (0,05), nous pouvons rejeter H_0 et conclure qu'il existe une différence significative entre les séries de tests.
    - Nous utiliserons le test t de Student (avec échantillons appariés) pour le cCTt pour déterminer si les changements observés entre les tests sont statistiquement significatifs. Si la différence entre les moyennes est significative, cela suggère que l'intervention a eu un effet sur les performances des élèves.
    - Le test de débogage n'ayant pas une distribution normale, nous utiliserons le test des rangs signés de Wilcoxon qui est l'équivalent non-paramétrique du t de Student (avec échantillons appariés).
2. **Taille d'effet :**
    - Le d de Cohen mesure la taille de l'effet (la différence entre les moyennes des groupes) en standardisant cette différence par rapport à l'écart-type.
    - Il nous permet de quantifier l'ampleur de la différence entre les scores des élèves. Une valeur de d de Cohen plus grande indique une plus grande différence entre les groupes. Nous l'appliquerons au test t de Student.
    - Pour le test de Wilcoxon (test non paramétrique), nous utiliserons la taille de l'effet r de corrélation entre rangs bisériés.

Nous présenterons les résultats des tests sous forme de tableaux et de boîtes à moustaches pour permettre de comparer les différences entre les classes et entre les tests. Nous exposerons les données des statistiques descriptives ainsi que les valeurs clés.
Ces données seront accompagnées d'une analyse du sens de ces résultats, au regard de notre cadre théorique et de notre question de recherche.

Les variables observées sont les suivantes :

- **Variable indépendante (intervention)** :
    - L'enseignement explicite des techniques de débogage systématique sur Scratch.
- **Variables dépendantes (mesurées lors des tests)** :
    - Les scores aux tests de programmation et de débogage entre les tests.

5.5 Contrôle des tests

Pour nous assurer que les réponses pourraient être liées entre les tests, nous avons vérifié lors de leur ramassage que chaque élève avait bien noté son prénom et n'avait manqué aucune question (à leur âge il peut être difficile d'être attentif à l'ordre d'un dossier de pages imprimées recto-verso). Toutefois certains ont échappé à notre vigilance, les conditions en classe rendant une rigueur absolue très difficile, entre les questions des élèves durant le test, la nécessité de vérifier qu'ils ne collaboraient pas, le renvoi des élèves au travail lorsque des champs étaient restés vides ou lorsqu'un grand nombre de tests étaient rendus en même temps et qu'il fallait à la fois contrôler, renvoyer, et distribuer les tests suivants.

Concernant les tests de débogage, la correction a été plus qualitative concernant les justifications apportées par les élèves. Il s'agissait de juger de la pertinence d'une réponse dans un contexte, les interprétations pourraient donc varier étant donné qu'une seule personne a corrigé les tests.

6. Résultats

Nous allons à présent exposer les résultats obtenus et les mettre en relation avec les questions de recherche auxquelles ils permettent de répondre. Nous présenterons les scores des tests de programmation en premier, puis ceux des tests de débogage en deux parties (Scratch et Thymio/général). Pour les figures 14, 15, 18, 22, 23 et 24, nous avons utilisé un outil IA (Vizly, https://vizly.fyi/app) pour faciliter l'édition et la mise en page des boîtes à moustaches. Tous les graphiques et les statistiques ont été vérifiés à l'aide du programme Jamovi (https://www.jamovi.org/).

6.1 Résultats du test de programmation (cCTt)

Nous réunirons ici les résultats des deux classes, la valeur p des différences de résultats n'étant pas significative ($p>0.05$). Pour rappel, le pré-test indique le niveau des élèves avant l'introduction à la programmation, le mid-test les résultats après introduction à Scratch, et le post-test après l'intervention, en fin de séquence.

Les résultats du p de Shapiro-Wilk indiquent des valeurs de 0.113 pour le pré-test, 0.305 pour le mid-test et 0.180 pour le post-test. Elles sont supérieures à 0.05 ce qui indique une distribution normale pour la taille de l'échantillon. Nous avons également contrôlé les valeurs résiduelles à l'aide d'un tracé (Fig. 14).

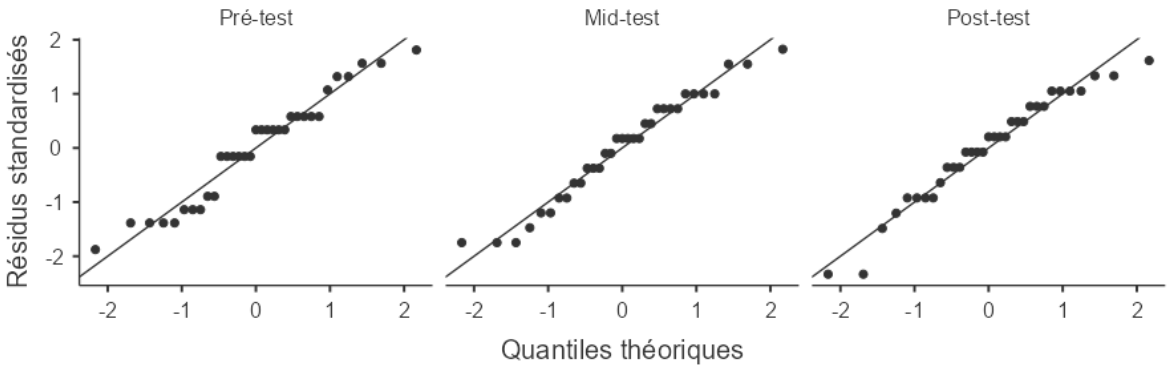


Figure 14 : Tracés résiduels pour les tests de programmation

6.1.1 Résultats généraux

La figure 15 résume les scores obtenus dans des boîtes à moustaches. Les détails des statistiques descriptives sont disponibles en annexe 2 pour les deux classes réunies mais aussi séparément.

Nous avons soumis nos résultats à plusieurs tests statistiques afin de déterminer l'ampleur des changements et répondre à notre question de recherche à l'aide de mesures conventionnelles. Pour ce faire, nous avons d'abord comparé les scores du pré-test et du post-test, soit l'ensemble de la séquence, puis entre le mid-test et le post-test, soit l'intervention. Pour le calcul de la p-valeur, nous avons utilisé un alpha standard de 0.05.

Entre le pré-test et le post-test :

- La statistique t est de 2.78 avec une p-value de 0.009, ce qui est inférieur à 0.05. Cela signifie que l'amélioration entre le pré-test et le post-test est statistiquement significative.
- Le d de Cohen est de 0.43, ce qui indique un effet modéré. Les élèves ont donc montré une amélioration notable entre ces deux tests.

Entre le mid-test et le post test :

- La statistique t est de 1.55 avec une p-valeur de 0.131, ce qui est supérieur à 0.05. L'amélioration entre le mid-test et le post-test n'est pas statistiquement significative. Il en était de même du pré-test au mid-test.
- Le d de Cohen est de 0.25, ce qui indique un effet faible. Cela indique que, bien qu'il y ait une amélioration, elle n'est pas suffisamment grande pour être considérée comme significative.

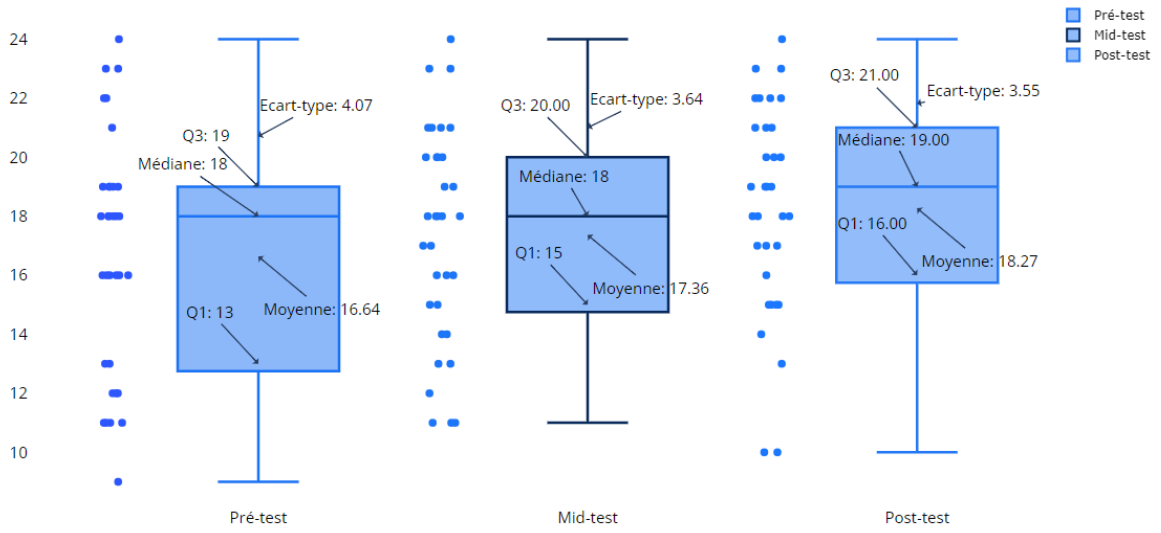


Figure 15 : Score totaux et statistiques descriptives pour les deux classes réunies, par test

6.1.2 Résultats par concept

Les questions ne portant pas toutes sur les mêmes concepts de programmation, nous avons affiné notre analyse descriptive en regroupant les résultats par catégorie de question (tableau 1 en annexes). Nous avons à nouveau réuni les données des deux classes pour la suite (n=33).

Étant donné que ces résultats n'avaient pas une répartition normale ($p<0.05$), nous avons utilisé le test de rangs signés de Wilcoxon. Des améliorations significatives ont été observées à trois endroits dans la première analyse exploratoire :

- Pour les boucles simples, du pré-test au mid-test ($p=0.047$), avec une taille d'effet élevée ($r=0.559$) et du pré-test au post-test ($p=0.025$), avec une taille d'effet élevée ($r=0.703$).
- Pour les boucles « while » du pré au post test ($p=0.007$) avec une taille d'effet élevée ($r=0.667$).

Comme nous avons multiplié les tests augmentant la probabilité d'erreurs de type 1, nous avons ensuite appliqué le test de Bonferroni. Le seuil corrigé est de 0.0167, ce qui nous montre que seules les boucles « while » ont augmenté significativement du pré au post-test.

Les autres résultats n'étaient pas significatifs mais peuvent tout de même servir d'indicateurs sur la complexité des concepts. La figure 16 (page suivante) permet de visualiser l'évolution et les différences des scores entre les concepts. Des graphiques agrandis sont disponibles en annexe 3, ainsi que les statistiques descriptives détaillées.

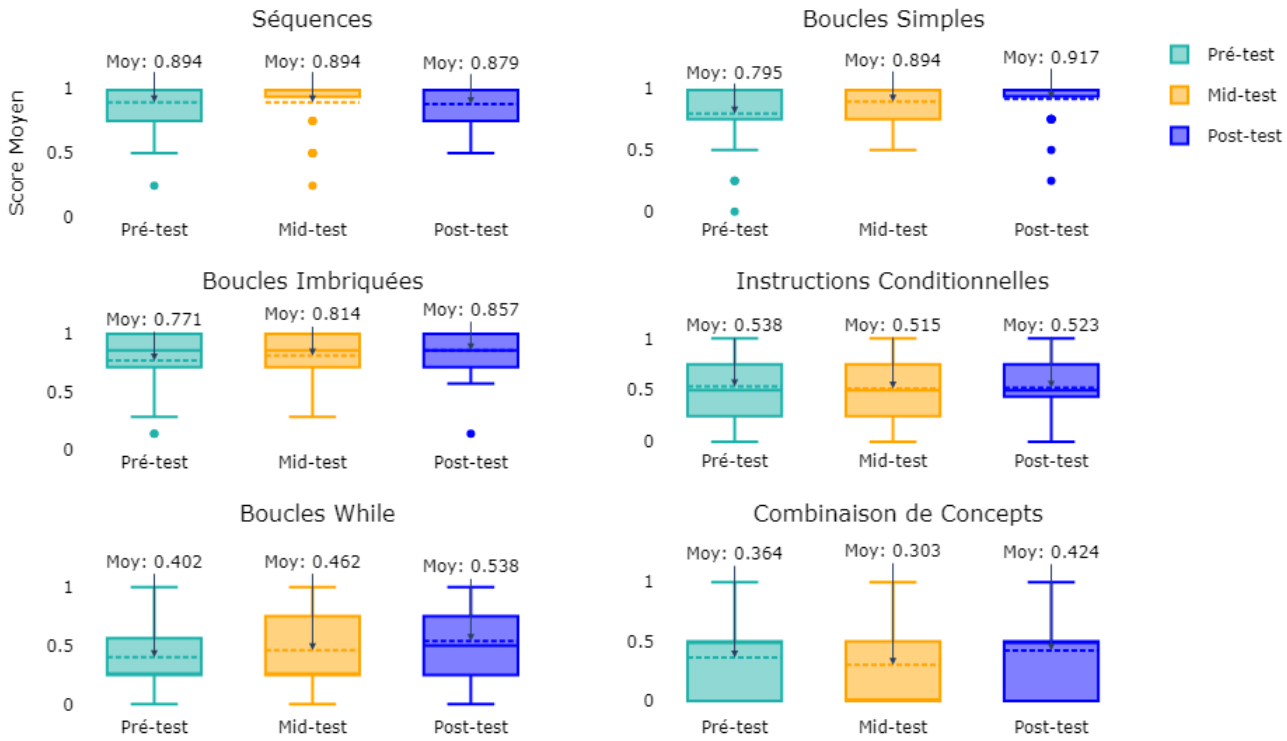


Figure 16 : Résumé des boxplot par catégorie et par test

6.2 Discussion des résultats du test de programmation (cCTt)

Étant donné que le même test a été passé trois fois, nous nous attendions à ce que les élèves montrent une plus forte amélioration après l'introduction de la programmation qu'après l'intervention. De plus, la marge d'amélioration après le mid-test était plus faible, le test permettant d'obtenir au maximum 25 points, plus les élèves s'en rapprochent moins il y a de possibilité d'observer une amélioration significative. On observe une progression significative des résultats entre le pré-test et le post-test ($p=0.009$), après l'ensemble de notre passage en classe, indiquant que la séquence dans sa globalité a permis aux élèves de s'améliorer avec un effet modéré ($d=0.43$) en programmation. Les détails ont permis de relever une amélioration significative pour les concepts de boucle simple du pré-test au mid-test et du pré-test au post-test ($p<0.05$) avec un effet élevé ($r>0.5$), ainsi que pour les boucles « while » du pré-test au post-test ($p<0.05$, $r>0.5$). Précisons toutefois que ces valeurs sont uniquement indicatives, le test est fait pour être analysé dans son ensemble et non par concept.

