Quelques pistes pour l'étude des situations d'informatique débranchée

Eric Duchene, Aline Parreau

Quelques pistes pour l'étude des situations d'informatique débranchée

Eric Duchêne & Aline Parreau

Abstract. Computer science unplugged is a scientific popularization project initiated in the 1990s by a team of New Zealand researchers. It enables participants to discover the major concepts of computer science, without a computer, through physical activies or the use of physical material, and also to initiate them into the computer science research process. This device, which is currently widely considered by both mediators and teachers, requires an in-depth analysis as a tool for transmitting knowledge. This analysis is currently in its beginning. Through three examples of popularization situations in computer science unplugged, we propose lines of thought based on observations with the perspective of a more complete didactic analysis of these situations and their transposition into classrooms.

Keywords. Theory of Didactic Situations, Anthropological Theory of the Didactic, Fundamental Theorem of Algebra

Résumé. L'informatique débranchée est un dispositif de médiation scientifique introduit dans les années 1990 par une équipe de chercheurs néo-zélandais. Elle permet, sans ordinateur et via des mises en scène corporelles ou l'utilisation de matériel original, de faire découvrir aux participant.e.s les grands concepts de la science informatique, mais aussi de les initier à la démarche de recherche en informatique. Ce dispositif actuellement largement considéré à la fois par les médiateur.rice.s et les enseignant.e.s nécessite une analyse approfondie en tant qu'outil de transmission de savoir. Cette analyse n'en est actuellement qu'à ses balbutiements. A travers trois exemples de situation de médiation en informatique débranchée (SMID), nous proposons des pistes de réflexions issues d'observations sur le terrain en vue d'une analyse didactique des SMID plus complète et d'une transposition en classe.

Mots-Clés. Théorie des Situations Didactiques, Théorie Anthropologique du Didactique, Théorème fondamental de l'algèbre

Table des matières

Nous remercions episciences.org pour l'hébergement en libre accès de l'*Épijournal de Didactique et Epistémologie des Mathématiques pour l'Enseignement Supérieur*.
Ce travail a été soutenu par l'Agence Nationale de la Recherche dans le cadre du projet ASMODEE - SAPS - RA - MCS 2021.

1. Introduction

La révolution numérique a induit, pour l'ensemble de la société, des questionnements, des besoins de connaissance ou encore des inquiétudes sur ce qui régit nos systèmes et outils modernes. Pour répondre à ces attentes, la médiation scientifique en informatique permet de construire un échange entre la communauté scientifique et le reste de la société.

Parmi les diverses activités de médiation en informatique, notre hypothèse de travail est que celles dites d'*informatique débranchée* constituent un levier particulièrement efficace pour faire comprendre au public ce qu'est la pensée informatique (Académie des Sciences, 2013 ; Bell & Lodi, 2019). Si ces activités de médiation en informatique débranchée (SMID dans la suite du papier) ont émergé dans les années 1960 (notamment avec Seymour Papert et la fameuse tortue Logo), elles sont devenues très populaires depuis l'initiative CS Unplugged [1] portée par des chercheurs néo-zélandais à fin des années 1990 (Bell, Witten, & Fellows, 1998).

En s'appuyant sur du matériel tangible ou une mise en jeu corporelle, les activités d'informatique débranchée permettent au public de s'abstraire de l'ordinateur pour mieux comprendre l'essence et les concepts de la science informatique. Il s'agit par là d'expérimenter la science informatique en insistant davantage sur les quatre grands piliers de cette science comme définis par (Dowek, 2011) : les données, les algorithmes, les langages et la machine. Pour traiter chacun de ces piliers, les supports d'activité sont variés : tapis grand format, objets en bois, cartes, légos, élastiques... En termes de notions abordées, les SMID cherchent à couvrir tous les champs de l'informatique, qu'il s'agisse de ceux qui figurent dans les programmes scolaires (algorithmes,réseaux), ou de ceux qui font l'objet de recherches accrues de nos jours (intelligence artificielle, sécurité informatique...).

Les activités d'informatique débranchée sont désormais utilisées par de nombreux chercheur.euse.s pour leurs activités de médiation, mais également par de nombreux enseignant.e.s en classe. En France, des groupes de travail ont émergé pour élaborer et analyser certaines SMID existantes, et en construire de nouvelles. Ces groupes se sont constitués au sein des IREM, de la Société Informatique de France [2], ou encore de projets de recherche soutenus par le ministère. Du matériel à disposition des enseignant.e.s sous forme de fiches détaillées sont également régulièrement proposés pour favoriser leur déploiement auprès des scolaires.

Puisque ces SMID constituent un outil de transmission de connaissances, il semble naturel de s'interroger sur la nature de cette transmission, notamment vers les scolaires. Naturellement, l'analyse de cette transmission du savoir peut se faire en suivant une approche éprouvée en didactique des sciences, à savoir articuler les aspects épistémologiques avec les aspects didactiques (Artigue, 1988).

Dans la littérature, les études épistémologiques autour des SMID ne sont pas si fréquentes. Un travail d'identification et de structuration des savoirs élémentaires reste à développer pour de nombreuses situations. De manière générale, dans les travaux existants, le niveau de description des notions en jeu reste assez général. L'analyse didactique détaillée de la situation elle-même, comme des savoirs de la discipline, nécessite une granularité plus fine. En effet, un schéma d'ingénierie didactique rigoureux se doit d'identifier tous les savoirs qui peuvent entrer en jeu, notamment lorsque les variables didactiques de la situation évoluent. Par ailleurs, certaines stratégies de résolution d'une situation sont souvent occultées. Actuellement, en France, seules quelques SMID ont été éprouvées de la sorte, notamment suite aux travaux conjoints entre informaticiens, didacticiens et enseignants dans certains IREM ou au sein de la Fédération Recherche Maths à Modeler [3] (voir par exemple (Godot, 2005 ; Meyer &

1. https://www.csunplugged.org/en/
2. https://www.societe-informatique-de-france.fr/mediation/infosansordi/
3. https://mathsamodeler.ujf-grenoble.fr/accueil.html

Modeste, 2022 ; Modeste, 2012)).

Des concepteurs du mouvement CS Unplugged viennent récemment de partager ce constat au niveau international (Bell & Vahrenhold, 2018). Leur survey illustre en quoi cette étude des savoirs est cruciale, en particulier pour les enseignant.e.s, à travers l'analyse approfondie de deux exemples de SMID (coloration de graphes et réseaux de tri).

En ce qui concerne l'analyse didactique des SMID, et plus généralement de l'informatique, il s'agit d'un chantier d'envergure qui devrait nourrir les prochaines décennies étant donné la présence grandissante de l'informatique dans les programmes scolaires et la nécessité de structurer les pratiques. Pour cela, la didactique de l'informatique (comme discipline scientifique), née dans les années 1980 (Pea, 1987), trouve un regain d'intérêt récent et se structure encore. Au niveau national, les études récentes dans ce domaine portent principalement sur l'enseignement en classe. Celles-ci s'inscrivent dans la lignée du travail de Baron et Bruillard (Caillot, 1997), qui illustre ce besoin de complémentarité entre chercheurs en Sciences Humaines et Sociales et informatique pour aborder le sujet. Récemment, le projet ANR JCJC DEMaIn [4] s'est intéressé aux relations entre didactique des mathématiques et de l'informatique et aux enjeux des interactions mathématiques-informatique sur les plans épistémologique et didactique. Le projet ANR Asmodée cherche quant à lui à développer des outils conceptuels qui permettent d'analyser les SMID avec rigueur, en vue d'une réappropriation adéquate par les médiateur.ice.s, chercheur.euse.s et enseignant.e.s.

