La conception préliminaire de Dance Your Algorithm, un langage de modélisation-en-action

The early design of DYA, a modelling-in-action language

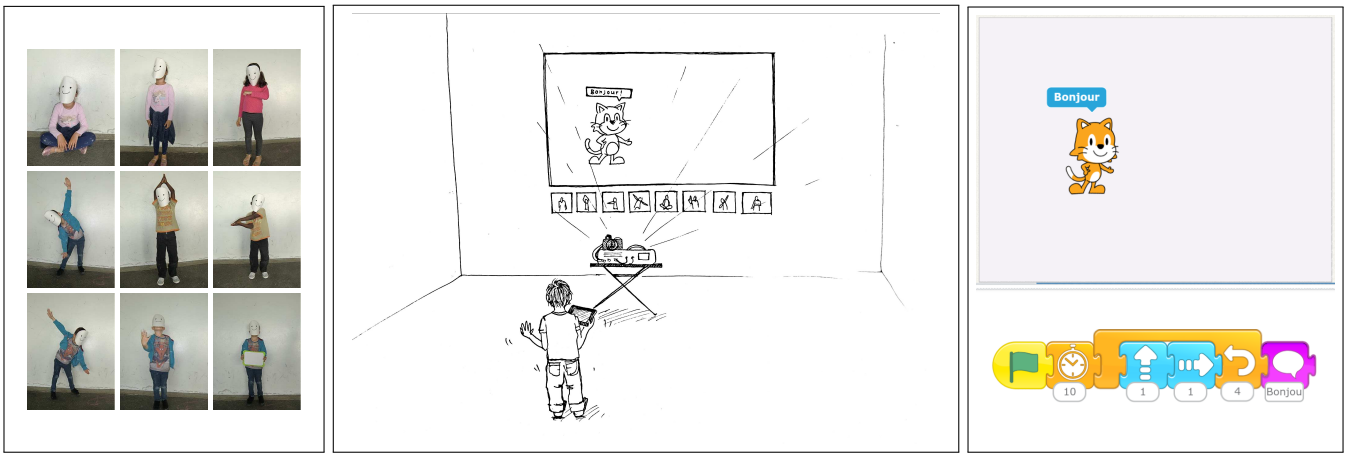

Figure 1: à gauche le programme dansé, au milieu le système en cours de développement, à droite l'équivalent en Scratch Jr

ABSTRACT

Novice programmers are faced with many difficulties. One of them is the correct understanding of the notional machine, an idealized computer whose properties are implied by the constructs in the programming language employed. Another significant difficulty is the difference between program comprehension and program writing. To address these issues, we propose the idea that novices dance algorithm(s). The "Dance your algorithm (DYA)" system makes the assumption that dancing her algorithm in time and space allows the user to reify the notional machine and helps to write programs. Technically speaking, because a dance is one of the many executions of the same program, the DYA environment under construction should gather different executions and should compose the corresponding program in a Scratch Jr program. Early design has been made using the Moving and Making Strange methodology proposed by Loke and Roberston, an approach to movement-based interaction design that recognizes the central role of the body and movement in lived cognition.. Because DYA performs roundtrips between executing a program and designing a program, the duality is addressed with the mover and the observer perspectives of the methodology. Within both perspectives, a faceted analysis is used to present early design issues and choices, and the impact on the notional machine.

CCS CONCEPTS

• **Social and professional topics** → **Computing education**; **K-12 education**; • **Software and its engineering** → **Design languages**;

KEYWORDS

Performing arts, gesture recognition, inductive inference, novice programmers

RÉSUMÉ

Les programmeurs novices doivent faire face à de nombreuses difficultés. L'une d'entre elles est la compréhension correcte de la machine notionnelle, un ordinateur idéalisé dont les propriétés sont des conséquences des capacités du langage de programmation utilisé. Une autre difficulté majeure est la différence entre la compréhension d'un programme et l'écriture d'un programme. Pour résoudre ces questions, nous proposons que les programmeurs novices dansent leur(s) algorithme(s). Le système "Dance your algorithm" (DYA) repose sur l'hypothèse que danser son algorithme aide l'utilisatrice ou l'utilisateur à réifier la machine notionnelle et à écrire

des programmes. D'un point de vue technique, puisqu'une danse est l'une des nombreuses exécutions possibles pour un programme, l'environnement DYA, en construction, doit assembler les différentes exécutions et doit les composer en un programme Scratch Jr. La conception préliminaire a été réalisée en utilisant la méthode Moving and Making Strange proposée par Loke et Roberston, une approche des interactions basées sur le mouvement qui reconnait le rôle central du corps et du mouvement dans la cognition vécue. Parce que DYA fait des allers-retours entre l'exécution et la conception d'un programme, cette dualité est traitée avec deux perspectives de la méthode : le danseur et l'observateur. Dans ces deux perspectives, une analyse par facette est utilisée pour présenter les attendus et les choix de conception préliminaires et leurs impacts sur la machine notionnelle.