Nous ne pouvons pas directement comparer nos résultats aux autres études ayant étudié l'impact de l'enseignement du débogage sur les performances en programmation, étant donné l'importance des variables différentes (langage de programmation, âge des participants). Une

comparaison avec une autre étude ayant uniquement un dispositif différent serait pertinente, mais celles de Gao & Hew (2023) et de Michaeli & Romeike (2019a) qui se rapprochaient le plus de notre étude ont uniquement mesuré les compétences en débogage. Les études comparant les liens entre compétences en débogage et performances en programmation (Romero et al., 2007, Fitzgerald et al., 2008, Ahmadzadeh et al., 2005, cités dans Michaeli & Romeike, 2019b) n'ont pas testé les effets de l'implémentation d'un dispositif d'enseignement du débogage, et n'avaient pas de situation comparable (âge, langage de programmation). Ahmadzadeh et al. (2007) ont bien prolongé leur recherche pour tester les effets d'un enseignement du débogage sur les performances en programmation, mais sur des étudiants adultes et en Java. Ils n'ont toutefois pas relevé d'amélioration significative dans les performances en programmation.

En conclusion, l'enseignement de techniques de débogage systématique n'a pas eu un effet significatif sur les performances en programmation des élèves.

6.3 Résultats des tests de débogage

Cette seconde partie permettait d'évaluer si notre méthode d'enseignement du débogage permettait aux élèves de s'améliorer en débogage dans le langage enseigné et de transférer ces compétences vers d'autres langages. Le premier test porte donc sur Scratch, et le second sur VPL (Thymio) ainsi que les exercices adaptés du langage neutre du test de programmation.

Comme pour le test de programmation (cCTt), nous avons commencé par comparer les résultats des classes entre elles afin d'identifier d'éventuelles différences. La p-valeur du test de Shapiro étant inférieure à 0.05, nous avons choisi le test des rangs signés de Wilcoxon pour calculer la significativité des résultats et le test de Mann-Whitney pour comparer les classes.

6.3.1 Comparaison entre les classes

La classe 2 avait des résultats significativement plus bas en Scratch au pré-test (test de Mann-Whitney, p=0.03). Ils ont toutefois mieux performé au post-test (moyenne de 6.75 pour la classe 1 et 8.4 pour la classe 2). Comme nous avons uniquement mesuré les scores, nous n'avons pas d'explication particulière pour justifier cette différence. Nous présenterons donc les résultats du test Scratch the manière séparée pour chaque classe dans un premier temps avant de les combiner pour la conclusion. Le détail des statistiques descriptives est disponible dans le tableau 2 (page suivante).

Tableau 1 : Statistiques descriptives du pré-test Scratch pour les deux classes

Statistique	Pré-Test classe 1, n=16	Pré-Test classe 2, n=20
Moyenne	4.75	2.65
Écart-type	2.52	2.56
Q1	2.75	0.75
Médiane	4.0	2.0
Q3	5.75	4
Min	1	0
Max	10	8

Les tests de débogage sur Thymio/VPL n'ont pas montré de différence significative entre les classes (test de Mann-Whitney, $p>0.05$). Nous rassemblerons les données des deux classes pour l'analyse, mais les détails par classe sont disponibles en annexe 4 dans des tableaux.

6.3.2.1 Résultats du test Scratch pour la classe 1

La classe 1 s'est significativement améliorée du pré-test au post-test ($p=0.032$) avec un effet élevé ($r=0.657$), malgré une grande dispersion des résultats (test de Levene<0.05, $F=8.8$) au post-test (écart-type = 4.04). La figure 17 résume les statistiques descriptives et montre l'évolution des résultats :

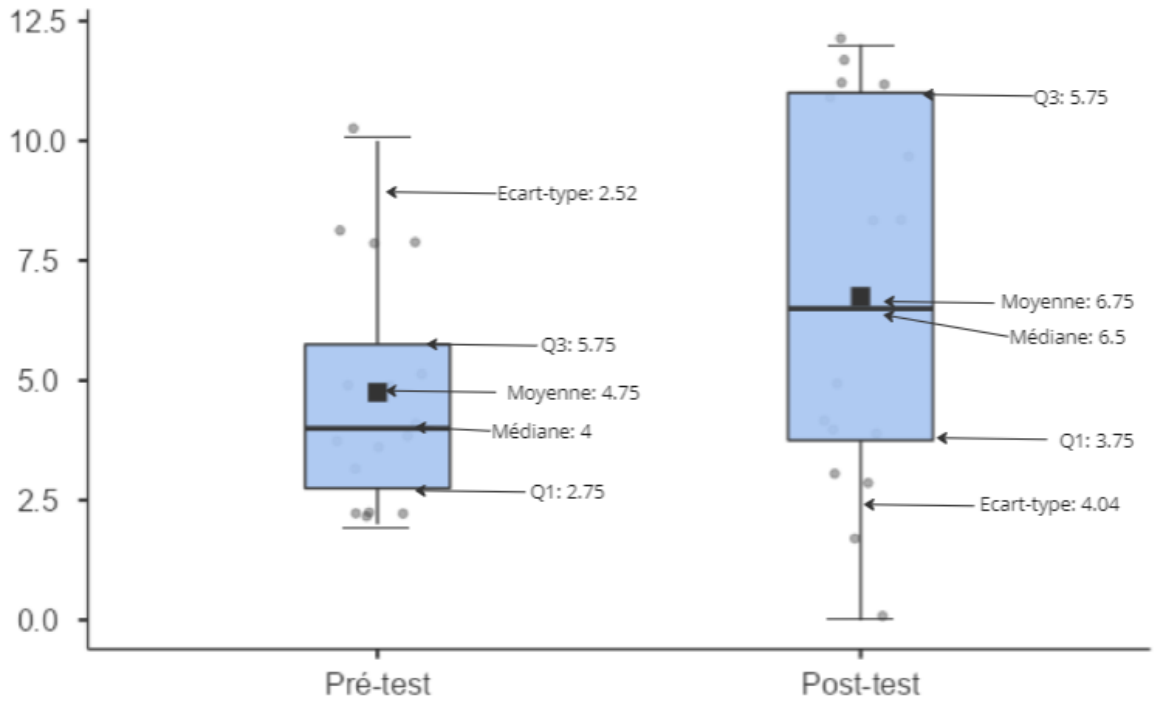


Figure 17 : Boxplot des résultats aux pré et post-test Scratch de la classe 1

6.3.2.2 Résultats du test Scratch pour la classe 2

Bien que la classe 2 soit partie avec une moyenne plus faible (2.65 contre 4.75 pour la classe 1), elle a dépassé la première après le post test (8.4 contre 6.75 pour la classe 1). L'amélioration est à nouveau significative ($p<0.001$) avec une grande taille d'effet ($r=1$). La figure 18 résume ces informations visuellement dans des boîtes à moustache.

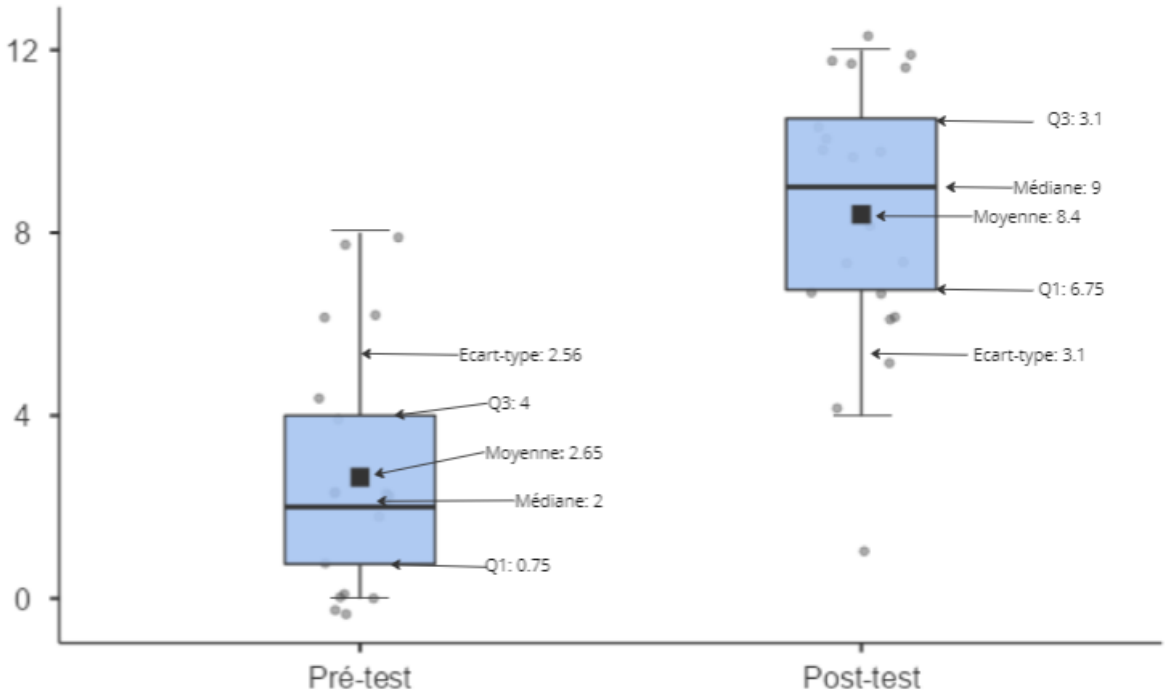


Figure 18: Boxplot des résultats au pré et post-test Scratch de la classe 2

6.3.2.3 Statistiques combinées du test Scratch

Pour généraliser les résultats sur tout l'échantillon, nous avons refait les tests pour l'ensemble des élèves. Le tableau 3 ci-dessous résume les statistiques descriptives :

Tableau 2 : Statistiques descriptives des deux classes combinées pour le test Scratch

n=36	Pré-Test	Post-Test
Moyenne	3.44	7.67
Écart-Type	2.72	3.59
Q1	2.0	4.75
Médiane	2.0	8.0
Q3	5.0	11.0
Min	0	0
Max	10	12

Les résultats combinés montrent une amélioration significative (test des rangs signés de Wilcoxon, $p<0.001$) et un grand effet ($r=0.913$).

On peut donc confirmer la première partie de notre seconde hypothèse, l'enseignement du débogage améliore bien les compétences en débogage dans le langage dans lequel les élèves l'ont travaillé.

6.3.3 Thymio et général

Comme les deux classes n'avaient pas de différence de niveau significative, nous les avons réunies pour les tests.

Les résultats montrent également une amélioration significative des compétences des élèves après l'intervention (test de Wilcoxon, $p<0.001$) avec une grande taille d'effet ($r=0.877$).