Dans cet article, nous proposons des pistes de réflexions issues d'observations sur le terrain en vue d'une analyse didactique des SMID et d'une transposition en classe. Nous présentons tout d'abord (en partie 2) trois SMID qui nous semblent représentatives de la diversité des pratiques possibles. Puis nous proposons (en partie 3) une réflexion sur différents critères qui peuvent caractériser une SMID, et nous verrons en quoi certaines d'entre elles sont vraiment spécifiques à la discipline informatique. La quatrième partie se concentre sur le potentiel des SMID pour une utilisation en classe, la mise en pratique et les freins potentiels.

2. Trois situations d'informatique débranchée

Nous présentons ici trois situations qui serviront de fils rouges pour illustrer notre compréhension des SMID. Pour chacune d'entre elle, nous donnons les éléments principaux nécessaires à sa compréhension ainsi que des liens vers des descriptifs plus détaillés.

2.A. Le base-ball multicolore

Cette situation est issue de l'ouvrage original CS Unplugged (Bell et al., 1998), et aussi connu sous le nom de *jeu de l'orange*. Il permet entre autres d'aborder les notions d'algorithme et de complexité, ainsi que la problématique de ressources partagées et d'accès concurrent à ces ressources. Cette situation peut être retrouvée sur le site web [5] de l'IREM de grenoble, avec un séquencement possible en classe, sur la page web de Marie Duflot-Kremer [6] ou encore dans l'ouvrage collectif (Collectif, 2017).

2.A.a. Description de la situation

On considère une situation avec N participant.e.s. En pratique, la situation fonctionne bien pour N compris entre 4 et 8. Chaque participant.e se place dans un cerceau de couleur (sa base) disposés de façon circulaire et reçoit deux balles de la même couleur que sa base (chaque participant.e a une couleur qui lui est spécifique). Les deux balles sont dans chaque main du joueur.euse. On retire une

4. https://imag.umontpellier.fr/~modeste/demain.html
5. https://irem.univ-grenoble-alpes.fr/recherche-action/themes/informatique-de-l-ecole-jusqu-au-lycee/activite-algorithmique-base-ball-multicolore-498730.kjsp
6. members.loria.fr/MDuflot/files/med/baseball.html

balle au hasard dans l'une des mains d'un.e des joueur.euse.s. La position ainsi obtenue correspond à la position finale voulue. La figure 1 illustre une telle position avec $N = 5$ et une balle violette retirée. Les bases y sont représentées par des grands disques colorés et les balles par les petits disques. Le disque blanc correspond au trou.

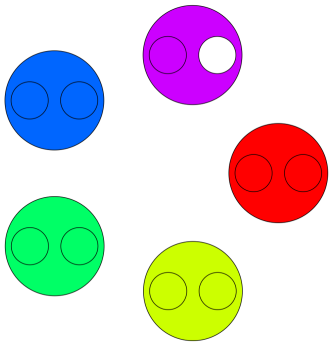

FIGURE 1 – Position finale du base-ball à cinq couleurs

On mélange ensuite toutes les balles des mains des joueur.euse.s pour obtenir une position de départ aléatoire. Le but du jeu est de revenir à la position finale, sachant que la seule règle autorisée consiste à prendre une balle à l'un.e. de ses voisin.e.s, lorsqu'on a une main de libre. Une configuration possible de départ suivie de deux premiers coups est représentée sur la figure 2.

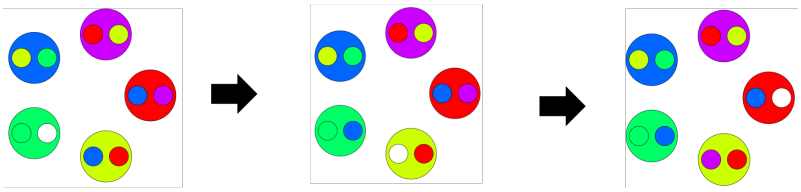

FIGURE 2 – Exemple de deux premiers coups au base-ball multicolore

Pour mieux comprendre la situation, une version jouable en ligne est disponible sur le site de Benjamin Wack[7]. Il existe plusieurs variantes de cette activité, qui consistent à changer le rôle du participant.e dans la situation. Dans la version originale, le participant contrôle sa base et c'est celui.elle qui a une main libre qui peut choisir une balle parmi les quatre balles voisines. Dans une version centralisée, on peut supposer qu'une seule personne contrôle la situation et le déplacement des balles entre les joueur.euse.s. Enfin, dans la version *tous pour un*, les personnes jouent elles-mêmes le rôle des balles. Dans la section suivante, nous allons analyser la situation dans sa version originale.

2.A.b. Stratégies de résolution possibles

Plusieurs stratégies de résolution peuvent émerger chez les participant.e.s. Parmi elles, on rencontre souvent les deux algorithmes suivants dans la plupart des expérimentations qui ont été faites sur ce problème.

Algorithme "tournant". On ne décide de déplacer les balles que dans le sens des aiguilles d'une montre. Ainsi il y a à chaque fois seulement deux possibilités. On prend parmi les balles possibles celui qui est le plus loin de sa base. C'est celui qui est mis en place sur la page de Benjamin Wack.

Algorithme "en ligne". On coupe en deux le cercle formé par les joueur.euse.s (par exemple au niveau de la base où il manque une couleur) pour obtenir une ligne, dont l'élément le plus à gauche est la base ayant une couleur en moins. Puis on remplit les bases de la droite vers la gauche avec la bonne

7. www-verimag.imag.fr/~wack/baseball

couleur : on considère la dernière base en partant de la droite qui n'est pas correcte, et on cherche à ramener les balles de sa couleur vers cette base en déplaçant le trou à droite de la balle concernée, puis en déplaçant la balle d'un cran à droite jusqu'à ce qu'il retrouve sa base.

Résolution avec une routine. Un troisième algorithme peut être trouvé en utilisant une routine qui déplace une balle sur le trou sans toucher aux autres. On déplace ainsi une par une les balles à leur position finale. Cela simule le concept de fonction en programmation et de décomposition de programme.

2.A.c. Analyses des algorithmes proposés

Dans cette sous-partie, nous analysons les solutions proposées sous deux aspects correspondant aux enjeux principaux en algorithmique : à savoir leur correction (càd le fait que l'algorithme réponde bien au problème posé au bout d'un temps fini, quelque soit l'instance de départ), et leur complexité (càd sa qualité en termes de nombres d'opérations élémentaires effectuées). Une analyse plus poussée a été menée par l'IREM de Grenoble (Althuser, Brasset, Rasse, Vincent, & Wack, 2019).

L'algorithme "tournant" n'est en fait pas correct. S'il marche pour la plupart des cas rencontrés, il peut boucler infiniment pour d'autres. Pour s'en convaincre, on peut prendre la configuration finale à quatre bases et échanger deux couleurs consécutives. Alors on se convainc assez rapidement que l'algorithme boucle infiniment. Le problème vient du fait qu'on peut tout à fait déloger une balle bien placée si les deux balles qui peuvent se déplacer sont bien placées dans leur base et qu'il faut libérer l'une des deux.