Nous avons résumé les statistiques descriptives dans la figure 19 qui aide à visualiser l'amélioration pour les deux sujets :

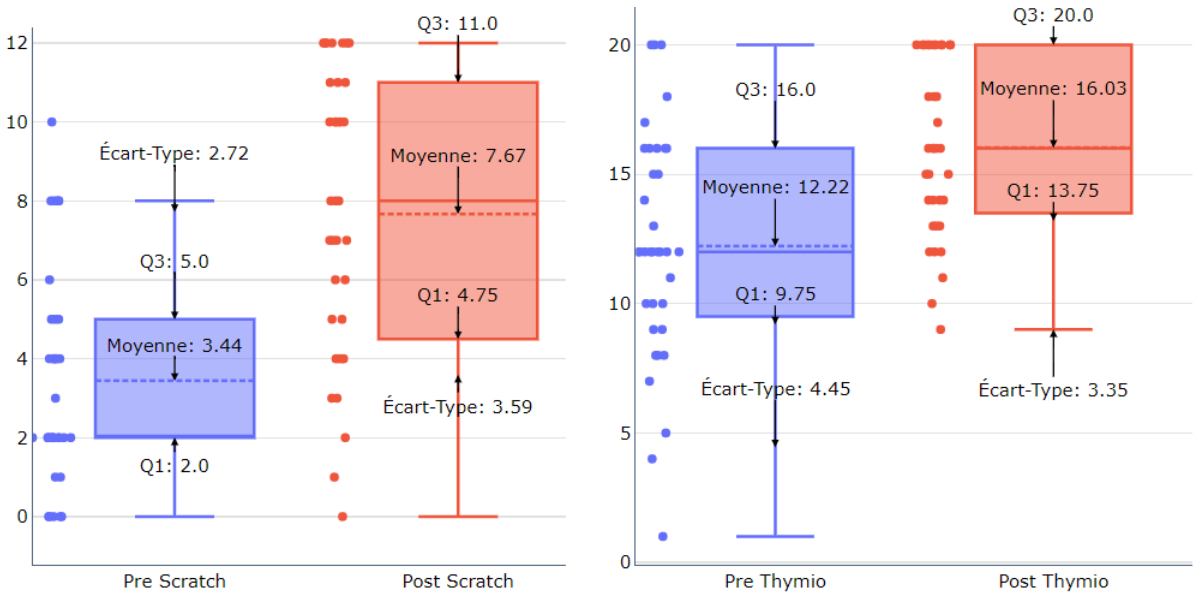


Figure 19 : Scores au pré (violet) et post test (rouge) de débogage des deux classes combinées

6.4 Discussion des résultats des tests de débogage

Comme nous l'avons vu, les scores des élèves se sont significativement améliorés ($p<0.05$) entre les pré et post tests tant pour le test Scratch que pour le test Thymio/général avec une grande taille d'effet ($r>0.5$). Les résultats du test Scratch viennent confirmer ceux obtenus par Michaeli & Romeike (2019b) et Gao & Hew (2023). Gardons toutefois à l'esprit que ces tests n'ont pas été soumis à une procédure de validation psychométrique auparavant. Ils permettent cependant d'ouvrir des pistes d'investigation. Concernant la différence observée lors des pré-tests en Scratch pour la classe 2, les élèves étaient quelque peu désarçonnés qu'on leur demande de passer un « examen » sur une matière qu'ils n'avaient pas travaillée. De fait, certains d'entre eux ont rapidement abandonné et n'ont pas essayé de trouver les solutions avec beaucoup de détermination. Ceci peut être dû à la culture scolaire, les élèves n'ayant pas l'habitude de devoir répondre à des questions pour lesquelles ils n'ont pas les réponses, ils ont souvent préféré répondre honnêtement « je ne sais pas » et passer à la suite. D'autant plus qu'ils savaient que ne pas répondre à une question n'entraînerait aucune conséquence (sanction de la note). Après discussion avec leur enseignant, il s'est avéré qu'il partageait cet avis et avait aussi observé une tendance à la « learned helplessness » (Michaeli & Romeike, 2019b) chez ses élèves. Cette théorie demanderait toutefois une autre étude pour la confirmer.

En conclusion, nous pouvons confirmer nos hypothèses, l'enseignement explicite de techniques de débogage systématique améliore bien les compétences en débogage dans le langage travaillé, et permet un transfert vers d'autres langages.

6.5 Limites

Notre échantillon reste assez faible, et nous avons utilisé un test de débogage qui mériterait d'être d'abord éprouvé avant de pouvoir officiellement valider nos résultats. De plus, nous n'avions pas de groupe témoin pour nous permettre de comparer la taille d'effet de l'intervention avec, par exemple, un groupe qui aurait continué à s'exercer en programmation. Les caractéristiques de l'échantillon (niveau moyen, âge des élèves, absence de connaissances préalables en programmation), le matériel utilisé (Scratch, Thymio, tests), la durée et la fréquence des séances d'apprentissage sont autant de variables contrôlées qui mériteraient des approfondissements. Enfin, de nombreuses variables parasites ont pu influencer les observations, comme la motivation, l'intérêt, l'attention, la fatigue, le stress, l'effet Hawthorne (induit par l'observation), la forme du travail (par deux), l'absence de réponse « je ne sais pas » au test de programmation (cCTt), et le fait que le test ait été mené sur papier sans possibilité de tester le programme alors que les élèves s'étaient entraînés sur ordinateur.

7. Conclusion

Ce travail visait à enseigner une méthode de débogage systématique à des élèves de 5H (8-9 ans), afin d'analyser son impact sur les performances en programmation, puis à évaluer si les compétences acquises permettaient une amélioration des performances en débogage et si elles étaient transférables à d'autres langages indépendamment de celui dans lequel elles avaient été travaillées.

Les résultats de la séquence pédagogique dans son ensemble ont mis en évidence une amélioration significative ($p=0.009$) des scores entre le pré-test et le post-test du cCTt pour les deux groupes en programmation avec un effet modéré ($d=0.43$). L'amélioration des scores entre le mid-test et le post-test du cCTt n'étaient toutefois pas significative ($p=0.131$), ce qui indique que l'intervention avait un effet inférieur à celui de l'introduction à la programmation sur les performances des élèves dans ce contexte, ne nous permettant pas de valider un effet de l'enseignement du débogage systématique sur les performances en programmation.

L'enseignement explicite du débogage s'est avéré déterminant pour renforcer les compétences en débogage des élèves, aussi bien dans le langage d'apprentissage (test Scratch) que dans les langages pour lesquels les élèves n'avaient pas été entraînés (test Thymio VPL et langage neutre). Dans les deux cas, l'amélioration était significative ($p<0.001$) avec un effet important ($r>0.5$).

Par rapport aux études similaires (Gao et Hew, 2023 et Michaeli et Romeike, 2019 a et b), nos résultats viennent renforcer les observations montrant que l'enseignement explicite du débogage peut avoir un impact positif sur les performances des élèves en débogage dans un même langage. La confirmation de notre hypothèse sur la transférabilité de ces compétences à d'autres langage vient ajouter des perspectives de recherche sur ce domaine.

Dans une perspective pédagogique, ces résultats soulignent l'importance d'intégrer de manière explicite l'enseignement du débogage dans les programmes scolaires d'informatique. Pour de futures études, nous pourrions imaginer un prolongement à celle menée dans ce travail en testant une méthode d'enseignement du débogage différente, afin de comparer les effets d'un changement d'approche sur les compétences en programmation et en débogage.

8. Références

Ahmadzadeh, M., Elliman, D., & Higgins, C. (2005). An analysis of patterns of debugging among novice computer science students. Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, 84-88. https://doi.org/10.1145/1067445.1067472

Ahmadzadeh, M., Elliman, D., & Higgins, C. (2007). The Impact of Improving Debugging Skill on Programming Ability. Innovation in Teaching and Learning in Information and Computer Sciences, 6(4), 72–87. https://doi.org/10.11120/ital.2007.06040072

Audrin, C., Capron Puozzo, I., (2023, semestre d'automne). Séminaire de mémoire (MD303 – séminaire de mémoire) [Slides PowerPoint]. Teams.

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. American Educational Research Association Meeting, Vancouver, BC, Canada, 1–25. https://www.media.mit.edu/publications/new-frameworks-for-studying-and-assessing-the-devel opment-of-computational-thinking/

Carver, S., & Risinger, S. (1987). Improving children's debugging skills. Journal of Educational Computing Research, 3(2), 147-171.

Chiu, C.-F., & Huang, H.-Y. (2015). Guided Debugging Practices of Game Based Programming for Novice Programmers. International Journal of Information and Education Technology, 5(5), 343-347. https://doi.org/10.7763/IJIET.2015.V5.527

El-Hamamsy, L., Zapata-Cáceres, M., Barroso, E. M., Mondada, F., Zufferey, J. D., & Bruno, B. (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), 1818-1866. https://doi.org/10.1177/07356331221081753

Falloon, G. (2016). An analysis of young students' thinking when completing basic coding tasks using Scratch Jnr. On the iPad. *Journal of Computer Assisted Learning*, *32*(6), 576-593. https://doi.org/10.1111/jcal.12155

Fitzgerald, S., Lewandowski, G., McCauley, R., Murphy, L., Simon, B., Thomas, L., & Zander, C. (2008). Debugging : Finding, fixing and flailing, a multi-institutional study of novice debuggers. Computer Science Education, 18(2), 93-116.

Frädrich, C., Obermüller, F., Körber, N., Heuer, U., & Fraser, G. (2020). Common Bugs in Scratch Programs. *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, 89-95. https://doi.org/10.1145/3341525.3387389

Gao, X., & Hew, K. F. (2023). A Flipped Systematic Debugging Approach to Enhance Elementary Students Program Debugging Performance and Optimize Cognitive Load. Journal of Educational Computing Research, 61(5), 1064-1095. https://doi.org/10.1177/07356331221133560

Heikkilä, M., & Mannila, L. (2018). Debugging in Programming as a Multimodal Practice in Early Childhood Education Settings. Multimodal Technologies and Interaction, 2(3), 42. https://doi.org/10.3390/mti2030042

Lahtinen, E., Ala-Mutka, K., & Järvinen, H.-M. (2005). A study of the difficulties of novice programmers. Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, 14-18. https://doi.org/10.1145/1067445.1067453

Lee, M. J., Bahmani, F., Kwan, I., LaFerte, J., Charters, P., Horvath, A., Luor, F., Cao, J., Law, C., Beswetherick, M., Long, S., Burnett, M., & Ko, A. J. (2014). Principles of a debugging-first puzzle game for computing education. 2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 57-64. https://doi.org/10.1109/VLHCC.2014.6883023

Li, C., Chan, E., Denny, P., Luxton-Reilly, A., & Tempero, E. (2019). Towards a Framework for Teaching Debugging. *Proceedings of the Twenty-First Australasian Computing Education Conference*, 79-86. https://doi.org/10.1145/3286960.3286970

Liu, Z., Zhi, R., Hicks, A., & Barnes, T. (2017). Understanding problem solving behavior of 6–8 graders in a debugging game. Computer Science Education, 27(1), 1-29. https://doi.org/10.1080/08993408.2017.1308651

McCauley, R., Fitzgerald, S., Lewandowski, G., Murphy, L., Simon, B., Thomas, L., & Zander, C. (2008). Debugging : A review of the literature from an educational perspective. *Computer Science Education*, *18*(2), 67-92. https://doi.org/10.1080/08993400802114581

Michaeli, T., & Romeike, R. (2019a). Improving Debugging Skills in the Classroom : The Effects of Teaching a Systematic Debugging Process. Proceedings of the 14th Workshop in Primary and Secondary Computing Education, 1-7. https://doi.org/10.1145/3361721.3361724

Michaeli, T., & Romeike, R. (2019b). Current Status and Perspectives of Debugging in the K12 Classroom : A Qualitative Study. 2019 IEEE Global Engineering Education Conference (EDUCON), 1030-1038. https://doi.org/10.1109/EDUCON.2019.8725282

Neutens, T., & Wyffels, F. (2020). Analyzing coding behaviour of novice programmers in different instructional settings : Creating vs. Debugging. 2020 International Conference on Computational Science and Computational Intelligence (CSCI), 892-897. https://doi.org/10.1109/CSCI51800.2020.00167

Parker, M. C., Garcia, L., Kao, Y. S., Franklin, D., Krause, S., & Warschauer, M. (2022). A Pair of ACES : An Analysis of Isomorphic Questions on an Elementary Computing Assessment. *Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 1*, 2-14. https://doi.org/10.1145/3501385.3543979

Perkins, D. N., & Salomon, G. (2012). Transfer of learning. International Encyclopedia of Education (Third Edition), 654-660.

Sweller, J., Ayres, P., Kalyuga, S. (2011). Cognitive Load Theory. Springer.

Zhang, L., & Nouri, J. (2019). A systematic review of learning computational thinking through Scratch in K-9. Computers & Education, 141, 103607. https://doi.org/10.1016/j.compedu.2019.103607

Zúñiga Muñoz, R. F., Mejía Córdoba, I. C., Salazar España, B. G., Tenorio Melenje, M., Trujillo Medina, M. A., & Hurtado Alegría, J. A. (2023). Adjusting the ChildProgramming Methodology to Educational Robotics Teaching and Debugging. Education Sciences, 13(9), 936. https://doi.org/10.3390/educsci13090936

9. Annexes

Annexe 1: tableau récapitulatif de l'analyse à priori des types de bugs

Types de bugs	**Problématiques possibles**	**Prévenir le problème**	**Compétences**
Tous	Un bug s'inscrit dans du code et n'est pas isolé des autres concepts -Si trop de code nouveau inconnu -Si un concept n'est pas maîtrisé, par exemple une erreur de boucle mais dans une séquence, si le concept de séquence n'est pas maîtrisé -Se retrouver bloqué et ne pas savoir s'en sortir -Introduire des nouvelles erreurs en essayant d'en corriger d'autres -Les élèves plus faibles ont tendance à procéder par essai-erreur ce qui est peu efficace -Les élèves ne savent pas qu'il existe différents types d'erreur qui s'abordent différemment -Surcharge cognitive -Bugs trop nombreux -Utilisation de code pour différents avatars, interactions entre blocs et entre sprites	-Utiliser du code connu, construire sur du code connu -Utiliser l'approche systématique -Enseigner la méthode et s'assurer qu'elle soit acquise, distinguer les types de bugs **-Limiter le nombre et le type de bugs pour que ça reste crédible** **-Si différents avatars utilisent du code, le spécifier explicitement**	-Décomposer les différentes parties d'un programme -Maîtrise des concepts de base -Maîtrise de l'approche systématique et application -Familiarité avec l'IDE -Être capable d'identifier des liens de causalité -Distinguer des éléments
Séquence	-Événements séparés, avoir plusieurs ensembles de blocs peut troubler les élèves s'il y a plusieurs éléments simultanés -Accumulation de problèmes dans la séquence ; au-delà d'un certain nombre d'erreurs dans la séquence tout l'ensemble devient un grand bug	**-Ne pas dépasser un seuil de blocs mal placés par séquence** -Mieux vaut enseigner la simultanéité et	-Maîtriser le concept de séquence -Maîtriser le concept de simultanéité

	bon à jeter ; p.ex. si j'ai un événement suivi de 10 blocs, que j'inverse les 5 du haut et les 5 du bas puis mixe l'intérieur de ces sous-ensembles, ce n'est plus un bug mais une lacune de connaissances à la construction. Les bugs doivent rester réalistes -Simultanéité comme problématique principale	déplacer un bloc par événement, max. un ensemble de blocs s'il s'agit de boucles.	-Identifier l'étape qui ne s'exécute pas convenablement dans une suite de blocs
Boucles	-Si le concept de séquence n'est pas acquis celui de boucle peut l'être aussi -Si une condition de sortie pose problème, l'élève peut chercher du côté de ce qui la précède ou la suit au lieu de penser à vérifier la condition de sortie -Aussi au niveau d'une boucle infinie dans une boucle finie, l'identification de la sortie requiert de retracer le parcours de la boucle -Surcharge cognitive si des boucles sont imbriquées dans un code déjà compliqué	-Enseignement explicite des conditions de sortie pour que les élèves y soient suffisamment familiarisés pour envisager une erreur à ce niveau et contrôler systématiquement **-S'assurer de ne pas empiler les erreurs**, notamment avec les concepts compliqués ; si je veux un bug de boucles imbriquées, ne mettre que celui-ci	-Comprendre qu'une boucle est une répétition de séquence. -Identifier et comprendre les conditions de sortie d'une boucle
Conditions	-La condition ne survient qu'une fois et est invisible, par exemple un "si la touche espace est pressée" qui devrait être vérifiée dans un délai trop court -Confusion entre condition et événement -Problèmes de compréhension de séquence, ne pas savoir où la condition doit se situer même quand ça ne fait pas de différence et rester bloqué ou perdre du temps, ou penser que la solution était de déplacer la fonction -La condition se déclenche dans une situation spécifique qui nécessite de tester le programme en profondeur pour être apparente; p.ex. le chat dit	-Répéter à chaque leçon la différence entre un événement et une condition pour s'assurer que tous la maîtrisent -S'assurer d'utiliser les conditions dans des situations où soit les élèves comprennent l'instantanéité de la vérification de la condition, soit ce n'est pas nécessaire -Faire tester explicitement aux élèves différentes configuration, faire bouger les conditions	-Maîtrise des différences entre événement et condition -Maîtrise du concept de vérification de condition, aussi temporellement -Connaissance des situations traîtres

	miam quand il touche la pomme, le chat est en face de la pomme et l'erreur est dans la partie "si flèche gauche est pressée" qui n'arrive que rarement -Chevauchement de conditions, p.ex. une erreur dans si flèche haut est pressée et une dans si flèche droite est pressée, l'élève peut identifier l'un des deux mais ne pas voir l'autre car elles se sont déclenchées par hasard au même moment et la configuration de la seconde erreur ne réapparaît plus -Cas de si/alors/sinon ou les deux situation devraient se déclencher, comprendre l'exclusion de la seconde condition	**-Indiquer le nombre d'erreurs à trouver** -S'assurer de la bonne application de la méthode systématique, montrer explicitement pourquoi on l'utilise avec des situations qui pourraient empêcher l'élève de trouver l'erreur s'il s'y prend à tâtons ou sort de l'approche systématique -Entrainement suffisant en test de code pour que les élèves se méfient des bugs situationnels	
Événements	-Confusion entre événement et condition -Empilement de bugs -Habitude d'utiliser des événement tout en haut du code et invisibilisation vu que les problèmes se situent souvent dans le reste de la séquence -Utilisation d'événements simultanés, l'un contient un bug l'autre pas, risque de perte de temps sur la partie non boguée -Condition dans un événement qui s'opposent -Mix entre des événements classiques et l'événement du drapeau vert qui ne s'exécute qu'une fois -Événements concurrents entre différents sprites	-Enseigner explicitement les particularités du drapeau vert par rapport aux autres événements pour éviter que les élèves ne croient qu'il s'agisse d'un drapeau "start" et qu'il ne peut pas inclure d'erreurs ; expliquer qu'on peut aussi s'en passer **-Ne pas utiliser de bugs qui n'ont pas été rencontrés en cours, par exemple dans le cas d'événements concurrents**	-Maîtriser la différence entre événement et condition -Tester le programme systématiquement, y-compris en allant regarder si différents sprites utilisent du code -Maîtriser la notion de programme, qui inclut des ensembles de code pouvant s'exécuter simultanément

Annexe 2 : statistiques descriptives du test de programmation

Tableau 3 : Moyenne des scores par catégorie de questions. En rouge, diminution, en vert, augmentation

n=33	Moyenne pré-test	Moyenne mid-test	Moyenne post-test
Séquences	0.894	0.894	0.879
Boucles simples	0.795	0.894	0.917
Boucles imbriquées	0.771	0.814	0.857
Instructions conditionnelles	0.538	0.515	0.523
Boucles while	0.402	0.462	0.538
Combinaison de concepts	0.364	0.303	0.424

n=33	**Pré-test**	**Mid-test**	**Post-test**
Minimum	9	11	10
Maximum	24	24	24
Premier Quartile (Q1)	13	15	16
Médiane	18	18	19
Troisième Quartile (Q3)	19	20	21
Moyenne	16.64	17.36	18.27
Écart-type	4.07	3.64	3.55

Figure 19 : Statistiques descriptives pour les deux classes. En rouge, diminution, en vert, augmentation.

n=18	**Pré-test**	**Mid-test**	**Post-test**
Minimum	9	11	10
Maximum	24	24	24
Premier Quartile (Q1)	13.75	15	16.25
Médiane	17	17	18
Troisième Quartile (Q3)	20.5	19.75	20.5
Moyenne	17.17	17	18.06
Écart-type	4.36	4.04	3.52

Figure 20 : Statistiques descriptives pour la classe 1, du pré-test au post-test. En rouge, diminution, en vert, augmentation.

n=15	**Pré-test**	**Mid-test**	**Post-test**
Minimum	11	13	10
Maximum	23	23	23
Premier Quartile (Q1)	12	15	16
Médiane	18	18	20
Troisième Quartile (Q3)	18	20	21.5
Moyenne	16	17.8	18.53
Écart-type	3.74	3.17	3.68

Figure 21 : Statistiques descriptives pour la classe 2, du pré-test au post-test. En rouge, diminution, en vert, augmentation.

Annexe 3 : statistiques du test de programmation par concept

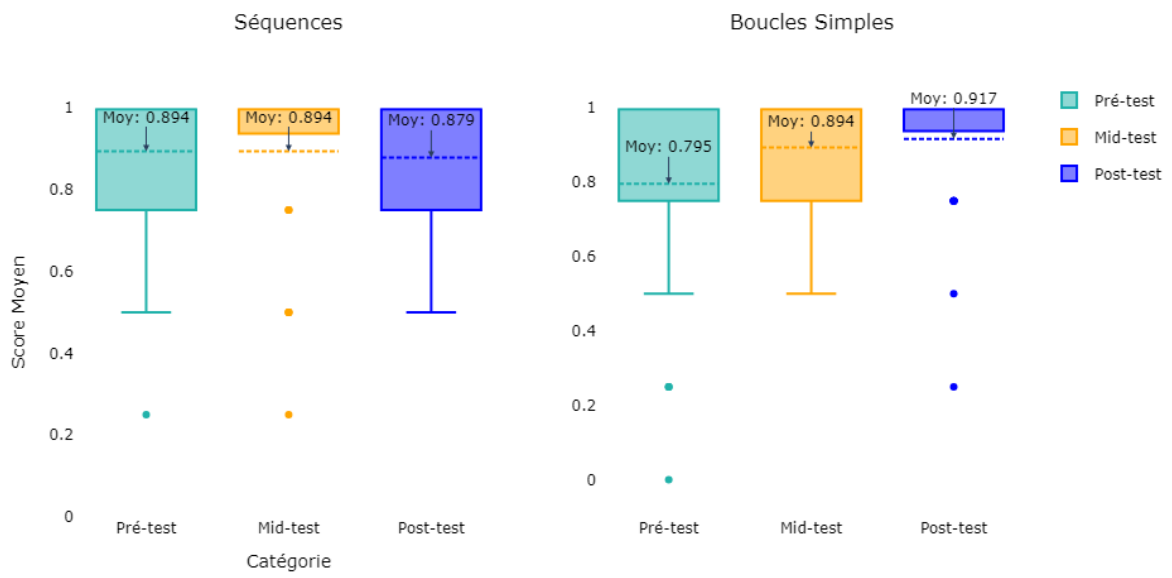


Figure 22 : Boxplot des scores par pour les séquences et les boucles simples par test

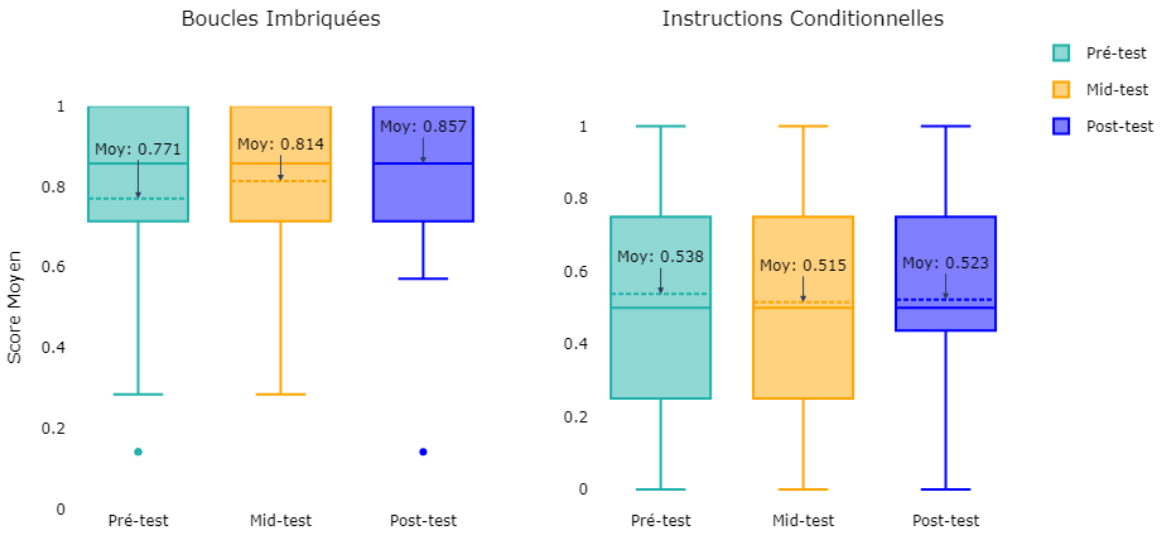


Figure 23 : Boxplot des scores pour les boucles imbriquées et les instructions conditionnelles par test

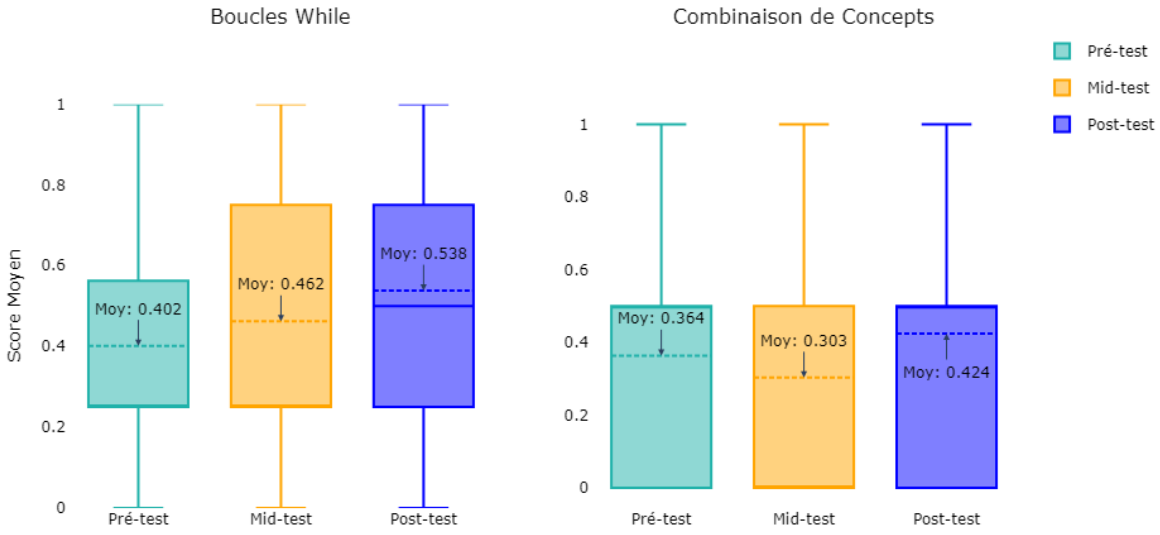


Figure 24 : Boxplot des scores pour les boucles « while » et les combinaisons de concepts par test