On peut démontrer la correction de l'algorithme en ligne assez facilement avec l'argument suivant : on ne touche jamais aux bases qui sont bien remplies en partant de la droite. À chaque étape, on amène une balle de plus dans une base à droite à laquelle on ne touchera plus. Ainsi à la fin, tout est bien placé. En ce qui concerne la complexité de cet algorithme, on peut regarder le nombre de déplacements qu'il faut à l'étape i où l'on essaie de remplir la i-ème base en partant de la droite. Il y a au pire deux balles à déplacer, qui sont à une distance au plus $n-i$ de leur base. Il faut donc calculer le coût de déplacement d'une balle dans sa base située à distance inférieure ou égale à $n-i$. Pour cela, il faut tout d'abord déplacer le trou dans la base juste à droite de la balle. Le trou se situant à une distance au plus $n-i$ de la balle, il y a au plus $n-i$ étapes pour ramener le trou à droite. Ensuite on peut déplacer la balle à droite d'une case (une étape), puis il faut remettre le trou à sa droite (deux étapes). En trois étapes on a donc décalé le trou et la balle d'une case à droite. Ainsi en au plus $4(n-i)$ étapes, la balle a retrouvé sa base. On répète cela deux fois pour chaque i, et i allant de 1 à $n-1$ ce qui donne un nombre d'opérations borné par $\sum_{i=1}^{n-1} 8(n-i)$. Au final, cet algorithme a une complexité de $O(n^2)$ dans le pire des cas. Cette borne est d'ailleurs atteinte lorsque toutes les couleurs sont inversées (càd les couleurs les plus à droites sont à gauche).

La correction de l'algorithme avec la routine est assez facile à montrer vu qu'à chaque étape de la routine, une balle de plus est placée sur sa base et ne sera plus "retouchée ensuite" (à part au sein de la routine). On peut trouve une routine qui fonction en $O(d)$ étapes où d est la distance entre le trou et la balle. Cela amène aussi à un algorithme final de complexité $O(n^2)$.

Une question naturelle qui se pose alors est l'optimalité de ces deux derniers algorithmes en terme de complexité. Est-il possible de construire des algorithmes de complexité inférieure à ceux décrits précédemment ? Pour répondre à cette question, on peut calculer un nombre minimal de déplacements à faire de la manière suivante :

- A chaque étape, une seule balle est déplacée
- Pour amener une balle à sa base, il faudra faire au moins d déplacements où d est la distance entre la balle et sa base.
- Il faut donc au minimum $\sum_{i\in P} d_i$ étapes où P est l'ensemble des balles et d_i la distance de la balle i à sa base.

En prenant la configuration où toutes les balles sont dans la base opposée, on a $d_i = n/2$ et on obtient donc au moins n^2 étapes. Ainsi la complexité du problème est en $\Theta(n^2)$.

2.A.d. Notions mises en jeux

Pour le base-ball multicolore, l'analyse qui précède illustre que la question majeure qui sous-tend cette situation est celle de l'étude de la correction et de la complexité d'un algorithme. Il est important de montrer aux participant.e.s que la validité d'un algorithme doit s'exprimer sur l'ensemble des instances possibles. En ce qui concerne la complexité, elle est étroitement liée à la durée de calcul des machines pour faire tourner l'algorithme, et donc il s'agit d'un autre enjeu majeur dans la réalisation d'un algorithme.

Cette situation fait aussi apparaître les notions d'accès concurrents dans les programmes, illustrée lorsque plusieurs personnes veulent simultanément une même balle. Une solution est alors d'exprimer des règles de priorité afin d'obtenir un comportement déterministe et non-bloquant de l'algorithme. Ceci est induit par la nature de la situation, où chaque participant.e joue le rôle d'une machine qui exécute une partie de l'algorithme, simulant ainsi les fondements d'un système distribué.

2.B. Le réseau de tri

Cette situation est également issue de l'ouvrage original CS Unpluuged (Bell et al., 1998). Elle permet tout d'abord d'expliquer la notion de tri en informatique, qui est l'une des actions que fait le plus souvent un ordinateur. Par ailleurs, elle aborde la notion d'algorithmique parallèle, qui permet d'accélérer la vitesse d'exécution d'un algorithme lorsque plusieurs ordinateurs se coordonnent pour le faire tourner.Des déroulés peuvent être retrouvés sur les pages web de CS Unplugged [8] et de Marie Duflot-Kremer [9].

2.B.a. Description de la situation

La situation a lieu avec N participant.e.s, et on choisira très souvent en pratique $N = 6$ pour que le réseau ne soit ni trop simple si trop compliqué. On dessine au sol le réseau suivant (avec de la craie, ou des cerceaux et des lattes de sport).

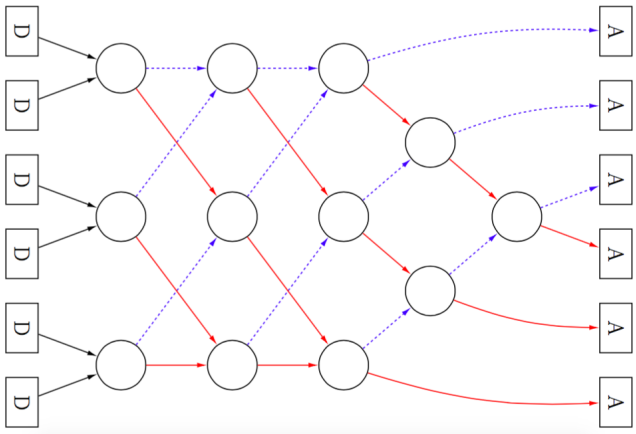

FIGURE 3 – Réseau de tri à six éléments

Sur chaque position D (départ) on place un élève. On distribue à chaque élève une petite fiche avec un nombre différent qu'ielles gardent face cachée. Chaque élève avance ensuite d'un cran en suivant la flèche dans la première rangée de cercles. Il y a donc après cette étape deux élèves par cercle. Puis les élèves appliquent la règle suivante jusqu'à être chacun.e dans une case A (arrivée) :

8. https://www.csunplugged.org/fr/topics/sorting-networks/
9. https://homepages.loria.fr/MDuflot/med/reseautri.html

— Tant que vous êtes tout.e seul.e dans un cercle, attendre que quelqu'un.e vous rejoigne.
— Une fois que vous êtes deux dans un cercle, comparez vos nombres. L'élève avec le plus petit nombre part à gauche dans le prochain cercle en suivant la flèche bleue, et l'autre suit la flèche rouge.

Une fois que tous les élèves sont dans les cases A, on leur demande de montrer le nombre. On constate qu'ils sont triés dans l'ordre croissant de gauche à droite !

2.B.b. Analyse de la situation

Cette activité est différente de la précédente, dans la mesure où l'on ne demande pas aux participant.e.s de trouver une solution à un problème. En revanche, c'est un ensemble de questions qui leurs sont suggérées ou qui émergent naturellement :

— Pourquoi cet algorithme a-t-il bien trié les nombres ?
— Cela marche-t-il à tous les coups quelque soit la disposition des nombres au départ ? Comment en être sûr ?
— Cela peut-il marcher en prenant le réseau à l'envers ?
— Combien de "temps" faut-il pour être trié ? (que veut dire la notion de temps lorsque des opérations sont menées en parallèles ?)
— Comment fabriquer de tels réseaux de tris ? Quels sont les plus efficaces ?

Nous donnons ci-dessous plusieurs éléments de réponses à ces questions.

Correction de l'algorithme. Pour un réseau avec N participant.e.s, ce n'est pas facile de démontrer sa correction car il y a $N!$ entrées possibles à l'algorithme. Le principe dit de "zero-un" explique qu'il suffit de montrer que le réseau trie les 2^N séquences de longueur N formées uniquement de 0 et 1. Malgré cela, il reste co-NP-complet de démontrer qu'un réseau donné est correct (Parberry, 1991). Dans l'instance fournie avec $N = 6$, il est assez facile de se convaincre que les valeurs min et max seront toujours placées au bon endroit à la fin. Pour les autres, l'étude de cas est un peu plus fastidieuse mais reste possible directement sur le réseau.