Statistiques descriptives

	Séquence pré-test	Séquence mid-test	Séquence post-test	Boucles simples pré-test	Boucles simples mid-test	Boucles simples post-test
Moyenne	0.894	0.894	0.879	0.795	0.894	0.917
Médiane	1.00	1.00	1.00	1.00	1.00	1.00
Ecart-type	0.188	0.208	0.141	0.283	0.153	0.173
Ecart interquartile	0.250	0.00	0.250	0.250	0.250	0.00
Minimum	0.250	0.250	0.500	0.00	0.500	0.250
Maximum	1.00	1.00	1.00	1.00	1.00	1.00

Figure 25 : Statistiques descriptives par test et par concept

Statistiques descriptives

	Boucles imbriquées pré-test	Boucles imbriquées mid-test	Boucles imbriquées post-test	Conditions pré-test	Conditions mid-test	Conditions post-test
Moyenne	0.771	0.814	0.857	0.538	0.515	0.523
Médiane	0.857	0.857	0.857	0.500	0.500	0.500
Ecart-type	0.247	0.236	0.179	0.307	0.299	0.282
Ecart interquartile	0.286	0.286	0.286	0.500	0.500	0.250
Minimum	0.143	0.286	0.143	0.00	0.00	0.00
Maximum	1.00	1.00	1.00	1.00	1.00	1.00

Figure 26 : Statistiques descriptives par test et par concept

Statistiques descriptives

	While pré-test	While mid-test	While post-test	Combinaison pré-test	Combinaison mid-test	Conditions post-test
Moyenne	0.402	0.462	0.538	0.364	0.303	0.523
Médiane	0.250	0.250	0.500	0.500	0.00	0.500
Ecart-type	0.265	0.343	0.349	0.287	0.394	0.282
Ecart interquartile	0.250	0.500	0.500	0.500	0.500	0.250
Minimum	0.00	0.00	0.00	0.00	0.00	0.00
Maximum	1.00	1.00	1.00	1.00	1.00	1.00

Figure 27 : Statistiques descriptives par test et par concept

Annexe 4 : statistiques des tests de débogage

Thymio et général – classe 1

n=20	Pré-Test	Post-Test
Moyenne	11.7	15.05
Écart-type	4.58	3.32
Q1	9.75	12.75
Médiane	12.0	14.5
Q3	14.25	17.0
Min	1	10
Max	20	20

Thymio et général - classe 2

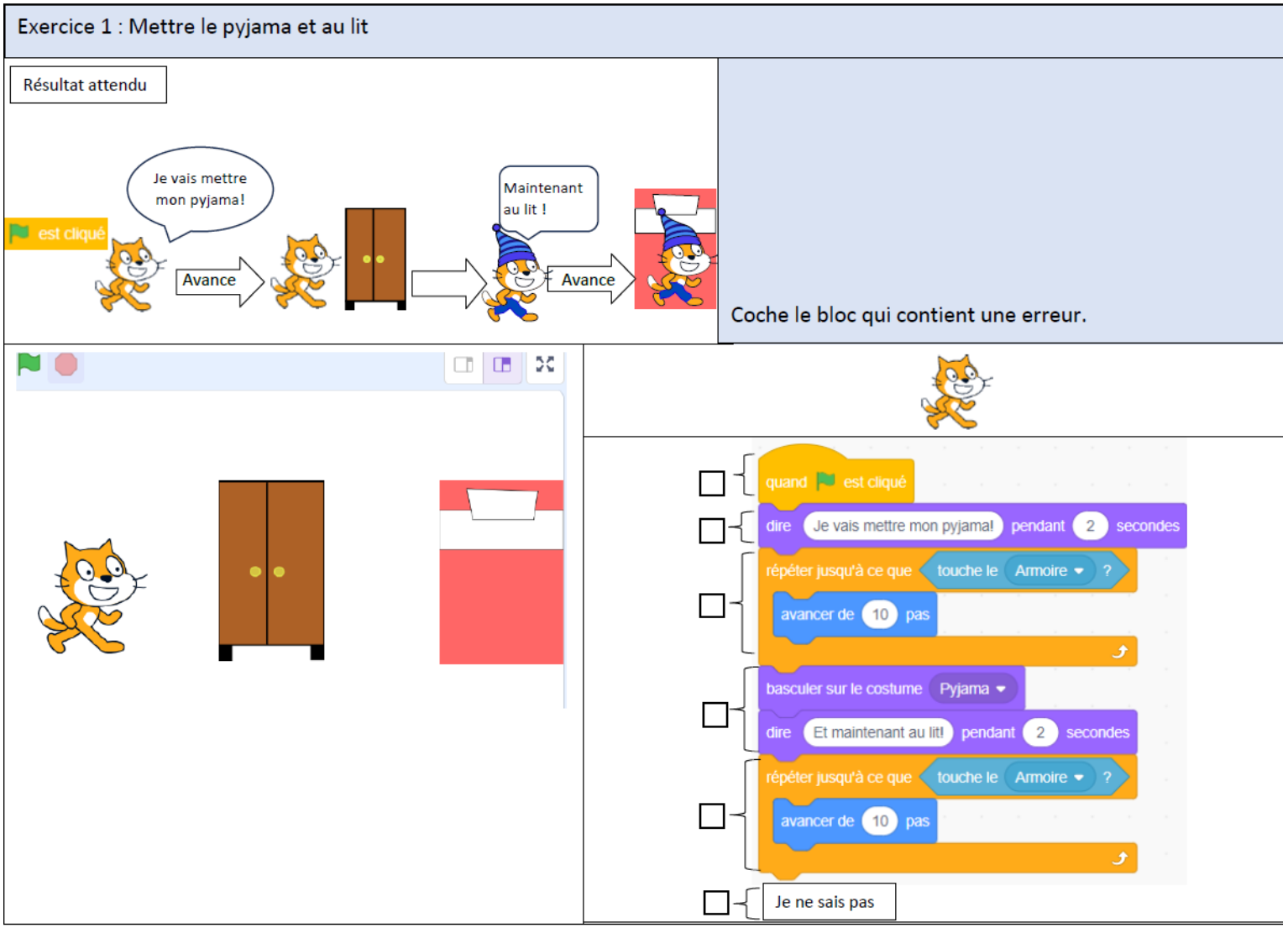
N=16	Pré-Test	Post-Test
Moyenne	12.88	17.25
Écart-Type	4.33	3.07
Q1	11.0	16.0
Médiane	12.0	18.0
Q3	16.0	20.0
Min	4	9
Max	20	20

Annexe 5 : tests de programmation, version originale traduite (El-Hamamsy et al., 2022)

Accessible au lien : https://journals.sagepub.com/doi/10.1177/07356331221081753

Consulté le 11.06.2024

Annexe 6 : tests de débogage, partie Scratch version originale

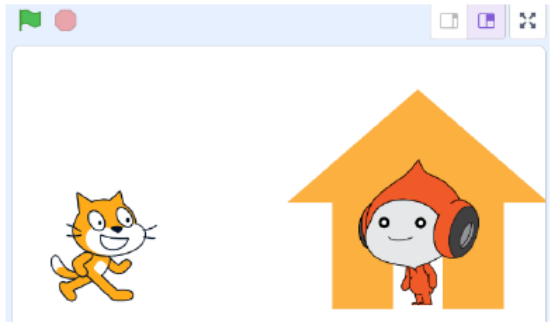


Deuxième partie
Qu'est-ce que tu changerais pour que ça marche ? Par quoi tu remplacerais ?
Ta réponse :

Exercice 2 : le cadeau

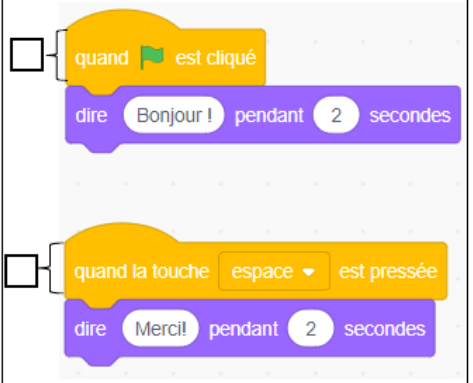
Résultat attendu

Coche l'élément qui pose problème.



☐

☐



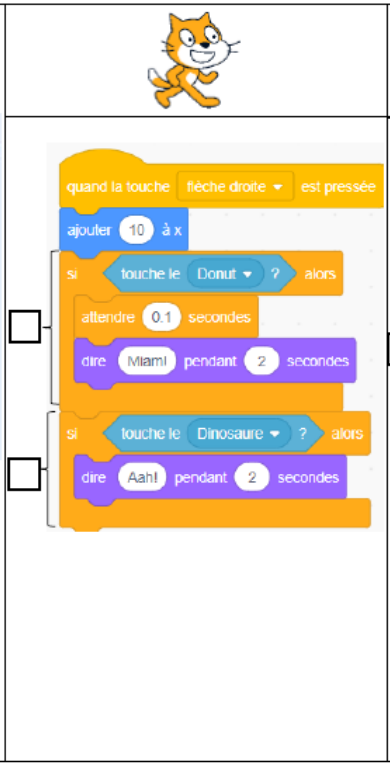
☐ Je ne sais pas

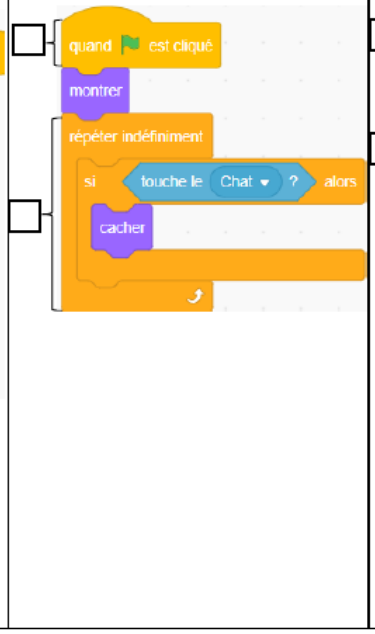
Deuxième partie
Qu'est-ce que tu changerais pour que ça marche ? Par quoi tu remplacerais ?
Ta réponse :

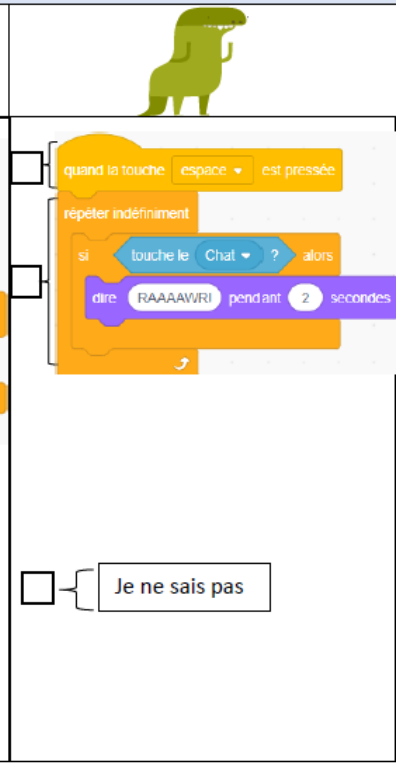
Exercice 3 : le dinosaure caché

Résultat attendu

Coche le ou les bloc(s) qui pose(nt) problème.





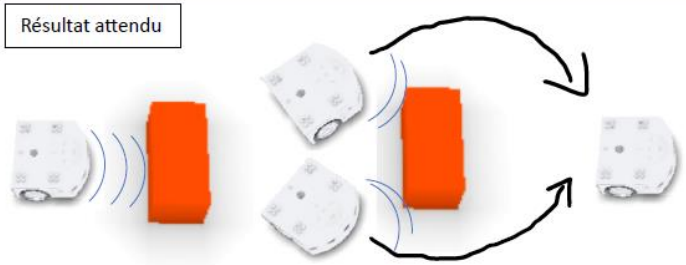


Deuxième partie
Qu'est-ce que tu changerais pour que ça marche ? Par quoi tu remplacerais ?
Ta réponse :

Annexe 7 : tests de débogage, partie Thymio version originale

Exercice 1 : Éviter l'obstacle

Résultat attendu

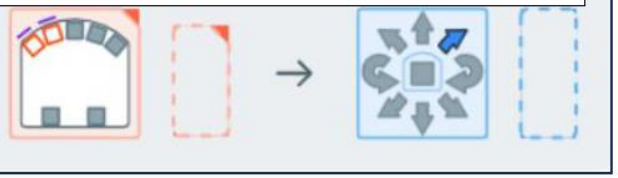


On veut que le robot évite l'obstacle.

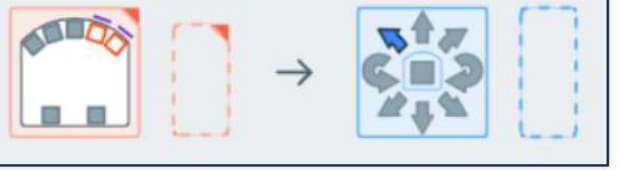
Quel bloc pose problème ?

Qu'est-ce que tu en ferais ? Tu le changerais/supprimerais ?

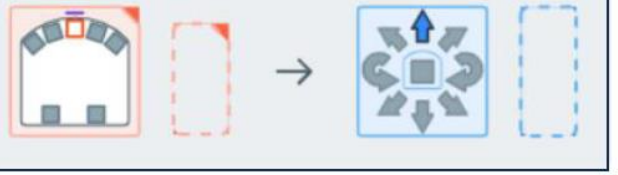
☐ Quand les capteurs de gauche détectent un objet, tourne à droite



☐ Quand les capteurs de droite détectent un objet, tourne à gauche

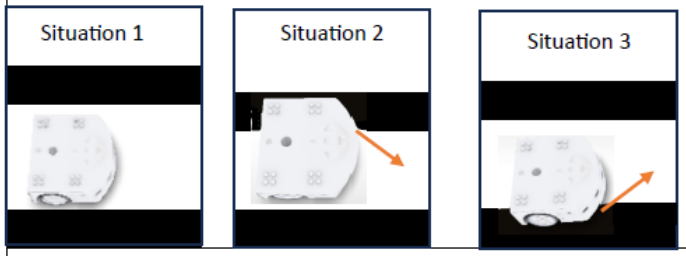


☐ Quand le capteur du milieu détecte un objet, avance tout droit



Exercice 2 : Suivre la piste

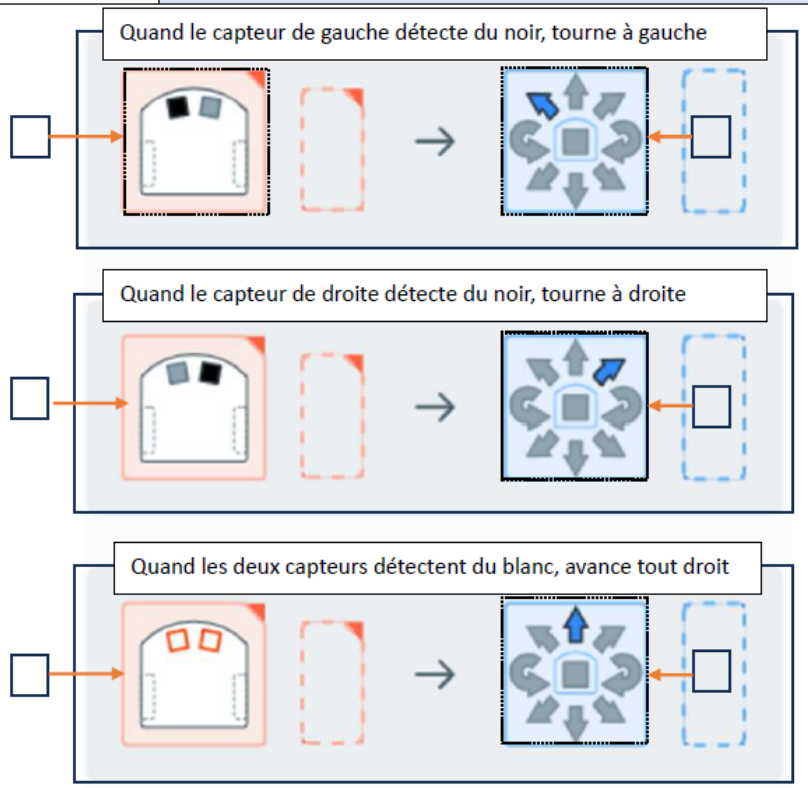
Résultat attendu



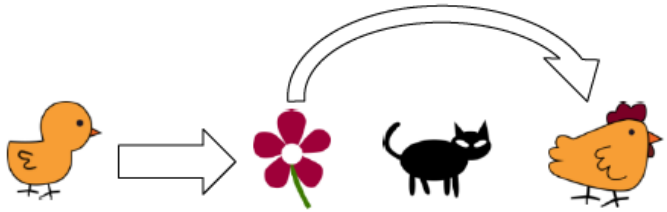
On veut que le robot suive la piste sans sortir dans le noir.

Quels blocs posent problème ?

Qu'est-ce que tu en ferais ? Tu les changerais/supprimerais ?

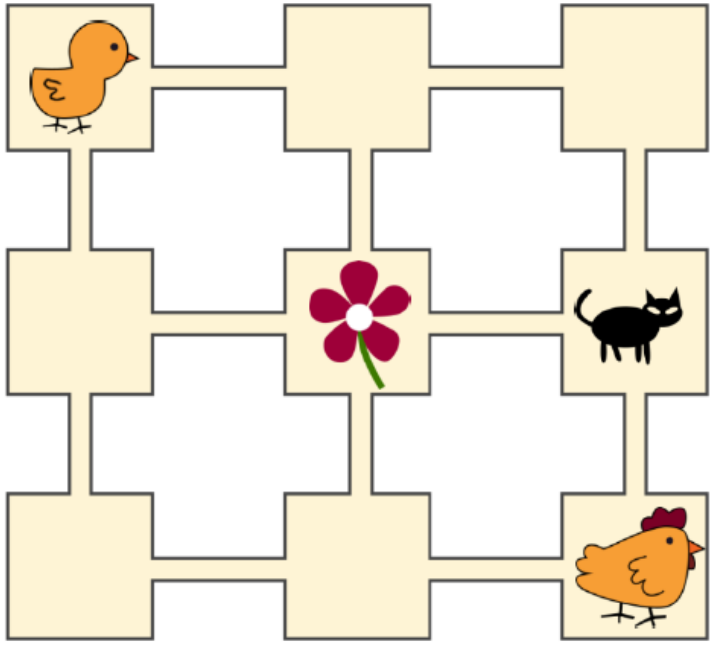


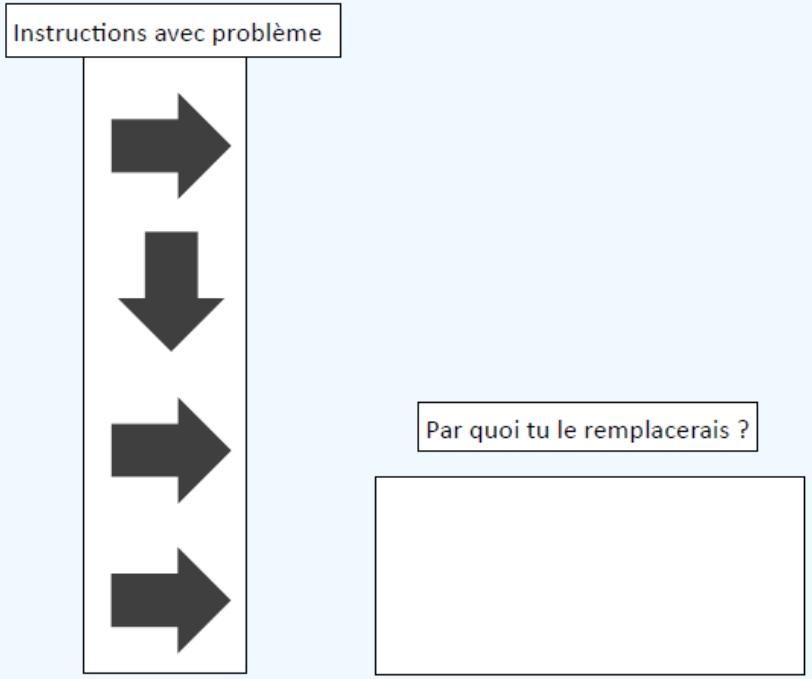
Exercice 3 : séquences



Le poussin doit rejoindre sa mère
Il doit ramasser la fleur sur son chemin
Il doit éviter le chat

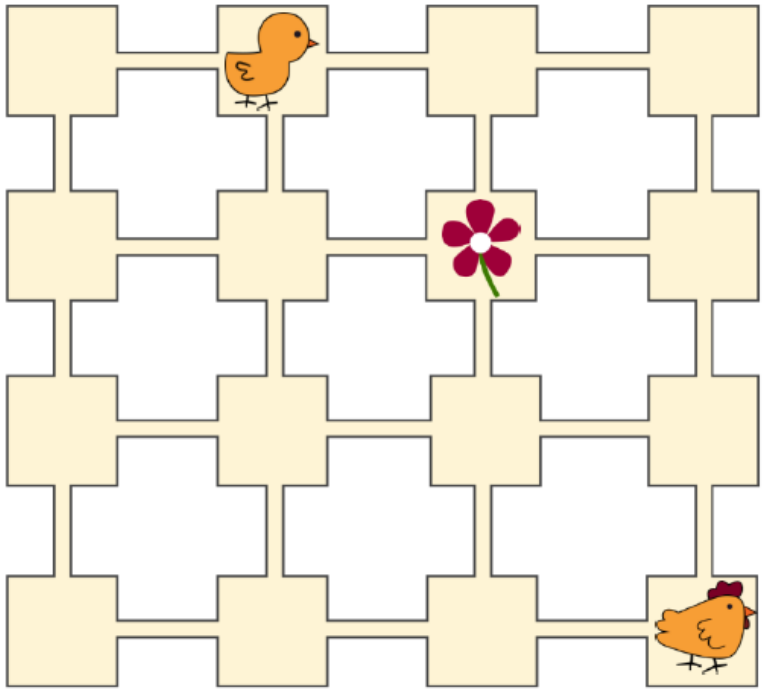
Dans la séquence, entour **le** bloc qui pose problème





Exercice 4 : boucles

Le poussin doit rejoindre sa mère
Il doit ramasser la fleur sur son chemin

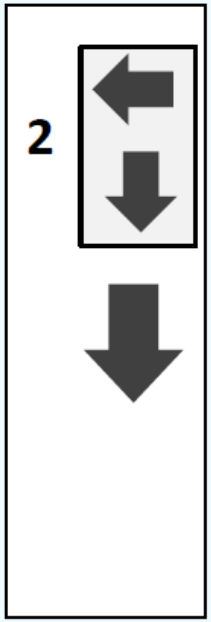


2 

Exemple: le poussin se déplace vers la droite, vers le bas, vers la droite et vers le bas.

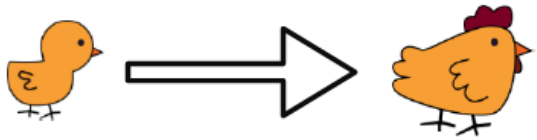
Entoure **le** bloc qui pose problème

Instructions avec problème



Par quoi tu le remplacerais ?

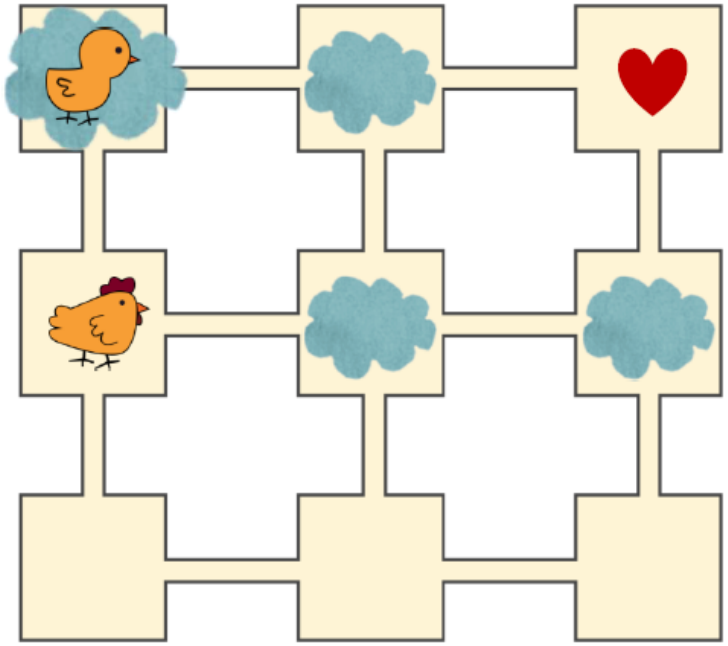
Exercice 5 : conditions



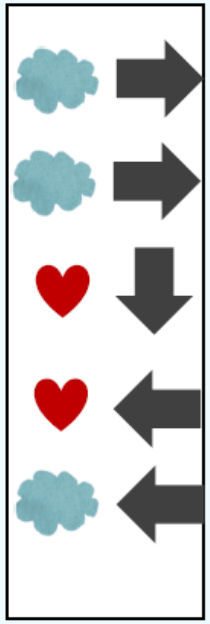
Le poussin doit rejoindre sa mère

Exemple: Si le poussin est dans un **nuage, alors** il **descend d'une case**

Entoure la partie qui pose problème



Instructions avec problème

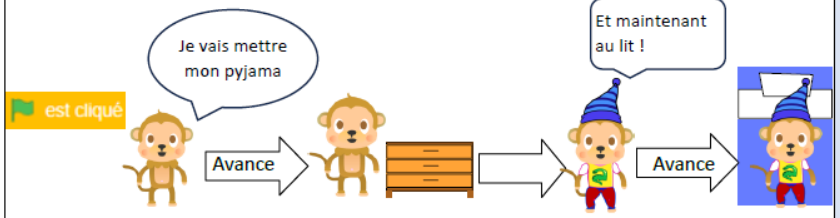


Par quoi tu le remplacerais ?