Dans le sens inverse. Le réseau ne fonctionne pas dans le sens inverse. Il suffit par exemple de prendre comme instance de départ les nombres $4, 1, 2, 3, 5, 6$ dans cet ordre. Il s'agit toutefois d'une bonne illustration de la nécessité de vérifier la correction des algorithmes car les contre-exemples ne sont pas si nombreux que ça pour $N = 6$.

Par ailleurs, une autre question naturelle est de savoir ce qu'il se passe si on change le sens de départ depuis un cercle (flèche gauche ou droite). Là, il sera plus aisé de vérifier qu'un tri dans l'ordre décroissant aura lieu.

Complexité algorithmique. La nouveauté d'une telle situation vis à vis de la complexité algorithmique, c'est que le nombre d'opérations élémentaires total n'est plus le critère principal. En revanche, le nombre de couches du réseau (càd le nombre de fois où l'on peut effectuer des opérations élémentaires en parallèle) devient le critère de qualité le plus scruté. Dans l'exemple du réseau ci-dessus, on voit qu'il fonctionne en cinq étapes (càd cinq couches de comparaisons). Pour $N = 6$, il a été démontré qu'il n'était pas possible de faire un réseau de tri avec quatre couches seulement. D'ailleurs, on connaît la taille optimale des réseaux jusqu'à $N = 16$ (Bundala & Závodný, 2014) mais le problème reste ouvert au-delà.

Construction de réseaux. Il est possible de construire des réseaux optimaux pour des petites valeurs de N. Il s'agit d'ailleurs d'une activité de prolongement à faire sur table avec les élèves.

Pour $N = 4$, on a besoin d'au minimum cinq comparaisons et donc trois couches. C'est possible avec le réseau suivant :

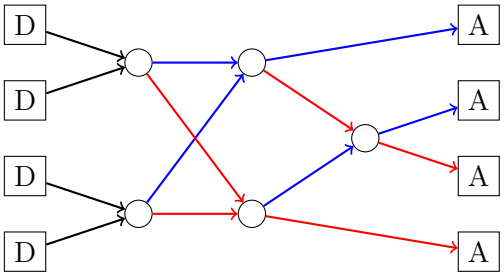

FIGURE 4 – Réseau de tri optimal à quatre éléments

Vérifier que les réseaux construits par les élèves fonctionne et trie les quatre nombres quelle que soit leur disposition de départ fait partie des activités possibles qui illustrent le besoin d'une preuve de correction.

Il existe ensuite des stratégies pour construire des réseaux par récurrence. Par exemple on peut ajouter les éléments un par un en comparant l'élément supplémentaire avec tous les autres. Cela donne des réseaux avec $2N-3$ couches pour N éléments en entrée (à chaque étape, on ajoute deux couches). On peut toutefois faire beaucoup mieux car il existe des réseaux de profondeur $\log^2 N$ comme le tri bitonique utilisé dans les processeurs graphiques (Batcher, 1968). Pour ce réseau, l'idée est de faire une dichotomie puis de fusionner les deux séquences triées.

2.B.c. Notions mises en jeu

La question du tri est l'objectif premier. Tout d'abord, il s'agira de discuter avec les élèves de la notion de tri, qui n'est pas identique de celle du langage courant (par exemple, trier ses déchets). En informatique, ce terme signifie ranger par ordre croissant et il sera intéressant d'aborder avec les élèves les raisons pour lesquelles cette activité est utilisée de façon centrale en informatique. En l'occurrence, le besoin d'avoir des algorithmes très rapides est un enjeu naturel, car les tris portent souvent sur des données massives. Par exemple, quand on demande une recherche sur un moteur de recherche, les résultats (souvent par milliers ou par millions) apparaissent de façon triée. Le fait que plusieurs processeurs puissent exécuter simultanément des parties d'un algorithme et sans conflit est une préoccupation majeure de l'informatique actuelle. Dans les enjeux liées à la parallélisation d'un algorithme, il reste à savoir quelles parties d'un algorithme peuvent être parallélisées et lesquelles ne peuvent pas l'être. On illustre bien souvent cet enjeu avec l'exemple d'un ensemble de personnes qui souhaitent creuser un trou. S'il s'agit d'un trou de 10m de long et de 50cm de profondeur, on pourra facilement paralléliser la tâche en donnant une pioche à chaque personne. S'il s'agit en revanche d'un trou de 10m de profondeur et de 50cm de large, la parallélisation ne semble pas adaptée.

2.C. La machine qui apprend à jouer toute seule

La SMID que nous allons décrire a été proposée par plusieurs membres du groupe Informatique Sans Ordinateur français. Elle permet d'illustrer la notion d'algorithmes d'apprentissage par renforcement qui font partie d'une classe d'algorithmes très utilisés de nos jours en intelligence artificielle. On trouvera ci-dessous plusieurs ressources sur ce sujet, avec un paramétrage parfois différent de la situation.

— La page initiale avec la description de notre activité. [10]
— Un déroulé de l'activité telle que proposée à la Maison des Mathématiques et de l'Informatique [11]
— La situation dans une version plus détaillée et contextualisée [12]

10. https://projet.liris.cnrs.fr/lirismed/index.php?id=la-machine-qui-apprend-a-jouer-toute-seule
11. https://mmi-lyon.fr/wp-content/uploads/Deroule_atelier_la_machine_qui_apprend_a_jouer_au_nim.pdf
12. https://culturesciencesphysique.ens-lyon.fr/ressource/IA-Nim-1.xml

— La version de Tangente Education par Florent Madelaine et Malika More (Collectif, 2017) [13]

Dans cette activité, le terme "machine" fait référence à un dispositif matériel non numérique, comme un ensemble de gobelets, de boites, de casiers, qui a pour objectif de simuler le comportement d'un algorithme. Dans cette situation, nous allons construire un dispositif (une machine donc) qui a pour objectif d'apprendre à bien jouer au jeu des bâtonnets (ou encore jeu de Nim). Dans sa version la plus générale, la position de départ de ce jeu est un entier positif N correspondant à la taille d'une pile de bâtonnets. Les règles du jeu sont fournies par un ensemble S d'entiers positifs. A tour de rôle, deux joueurs retirent un nombre x de bâtonnets de la pile, à condition que $x \in S$ et qu'il reste suffisamment de bâtonnets dans la pile. Le premier joueur qui ne peut plus jouer a perdu.

Lorsque $S = \{1, 2\}$, ce jeu correspond au jeu de Fort Boyard, où les deux joueurs retirent 1 ou 2 bâtonnets à tour de rôle et celui qui prend le dernier a gagné.

2.C.a. Description de l'activité

On construit une machine qui va apprendre à bien jouer toute seule au jeu des bâtonnets. L'exemple qu'on donne ci-dessous illustre le cas $N = 8$ et $S = \{1, 2\}$ mais on peut construire une machine pour n'importe quelles autres valeurs de S et N.

Dans sa version grand public, la machine est un objet en bois avec des casiers et des boules de couleurs à l'intérieur, telle qu'illustrée sur la page de notre activité [14]. Si cette machine assez imposante est souvent utilisée pour faire la démonstration de l'activité, nous allons ici en décrire une version mobile qui permet d'être multipliée facilement pour les scolaires. Cette machine est constituée des gobelets numérotés de 1 à N contenant des billes de couleurs, avec $|S|$ couleurs différentes. Chaque gobelet correspond à une position du jeu des bâtonnets. Dans notre exemple, comme $N = 8$, il y a 8 positions de jeu possibles : le gobelet 8 correspond à la position de départ avec 8 bâtonnets, le gobelet 7 à la position de jeu avec 7 bâtonnets, ...

Comme $|S| = 2$, chaque gobelet contient des billes de deux couleurs, disons jaune et rouge. Initialement, on initialise la machine avec autant de billes de chaque couleur (deux sur l'exemple ci-dessous).

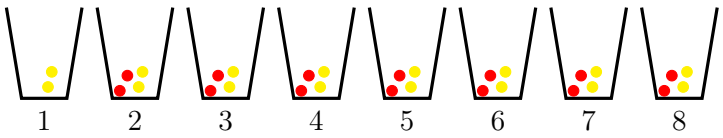

FIGURE 5 – Initialisation de la machine à huit gobelets

La machine va apprendre à jouer en affrontant un joueur humain. Il faut une personne qui joue pour la machine, et une autre qui joue le rôle de l'humain. On considère que la machine joue en premier, car dans la position de départ avec 8 bâtonnets, on sait qu'il existe une stratégie gagnante pour celui qui commence. Quand c'est à son tour de jouer, la personne qui simule la machine regarde dans quelle position de jeu elle se trouve (en comptant le nombre de bâtonnets restant), sélectionne le gobelet correspondant, et tire au sort une bille dans le gobelet (sans en regarder le contenu). Selon la couleur de la bille, elle retire 1 (si c'est jaune) ou 2 (si c'est rouge) bâtonnets. Chaque bille tirée au sort par la machine est posée devant le gobelet d'où elle vient. Le joueur humain joue ensuite son coup, puis la machine recommence et ainsi de suite jusqu'à la fin de partie.

Quand tous les bâtonnets ont été retirés, on regarde qui a gagné :

13. https://www.tangente-education.com/article.php?art=4076&dos=158
14. https://mmi-lyon.fr/wp-content/uploads/Deroule_atelier_la_machine_qui_apprend_a_jouer_au_nim.pdf

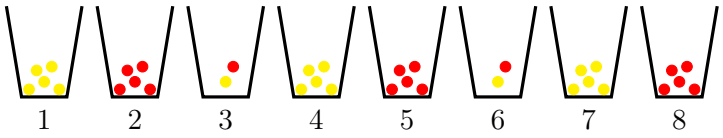

FIGURE 6 – Exemple d'état de la machine après plusieurs parties

— Si c'est la machine : pour chaque bille tirée au sort par la machine, on la remet dans son gobelet et on rajoute une autre bille de la même couleur. Ainsi, à la partie suivante, ce coup aura une probabilité plus forte d'être joué. On dit qu'on le "renforce".
— Si c'est l'humain : chaque bille jouée est écartée, de sorte à diminuer la probabilité de rejouer ces coups à la prochaine partie. Ils sont donc pénalisés.

On enchaîne ensuite des parties. Lorsqu'un gobelet se vide, on le recharge comme au début, avec deux billes de chaque couleur. Au bout d'un certain nombre de parties jouées, on peut constater l'état des gobelets et voir si une stratégie gagnante apparaît.

2.C.b. Analyse de la SMID

Étant donné que la machine joue au hasard au début, elle va perdre souvent. Plus le nombre de parties est grand, meilleur sera l'apprentissage de la machine. Le fait de renforcer ou de punir la machine après chaque simulation aléatoire est le principe de base des algorithmes d'apprentissage par renforcement.

Pour ce jeu avec $S = \{1, 2\}$, on sait que les positions perdantes sont les multiples de 3, et que depuis une position gagnante (non multiple de 3 donc), le coup gagnant consiste à se ramener à une position multiple de 3. Ainsi, la stratégie gagnante vers laquelle la machine devrait converger ressemble à celle de la figure 6.

Les positions perdantes (ici 3 et 6) correspondent à des gobelets qui ont tendance à se vider, car si la machine joue depuis ces positions, elle va perdre face à un humain expert. En pratique, il suffit d'un nombre relativement faible de parties jouées (entre 10 et 20) pour que l'état de la machine ressemble à celui de la figure 6.

Pour les élèves déjà familiers avec le jeu $S = \{1, 2\}$, on peut faire varier l'ensemble de coups autorisés en prenant par exemple $S = \{1, 3, 4\}$, et trois couleurs de billes. Dans ce cas, la stratégie gagnante est beaucoup moins triviale et l'enjeu de découverte de stratégie prend davantage de sens.

Comme pour la plupart des algorithmes d'apprentissage, le paramétrage peut avoir beaucoup d'influence sur la convergence de l'algorithme vers une stratégie gagnante. Dans cette SMID, on peut illustrer cette problématique via plusieurs facteurs qui rendent l'apprentissage de la machine plus ou moins rapide :

— Le niveau de l'humain qui joue. Lorsque l'humain connaît la stratégie gagnante, on peut observer que la convergence de la machine vers la stratégie gagnante est plus rapide. En revanche, lorsque la machine affronte un joueur non expert, on peut alors facilement constater des erreurs d'apprentissage qui ralentissent la convergence.
— Le nombres de billes dans les gobelets au départ. Un trop grand nombre peut réduire la vitesse de convergence, mais un nombre trop faible peut conduire à une réinitialisation trop fréquente des gobelets et donc une situation relativement figée.
— Le nombre de billes que l'on rajoute/enlève quand la machine gagne/perd. Ce nombre est souvent crucial pour les applications basées sur l'apprentissage par renforcement. On peut notamment envisager des renforcements non constants, c'est-à-dire qu'ils dépendent de la position de jeu. Par exemple, une position proche de la fin est souvent plus proche de la stratégie ga-

gnante qu'une position de départ. On aurait donc tendance à rajouter/enlever plus de billes pour ces positions lors de la phase de renforcement/pénalisation.

2.C.c. Vers une IA autonome dans son apprentissage

Quand on confronte cette situation à la pratique du chercheur en informatique, l'écueil principal est le fait de devoir entraîner la machine face à un humain. Les IA les plus performantes s'affranchissent bien entendu de cette contrainte en utilisant des programmes qui jouent contre eux-mêmes. En pratique, la SMID décrite ci-dessus est souvent complétée par les deux mises en situation suivantes :

Jeu contre une autre machine. Avec une classe, on peut répartir les élèves en groupes où chacun.e entraîne sa propre machine dans un premier temps. Puis vient une phase de confrontation entre les machines entre deux groupes. Ainsi, les coups sont joués alternativement par chaque machine et on poursuit l'apprentissage avec les mêmes règles que précédemment. La machine qui gagne voit ses coups joués renforcés, et celle qui perd a ses coups joués pénalisés. Pour que cela soit équitable, on alterne les machines qui commencent la partie. Pour ne pas trop frustrer les participant.e.s, on veillera à rappeler qu'un renforcement négatif (après une défaite) est tout aussi utile à une machine qu'un renforcement positif (victoire).

Jeu contre elle-même. De la même façon, on peut faire jouer une machine contre elle-même. On fait comme s'il y avait deux machines qui jouent avec le même ensemble de gobelets, et on sépare les billes jouées par la machine 1 (qui joue les coups pairs) et la machine 2 (qui joue les coups impairs). A la fin, on regarde la machine qui a gagné, on renforce (resp. on pénalise) les gobelets correspondant aux coups de la machine gagnante (resp. perdante). Ainsi, il y a en moyenne deux fois plus de gobelets qui sont renforcés/pénalisés que lorsque la machine joue contre un humain. La convergence est souvent plus rapide et l'apprentissage ne nécessite pas l'intervention d'un joueur humain.