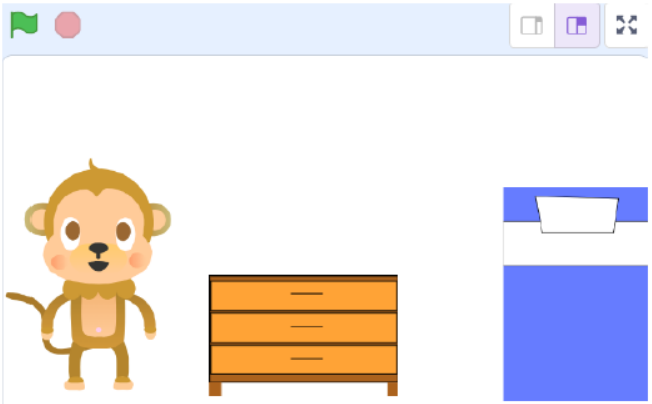
Annexe 8 : tests de débogage, partie Scratch version images modifiées

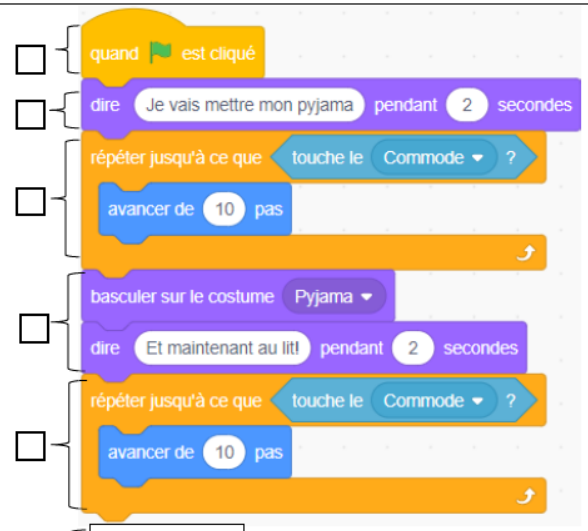
Exercice 1 : Mettre le pyjama et au lit

Résultat attendu



Coche le bloc qui contient une erreur.



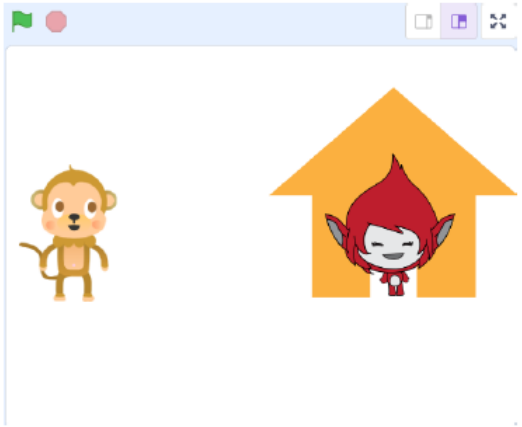


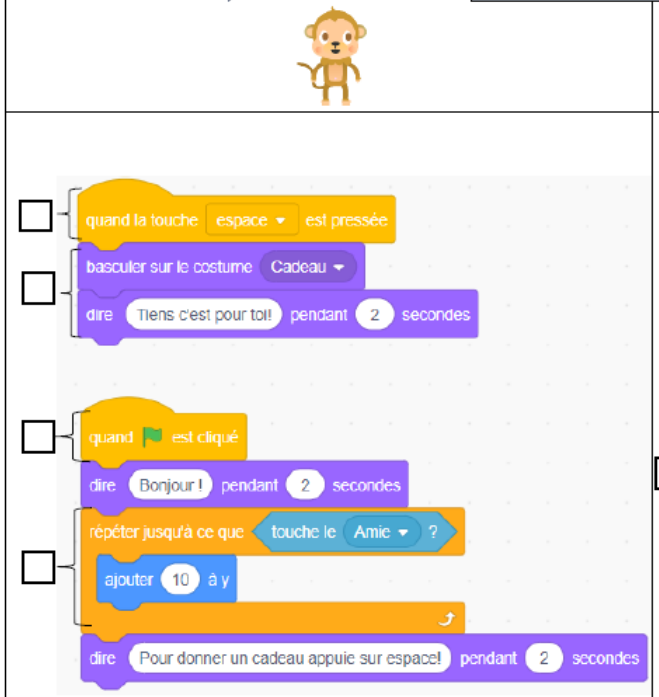
Deuxième partie
Qu'est-ce que tu changerais pour que ça marche ? Par quoi tu remplacerais ?
Ta réponse :

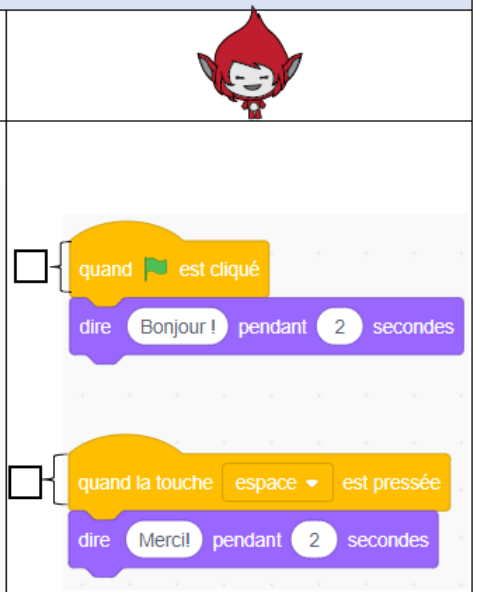
Exercice 2 : le cadeau

Résultat attendu

Coche l'élément qui pose problème.







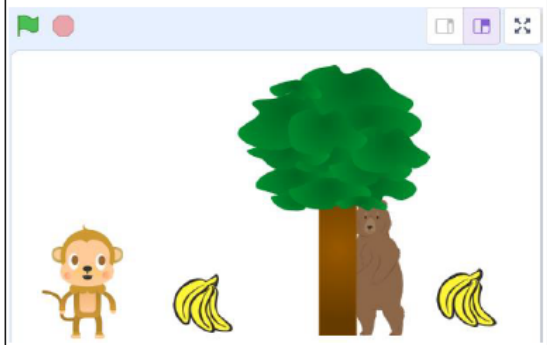
☐ Je ne sais pas

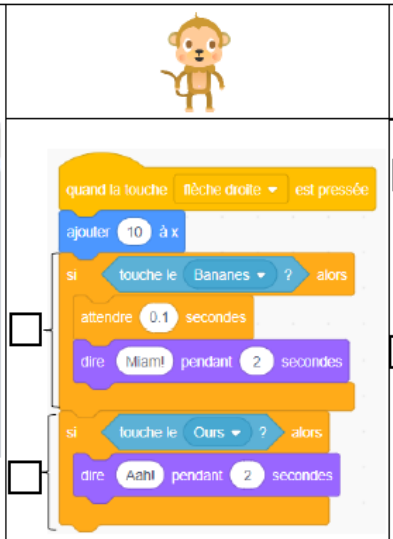
Deuxième partie
Qu'est-ce que tu changerais pour que ça marche ? Par quoi tu remplacerais ?
Ta réponse :

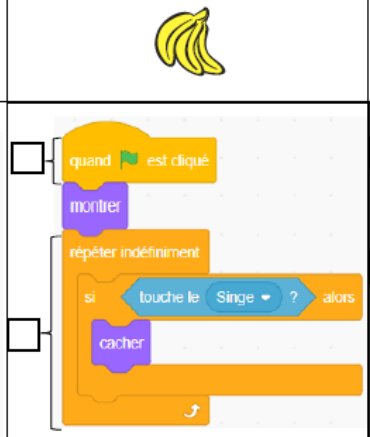
Exercice 3 : l'ours caché

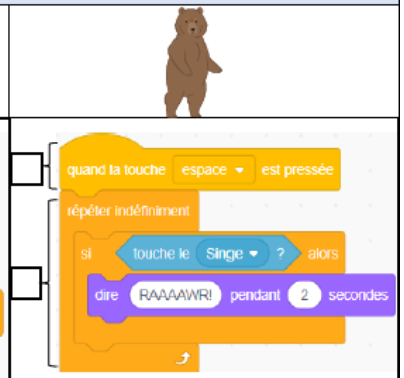
Résultat attendu

Coche le ou les bloc(s) qui pose(nt) problème.









☐ Je ne sais pas

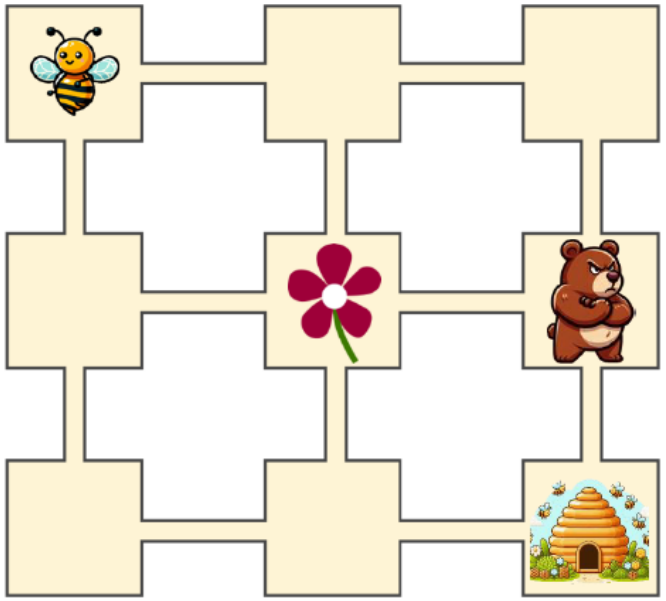
Deuxième partie
Qu'est-ce que tu changerais pour que ça marche ? Par quoi tu remplacerais ?
Ta réponse :

Annexe 9 : tests de débogage, partie Thymio version images modifiées

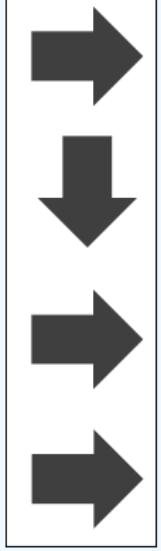
Exercice 1 : séquences

L'abeille doit rejoindre sa ruche
Elle doit ramasser la fleur sur son chemin
Elle doit éviter l'ours

Dans la séquence, entour **le** bloc qui pose problème



Instructions avec problème



Par quoi tu le remplacerais ?

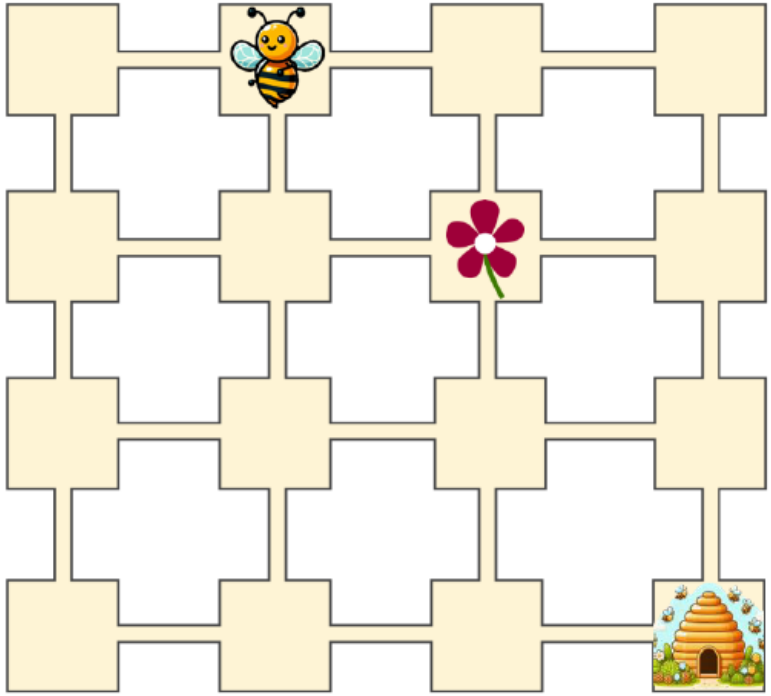
Exercice 2 : boucles

L'abeille doit rejoindre sa ruche
Elle doit ramasser la fleur sur son chemin

2 

Exemple: l'abeille se déplace vers la droite, vers le bas, vers la droite et vers le bas.

Entoure **le** bloc qui pose problème

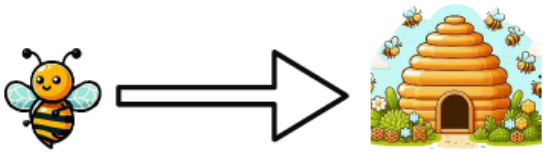


Instructions avec problème

2 

Par quoi tu le remplacerais ?