2.C.d. Notions mises en jeu

Cette SMID traite des algorithmes d'apprentissage par renforcement qui sont à l'origine des avancées récentes de l'intelligence artificielle et de leur percée dans notre quotidien : applications de reconnaissance d'image et de texte, robots conversationnels, voitures autonomes, robots... De manière générale, ils sont très utilisés pour résoudre des problèmes où l'humain n'a pas de bonne stratégie a priori. En l'occurrence, les jeux abstraits à deux joueurs à forte combinatoire sont un excellent terrain d'application pour ce type d'algorithme. On fera remarquer au passage que la façon de jouer de la machine et les phases de renforcement/pénalisation sont très faciles à coder sur un ordinateur.

C'est d'ailleurs sur ce socle que s'appuie le programme *Alpha Go* qui a battu le champion du monde humain du jeu de Go. Ce programme s'est montré le plus efficace lorsqu'il n'a été nourri par aucune connaissance autre que les règles du jeu (programme *Alpha Go zero*), comme c'est le cas dans notre situation avec le jeu des bâtonnets. Ce programme a appris à bien jouer en s'entraînant contre lui-même. Bien entendu, dans l'exemple précis d'Alpha Go et pour la plupart des applications mentionnées plus haut, l'apprentissage par renforcement est couplé à l'utilisation d'un réseau de neurones profond, ce qui explique les performances exceptionnelles de ce type de programmes. Ils ont depuis été adaptés à d'autres jeux comme les échecs, en battant largement le meilleur programme connu jusque là.

3. Vers une étude didactique des situations d'informatique débranchée

Nous pensons qu'il est pertinent de développer des outils conceptuels qui permettent d'analyser les SMID avec rigueur, en vue d'une réappropriation adéquate par les médiateur.ice.s, chercheur.euse.s et enseignant.e.s. Dans cet objectif, les outils conceptuels issus des théories de la didactique (Artigue,

1988 ; Brousseau, 1997 ; Chevallard, 2019) peuvent s'appliquer pour de nombreuses SMID, comme cela a été le cas pour la machine qui joue au jeu de Nim (Esclafit, Modeste, & Saby, 2020). Pour autant, les SMIDs possèdent certaines spécificités propre à la discipline dont certaines peuvent jouer un rôle de variable didactique dans un analyse approfondie de celles-ci. En guise d'illustration, les trois SMID décrites ci-dessus sont à nos yeux représentatives de la variété des activités qui sont menées en informatique débranchée. Tout d'abord, la nature même des activités présentées est diverse. Elles peuvent renforcer certaines notions en informatique connues des programmes scolaires (algorithmique notamment), ou alors faire découvrir aux participant.e.s des problématiques et techniques de la recherche actuelle qui font souvent partie de leur quotidien (intelligence artificielle, algorithmique distribuée dans les exemples précédents). Au niveau du format, il est fréquent que les SMID correspondent à des situations où les participant.e.s sont intégré.e.s au dispositif de médiation (comme c'est le cas pour le réseau de tri). Ce format très immersif est souvent loué dans le processus d'appropriation de la SMID par les élèves. Enfin, la question de la réintroduction de l'ordinateur dans une SMID se pose régulièrement, afin de redonner un contexte concret à l'activité, notamment lorsque ce contexte a pu être mis de coté par le coté immersif de la situation.

En particulier, les trois spécificités des SMID que nous avons identifiées nécessitent à nos yeux une étude didactique ciblée, afin de comprendre en quoi elles peuvent influer sur l'apprentissage et la compréhension des notions mises en jeu.

3.A. Objectifs principaux d'une SMID

Nous constatons une première dichotomie qui relève de la nature même des situations. Certaines SMID ont pour objectif principal de faire découvrir une notion informatique. Bien souvent, il s'agit de notions plutôt complexes mais qui ont un lien avec la recherche actuelle, ou que les participant.e.s peuvent rencontrer dans leur quotidien. Cet enjeu d'éducation aux nouvelles technologies est d'ailleurs propre à ces activités de médiation. Le public est de manière générale très demandeur d'explications à propos du fonctionnement d'outils qui sont omniprésents dans leurs vies. Les situations du réseau de tri et de la machine qui apprend à jouer toute seule ont clairement pour objectif de faire découvrir les notions de tri, d'algorithmique parallèle et d'apprentissage. Pour ces situations, on demande assez peu aux participant.e.s de raisonner, mais plutôt de constater des états après application d'un ensemble de règles, pour ensuite engager une discussion avec les médiateur.rice.s sur les problématiques inhérentes à la situation.

Pour d'autres SMID comme le base-ball multicolore, l'objectif premier est la résolution d'un problème donné. En cela, ces situations se rapprochent davantage des problèmes ouverts et la position de l'apprenant.e de celle du chercheur en informatique. Cependant, la plupart des SMID qui proposent ces activités de recherche concernent des domaines de l'informatique théorique (algorithmique, combinatoire, théorie des graphes). En ce qui concerne les autres domaines de l'informatique, les mises en situation de recherche sont beaucoup moins nombreuses, ce qui nous conduit à nous interroger sur la nature même de la recherche en informatique. Ce type d'étude a déjà été réalisé dans d'autres disciplines comme la physique ou les mathématiques (Hage & Ouvrier-Buffet, 2018). Sur la situation de la machine qui joue au jeu de Nim, on peut ainsi se demander comment créer une SMID qui reflète l'activité du chercheur en intelligence artificielle. Cela devrait se traduire par ce travail sur le paramétrage des algorithmes d'apprentissage, mais la mise en place de telles situations s'avère complexe.

3.B. Place de l'apprenant.e dans une SMID

Dans certaines SMID, les participant.e.s peuvent avoir des positions différentes qui ont une influence sur la façon dont ielles s'approprient et cherchent une résolution au problème posé. Plus concrètement, on peut constater deux types de positionnement pour les SMID : le positionnement dit *grandeur nature* et celui dit *centralisé*. Dans les SMID grandeur nature, les participant.e.s sont considéré.e.s comme

des objets de la situation. Le réseau de tri en fait partie, puisque les participant.e.s jouent eux-mêmes le rôle des données et des opérateurs qui effectuent les comparaisons. Dans le base-ball multicolore, ils jouent le rôle des bases (et aussi celui des balles dans la version grandeur nature un pour tous). Dans la version centralisée, les participants exécutent la situation en ayant une vue globale de celle-ci. Bien souvent, la version centralisée d'une SMID revient à son étude sur une table, avec pour matériel du papier, des jetons à la place des participant.e.s etc.

Lorsque le temps est suffisant, de nombreuses SMID sont organisées pour proposer une version grandeur nature lors du lancement de l'activité et ainsi favoriser l'appropriation et la compréhension du problème, suivie d'une version centralisée, qui tend à davantage placer les élèves dans une démarche de résolution. D'un point de vue informatique, il y a une différence fondamentale entre ces deux approches. La version grandeur nature propose de mettre en lumière les opérations élémentaires et l'aspect local (voire distribué) d'une démarche de résolution. La version centralisée permet de travailler sur l'aspect global et séquentiel qui est le plus souvent rencontré dans l'enseignement de l'informatique.

Sur les situations du base-ball et du réseau de tri, on peut constater cette différence de stratégie et d'objet d'étude principal des participant.e.s en fonction de ce format.