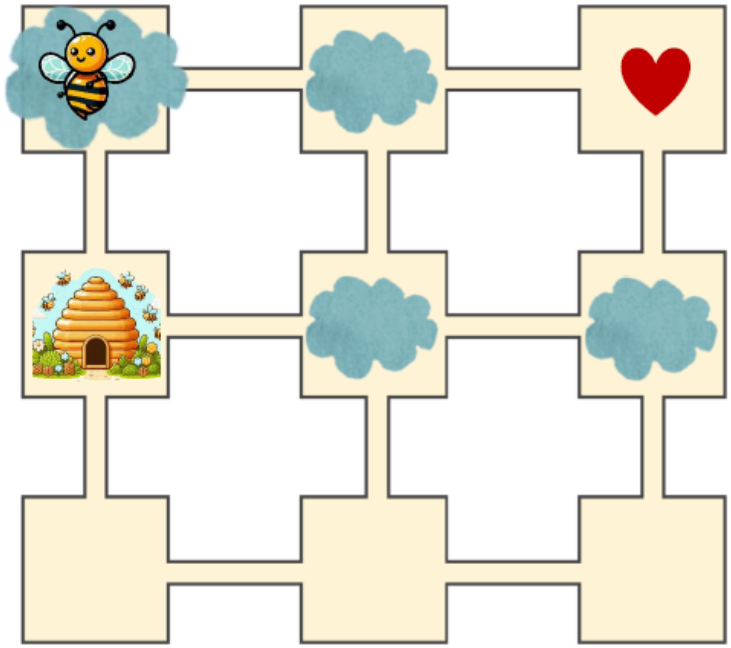
Exercice 3 : conditions



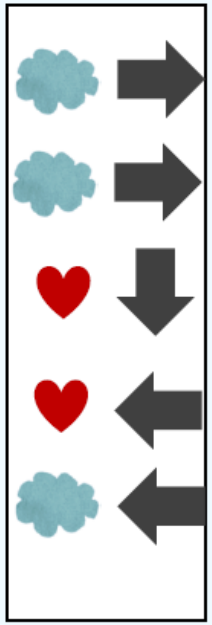
L'abeille doit rejoindre sa ruche

Exemple: Si l'abeille est dans un **nuage, alors** elle **descend d'une case**

Entoure la partie qui pose problème



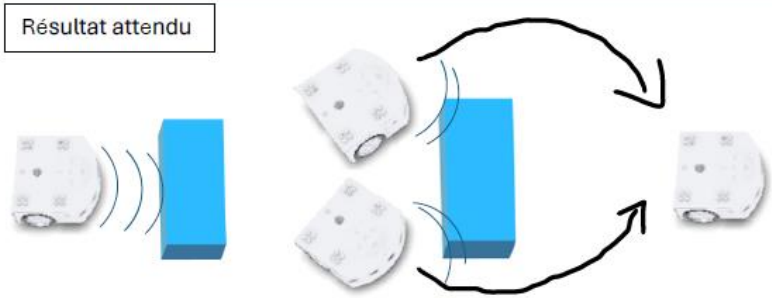
Instructions avec problème



Par quoi tu le remplacerais ?

Exercice 4 : Éviter l'obstacle

Résultat attendu

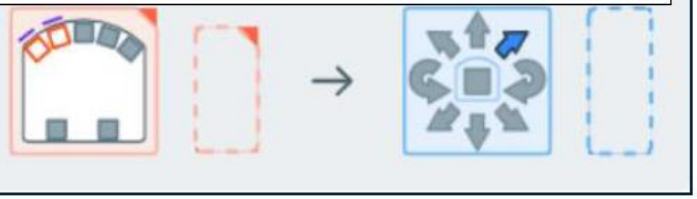


On veut que le robot évite l'obstacle.

Quel bloc pose problème ?

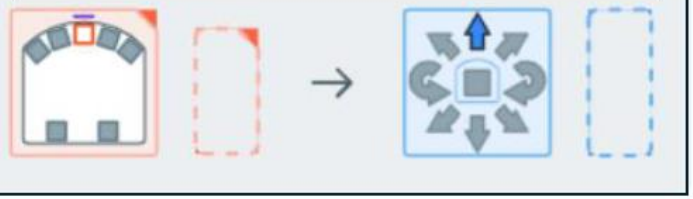
Qu'est-ce que tu en ferais ? Tu le changerais/supprimerais ?

☐ Quand les capteurs de gauche détectent un objet, tourne à droite



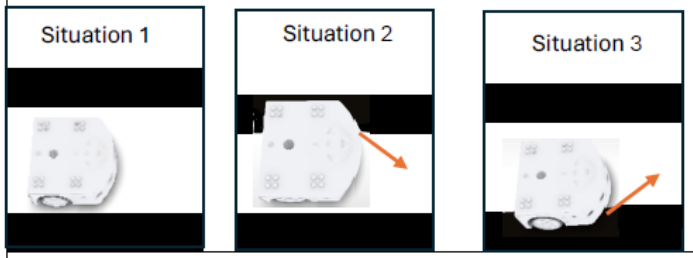
☐ Quand les capteurs de droite détectent un objet, tourne à gauche

☐ Quand le capteur du milieu détecte un objet, avance tout droit



Exercice 5 : Suivre la piste

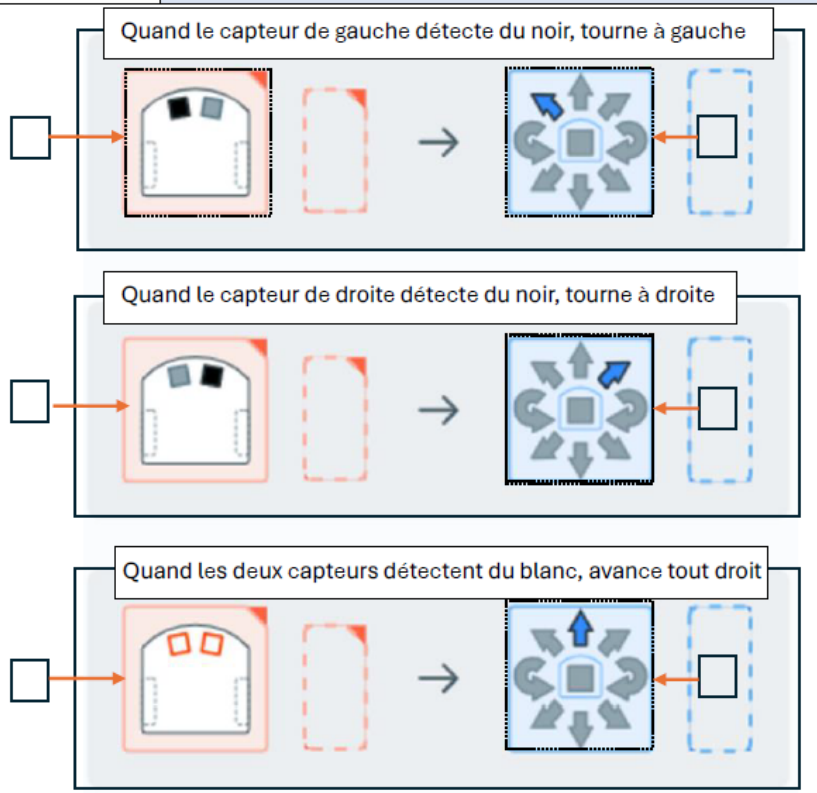
Résultat attendu



On veut que le robot suive la piste sans sortir dans le noir.

Quels blocs posent problème ?

Qu'est-ce que tu en ferais ? Tu les changerais/supprimerais ?

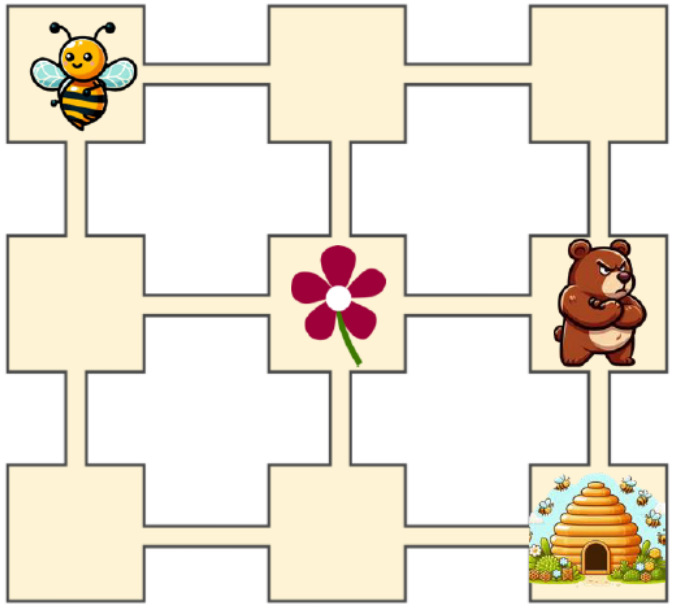


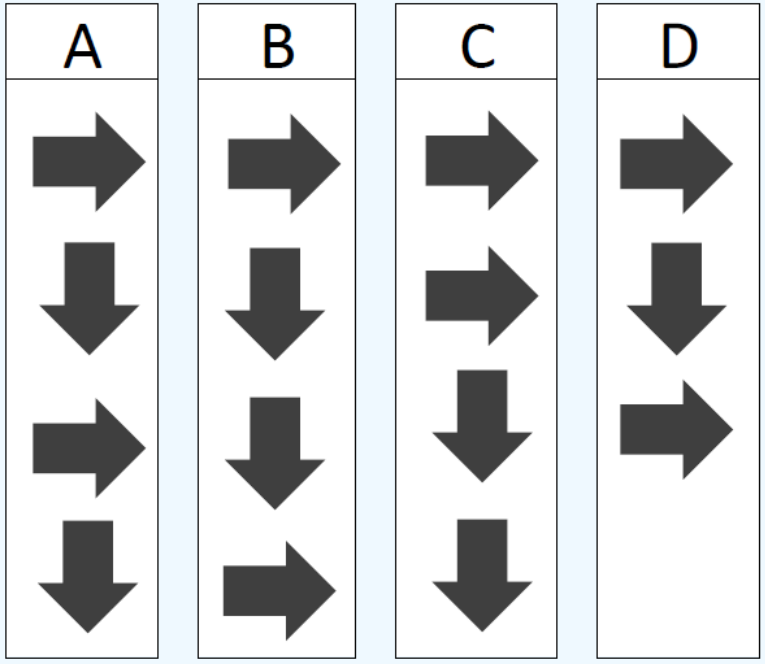
Annexe 10 : tests de programmation, variantes (images changées)

1

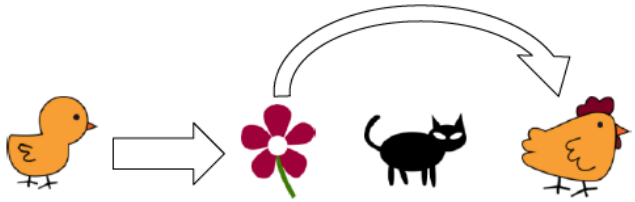
Emmène l'abeille à la ruche
Ramasse la fleur sur ton chemin
Attention à l'ours! Ne passe pas dans sa case

Teste les options A, B, C et D et choisis la bonne réponse



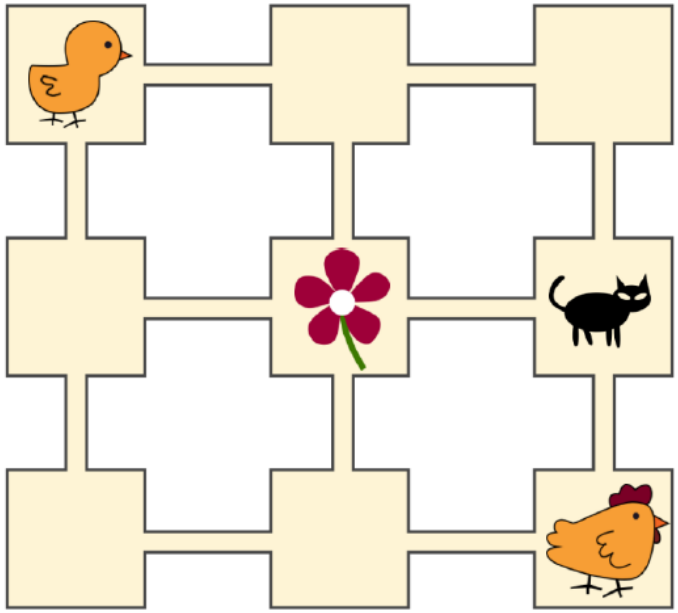


1



Emmène le poussin chez sa mère
Ramasse la fleur sur ton chemin
Attention au chat! Ne passe pas dans sa case

Teste les options A, B, C et D et choisis la bonne réponse



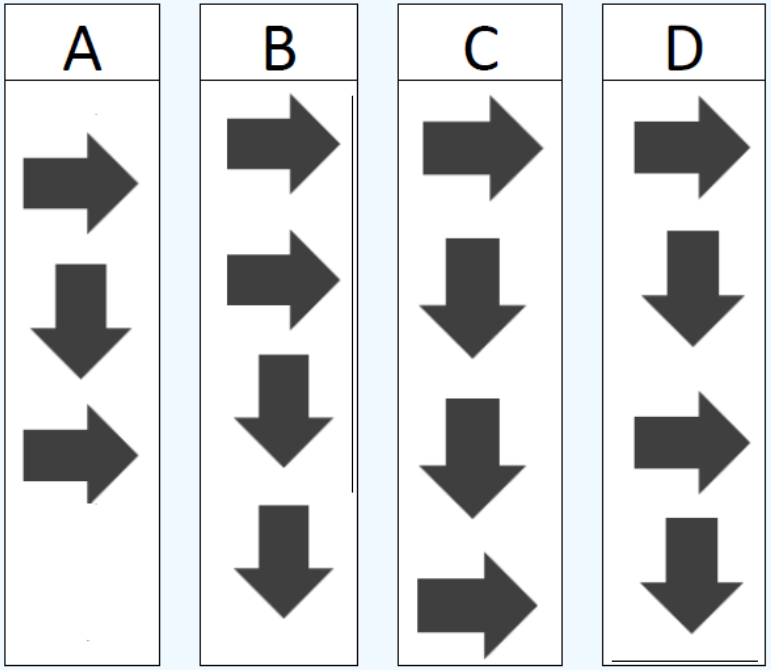


Figure 20 : réponses inversées (A<->D, B<->C)