Pour le réseau de tri, la situation initiale est présentée dans sa version grandeur nature, ce qui fait tout d'abord s'interroger sur les opérations élémentaires lors de la constitution d'un algorithme de tri, à savoir les comparaison deux à deux. L'autre enjeu qui est mis en avant est celui de la complexité algorithmique et donc de sa rapidité d'exécution. Pour illustrer ce point de vue, il est fréquent d'orienter la SMID sur la vitesse d'exécution des participant.e.s dans le réseau. Dans un second temps, le réseau de tri peut être proposé aux élèves sur table, notamment pour la construction de réseau optimaux pour $N = 4$. Dans cette version centralisée où chacun.e voit le réseau en intégralité, l'accent est mis sur la conception et la correction des réseaux proposés.

En ce qui concerne le base-ball multicolore, la version standard est grandeur nature où chaque participant.e joue le rôle d'une base. Là encore, cette version induit des stratégies locales et s'apparente à un algorithme distribué où chaque processeur exécute un algorithme en fonction de son voisinage. Ceci impose un processus collaboratif de décision afin d'éviter les blocages (que faire si deux personnes veulent la même balle ?). Cette nécessité de trancher les situations d'accès concurrent est en revanche beaucoup moins immédiate dans la version centralisée où les participant.e.s peuvent construire et tester leur algorithme sur papier. Dans cette version, c'est l'aspect séquentiel qui est mis en valeur et la correction des algorithmes proposés est beaucoup plus facile à appréhender en testant plus facilement différentes configurations de départ.

Les analyses décrites ci-dessus sont le fruit d'expérimentations multiples menées dans différentes classes et par différents chercheur.euse.s, mais il n'existe à l'heure actuelle pas d'étude didactique réelle qui permettrait de valider les effets de cette variable didactique que constitue la place de l'apprenant.e dans la situation. Nous pensons qu'il y a ici un vrai enjeu didactique dans l'analyse de certaines SMID, puisque ces situations grandeur nature font partie des spécificités les plus caractéristiques des SMID.

Au delà des démarches de résolution, un intérêt supplémentaire des situations en grandeur nature est leur côté immersif qui permet aux participant.e.s d'accorder un intérêt souvent immédiat à ce qu'on leur propose.

De manière générale, de nombreuses SMID sont désormais construites autour de ces deux aspects : grandeur nature pour immerger les participants dans la situation, puis passage sur table pour prendre de la hauteur sur la situation. A la Maison des Mathématiques et de l'Informatique à Lyon, une SMID sur les réseaux de neurones [15] vient d'être construite sur ce format. A Terra Numerica, il s'agit d'une

15. https://mmi-lyon.fr/?site_ressource_peda=connecte-tes-neurones

situation sur la coloration de graphes [16].

3.C. Place de l'ordinateur dans une SMID

Dans les situations d'informatique débranchée, l'objectif est de s'abstraire de l'ordinateur pour mieux comprendre les notions abordées. Additionnée au coté souvent immersif des SMID, cette abstraction peut parfois faire perdre le contexte dans lequel la notion s'intègre. Il existe alors un enjeu de *rebranchage* de la situation afin de montrer ou rappeler aux participant.e.s dans quelle cadre les notions qui viennent d'être vues interviennent. Concrètement, cela peut consister à implémenter certains des algorithmes présentés, ou tout simplement expliquer oralement dans quel champ de l'informatique elles interviennent.

Pour certaines SMID, il semble pertinent de s'interroger sur la nécessité de cette phase de rebranchage, et sur la nature de celui-ci. Il se joue de ce que les partipant.e.s retiendront de la situation de médiation qu'ils viennent de vivre.

Si l'on prend l'exemple de la machine qui joue au jeu de Nim, une phase de rebranchage qui permet de montrer une implémentation de l'algorithme semble nécessaire car elle permet de passer à l'échelle en terme de nombre de simulations. On peut utiliser pour cela l'application en ligne disponible en ligne [17]. Par ailleurs, cette application permet de changer le paramétrage de la machine, ce qui permet d'illustrer l'activité de recherche en informatique.

Pour la situation du base-ball, il existe également une application pour y jouer en ligne [18]. Cela dit, cela ne peut pas être considéré comme un vrai rebranchage, dans la mesure où il s'agit simplement de faciliter l'expérimentation de la version centralisée du jeu. La question pourrait en revanche se poser de faire programmer les algorithmes des participant.e.s (dans un langage simple) et de les tester. Il s'agirait là d'une vraie situation de rebranchage et on peut s'interroger sur son intérêt.

4. Des situations d'informatique débranchée en classe

Pourquoi faire de l'informatique débranchée en classe ? Les SMID sont des situations qui ont largement fait leurs preuves dans des contextes de médiation scientifique (dans des centres dédiés comme à la Maison des Mathématiques et de l'Informatique, ou encore dans des salons grand public type Fête de la Science). Elles sont également désormais présentes dans les classes et dans certaines parties des programmes, mais la présence d'un.e chercheur.euse reste souvent requise pour nombre d'entre elles.

Cela dit, dans le cadre de l'apprentissage de l'informatique dans le secondaire, nous voyons quatre raisons principales pour lesquelles cet outil mérite une attention toute particulière :

— Pour rendre les élèves actifs : enseigner la programmation peut être difficile en raison de manque de salles ou de matériel adéquat, ce qui peut conduire à des situations où des élèves partagent un ordinateur et où certain.e.s ne touchent pas au clavier pendant une séance entière.

— Pour réfléchir avant de coder : un problème courant chez les élèves apprenant l'informatique est leur envie immédiate de programmer sans prendre le temps de réfléchir. Cela peut entraîner des résultats incorrects ou illisibles. En supprimant (temporairement) les ordinateurs, les élèves apprennent à structurer leurs idées, à trouver une approche adéquate et à raisonner (Sentance & Csizmadia, 2017).

— Pour prendre du plaisir : l'informatique débranchée peut être pratiquée avec des objets comme du papier, des ciseaux et de la colle, et peut prendre la forme de défis, devinettes ou jeux en

16. https://portail.terra-numerica.org/res/rsrc?all=1
17. https://projet.liris.cnrs.fr/~mam/machine/
18. https://projet.liris.cnrs.fr/~mam/valise/

groupe. L'approche ludique peut captiver les élèves et atténuer l'aspect rébarbatif de certains enseignements traditionnels.

— Pour démystifier le numérique : le domaine souvent mal nommé du "numérique" est si vaste et flou qu'on pourrait presque y inclure tout ce qui implique la contemplation d'un écran. L'informatique ne doit pas être considérée comme une question de consommation de produits numériques ou d'utilisation d'applications magiques. Un des enjeux des SMID est de montrer que l'informatique est une science à part entière, en initiant les élèves à la pensée informatique.

Comment faire de l'informatique débranchée en classe ? De manière générale, il semble important de détecter quelles SMID peuvent être institutionnalisées pour un enseignement scolaire. Un premier travail dans ce sens a été réalisé par (Drot-Delange, 2013). Au vu des programmes, il va souvent d'agir de SMID qui ont trait à l'algorithmique ou au langage binaire. Si la version grandeur nature peut parfois être difficile à adapter en classe, elle apporte une plus-value indéniable dans l'acceptation de la situation par les élèves. Dans un second temps, la version centralisée des SMID est souvent celle qui permet aux élèves de raisonner sur la situation et de proposer des résultats. Au même titre que pour les situations de recherche en classe (Grenier & Payan, 2003), il y a une mise en danger pour les enseignants.e.s dans ce type de situations où les procédures des élèves sont moins cadrées qu'à l'habituel, et où les pistes de recherche peuvent être variables et sans réponse connue a priori.

Certains IREM comme celui de Grenoble [19] proposent des SMID avec des fiches détaillées à destination des enseignant.e.s et des élèves. Dans certains cas, on trouvera un séquencement précis qui permet de donner tous les outils aux enseignant.e.s pour une utilisation en autonomie. Sur le base-ball multicolore, on peut également mentionner un retour d'enseignant fait à Nancy avec des analyses de production d'élèves [20].

5. Conclusion

A travers l'exemple de trois situations d'informatique débranchée représentatives et éprouvées, nous avons identifié trois paramètres spécifiques des SMID qui nous semblent prometteurs en vue d'une analyse didactique globale de celles-ci. En particulier, les propriétés grandeur nature et de rebranchage présentent un intérêt évident dans la façon dont les élèves peuvent s'approprier une notion. Si le déploiement en classe des SMID n'en est encore qu'à ses balbutiements, une analyse didactique plus approfondie devrait aider les chercheur.euse.s impliqué.e.s dans ces situations à proposer une cartographie détaillée de ces activités, et au même titre que pour l'enseignement de l'informatique à l'école, donner un crédit supplémentaire à l'introduction de ces activités en classe.

Références

Académie des Sciences, . (2013). *L'enseignement de l'informatique en france, il est urgent de ne plus attendre (rapport 513).*

Althuser, M., Brasset, N., Rasse, A., Vincent, J.-M., & Wack, B. (2019). Le base-ball multicolore : pensée algorithmique et raisonnement. *IREM de Grenoble*. Consulté sur https://irem.univ-grenoble-alpes.fr/medias/fichier/article-baseball_1587997230572-pdf

Artigue, M. (1988). Ingénierie didactique. *Recherches En Didactique Des Mathématiques*, *9(3)*, 281–308.

Batcher, K. E. (1968). Sorting networks and their applications. In *Proceedings of the april 30–may 2, 1968, spring joint computer conference* (pp. 307–314).

Bell, T., & Lodi, M. (2019). Constructing computational thinking without using computers. *Constructivist Foundations*, *14(3)*, 342–351.

19. https://irem.univ-grenoble-alpes.fr/recherche-action/themes/informatique-de-l-ecole-jusqu-au-lycee-417052.kjsp

20. members.loria.fr/MDuflot/files/med/doc/Baseball/NavettesSpatiales2nde.pdf

Bell, T., & Vahrenhold, J. (2018). Cs unplugged—how is it used, and does it work ? In H.-J. Böckenhauer, D. Komm, & W. Unger (Eds.), *Adventures between lower bounds and higher altitudes : Essays dedicated to juraj hromkovič on the occasion of his 60th birthday* (pp. 497–521). Cham : Springer International Publishing. doi: 10.1007/978-3-319-98355-4_29

Bell, T., Witten, I. H., & Fellows, M. R. (1998). Computer science unplugged : off-line activities and games for all ages..

Brousseau, G. (1997). *Theory of didactical situations in mathematics – 1970-1990.* Dordrecht : Kluwer Academic Publishers.

Bundala, D., & Závodný, J. (2014). Optimal sorting networks. *Language and Automata Theory and Applications*, 236-247.

Caillot, M. (1997). Baron (g.-l.), bruillard (e.). — l'informatique et ses usagers dans l'éducation. *Revue française de pédagogie*, *120*(1), 153–154. Consulté sur https://www.persee.fr/doc/rfp_0556-7807_1997_num_120_1_3006_t1_0153_0000_2 (Included in a thematic issue : Penser la pédagogie)

Chevallard, Y. (2019, 06). On using the atd : Some clarifications and comments. *Educação Matemática Pesquisa : Revista do Programa de Estudos Pós-Graduados em Educação Matemática*, *21*. doi: 10.23925/1983-3156.2019v21i4p001-017

Collectif. (2017). *L'informatique sans ordinateur* (Vol. 42-43 ; Pole, Ed.). Tangente Education, éditions POLE.

Dowek, G. (2011, octobre). Les quatre concepts de l'informatique. In G.-L. Baron, É. Bruillard, & V. Komis. (Eds.), *Sciences et technologies de l'information et de la communication en milieu éducatif : Analyse de pratiques et enjeux didactiques.* (p. 21-29). Patras, Greece : Athènes : New Technologies Editions. Consulté sur https://edutice.archives-ouvertes.fr/edutice-00676169

Drot-Delange, B. (2013, août). Enseigner l'informatique débranchée : analyse didactique d'activités. In *AREF* (p. 1-13). France. Consulté sur https://archivesic.ccsd.cnrs.fr/sic_00955208 (ACTI Axe1)

Esclafit, P., Modeste, S., & Saby, N. (2020). A nim like game and a machine that plays it : a learning situation at the interface of mathematics and computer science. *Teaching Mathematics and Computer Science*, *18*, 317-326.

Godot, K. (2005). *Situations recherche et jeux mathematiques pour la formation et la vulgarisation. Exemple de la roue aux couleurs.* (Theses, Université Joseph-Fourier - Grenoble I). Consulté sur https://theses.hal.science/tel-00102171

Grenier, D., & Payan, C. (2003). Situation de recherches "en classe" : essai de caractérisation et proposition de modélisation. In A. IREM de Paris (Ed.), *Actes du séminaire national de didactique des mathématiques 2002.* (p. 201-203).

Hage, S. E., & Ouvrier-Buffet, C. (2018, novembre). Les démarches de chercheurs en physique et en mathématiques. Enjeux didactiques d'une nouvelle approche épistémologique. *Recherches en éducation*(34). Consulté sur https://hal.science/hal-03356685 doi: 10.4000/ree.1932

Meyer, A., & Modeste, S. (2022, mai). Situation didactique autour d'un jeu de recherche : expérimentation en classes de NSI. In *L'informatique, objets d'enseignement et d'apprentissage. Quelles nouvelles perspectives pour la recherche ?* (p. 100-112). Le Mans, France. Consulté sur https://hal.science/hal-03697943

Modeste, S. (2012). *Enseigner l'algorithme pour quoi ? Quelles nouvelles questions pour les mathématiques ? Quels apports pour l'apprentissage de la preuve ?* (Theses, Université de Grenoble). Consulté sur https://theses.hal.science/tel-00783294

Parberry, I. (1991). On the computational complexity of optimal sorting network verification. In E. H. L. Aarts, J. van Leeuwen, & M. Rem (Eds.), *Parle '91 parallel architectures and languages europe* (pp. 252–269). Berlin, Heidelberg : Springer Berlin Heidelberg.

Pea, R. D. (1987). *Logo Programming and Problem Solving.* Consulté sur https://telearn.archives-ouvertes.fr/hal-00190546 (technical report, 10 pages)

Sentance, S., & Csizmadia, A. (2017, mars). Computing in the curriculum : challenges and strategies from a teacher's perspective. *EDUCATION AND INFORMATION TECHNOLOGIES*, *22*(2), 469–495. doi: 10.1007/s10639-016-9482-0

Eric Duchêne
Univ Lyon, CNRS, INSA Lyon, UCBL, Centrale Lyon, Univ Lyon 2, LIRIS, UMR5205, Bâtiment Nautibus, 43 Bd du 11 novembre 1918
F-69622 Villeurbanne Cedex, France
e-mail: eric.duchene@univ-lyon1.fr

Aline Parreau

Univ Lyon, CNRS, INSA Lyon, UCBL, Centrale Lyon, Univ Lyon 2, LIRIS, UMR5205, Bâtiment Nautibus, 43 Bd du 11 novembre 1918
F-69622 Villeurbanne Cedex, France
e-mail: aline.parreau@univ-lyon1.fr