MOTS-CLEFS

Art, reconnaissance de geste, inférence, programmeurs novices

1 INTRODUCTION

Cet article présente les choix de conception préliminaire du langage DYA - Dance Your Algorithm(s) - et de son environnement de programmation, basés sur l'expression corporelle d'algorithmes. Nous employons la méthode "Moving an Making Strange" proposée par Loke et Robertson [19], une approche conçue pour rendre claire la trajectoire entre la perspective du danseur (en train d'évoluer), la perspective de l'observateur (tel que le concepteur du mouvement ou le danseur en train de réfléchir à sa conception), et la perspective de la machine (étant capable de détecter et d'interpréter correctement les mouvements) [19]. Nous envisageons le système ainsi : différentes interprétations dansées du même programme sont filmées et leurs mouvements sont reconnus ; les interprétations sont traduites dans une version équivalente à un programme Scratch Jr ; le programme Scratch Jr est exécuté dans son environnement. Notre recherche porte l'attention sur l'écriture d'un programme à partir de ses différentes exécutions dansées, ce qui conduit le programmeur novice à modéliser son algorithme au travers de performances corporelles, d'où le terme modélisation-en-action utilisé dans le titre de cet article.

Une des difficultés d'apprendre la programmation à un novice est de décrire, avec un niveau approprié de détail, la machine qu'il ou elle est en train d'apprendre à contrôler. Du Boulay définit la machine notionelle (*notional machine*) comme un ordinateur conceptuel et idéalisé dont les propriétés découlent des constructions du langage de programmation employé [4]. Une autre difficulté provient de la différence entre les connaissances et les stratégies de résolution de problèmes. Davies [7] fait la distinction entre la connaissance de la programmation (qui est de nature déclarative, i.e. être capable d'établir comment une boucle "for" fonctionne) et les stratégies de programmation (la manière dont la connaissance est utilisée et mise en application, i.e. utiliser une boucle de manière appropriée dans un programme). Cette distinction, lorsqu'elle est envisagée dans une perspective globale, conduit à une difficulté significative soulignée par Robins [26], qui est la différence entre la compréhension de programmes (étant donné le texte d'un programme, on doit comprendre comment il fonctionne) et la génération de programmes (où on doit créer tout ou partie d'un programme pour réaliser une tâche ou résoudre un problème).

Deux des questions de recherche envisagées dans le projet DYA et traitées dans cet article sont la conception d'une version simple mais extensible d'une machine notionelle pour des jeunes novices et l'assistance à l'écriture de programmes pour la machine notionelle. Pour qu'une stratégie basée sur une machine notionelle soit efficace, du Boulay énonce deux principes importants : "Premièrement, la machine notionelle employée doit être conceptuellement simple, et deuxièmement, on doit fournir au novice des méthodes pour observer une partie du fonctionnement de la machine en action [4]." Pour faciliter la mémorisation et la compréhension de la machine notionelle et des constructions de programmation qui y sont associées, nous utilisons l'effet d'énaction. "L'effet d'énaction fait référence aux performances supérieures de la mémoire sur les évenements énactés par rapport aux événements non-énactés [22]." Le premier principe guidant notre recherche est de fournir aux programmeurs un environnement (et un langage) où elles et ils peuvent énacter leurs programmes. Winslow rapporte que les études ont montré qu'il y a très peu de relation entre la capacité à écrire des programmes et la capacité à en lire [29]. Une recommandation majeure qu'on trouve dans la littérature est que l'instruction doit se concentrer non seulement sur l'apprentissage des caractéristiques d'un nouveau langage, mais aussi sur la combinaison et l'utilisation de ces caractéristiques, spécialement les problèmes sous-jacents à la conception basique de programmes.

Une autre suggestion importante est de traiter les différents modèles mentaux qui sous-tendent la programmation. "Les modèles sont cruciaux pour construire une compréhension. Les modèles de contrôle, les structures de données et la représentation des données, la conception de programmes et le domaine du problème sont d'égale importance. Si l'instructeur les omets, les étudiant.es vont construire eux-mêmes et elles-mêmes des modèles de qualité douteuse [29]." Mayer a montré que les étudiant.es qui ont eu un modèle de machine notionelle (que Mayer appelle un "modèle concret") avaient une performance de résolution de problèmes supérieure à ceux ou celles sans modèles [20]. Mayer a montré aussi, comme on peut le prévoir à partir de la littérature générale sur l'éducation, que les étudiant.es qui sont encouragé.es à s'engager activement et à explorer les informations relatives à la programmation (en paraphrasant / reformulant avec leurs propres mots) réussissent mieux la résolution de problèmes et le transfert créatif [20].

Combiner les approches "apprendre en faisant" et "apprendre par l'exemple" constitue le second principe de conception du langage et de son environnement. Green [10] suggère que la programmation, plutôt qu'être considérée comme une "transcription d'une représentation interne retenue" ou dans le

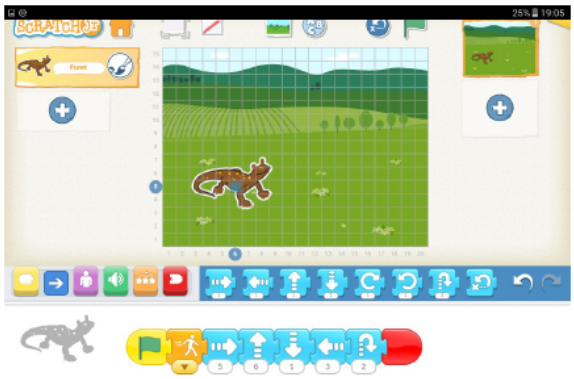

Figure 2: Aperçu de Scratch Jr

contexte de la "théorie pseudo-psychologique de la 'programmation structurée'", devrait être un processus exploratoire où les programmes sont créés "de manière opportuniste et incrémentale". Une conclusion similaire est émise par Visser [28] et par Davies : "les modèles émergents du comportement en programmation suggèrent un processus incrémental de résolution de problèmes dont la stratégie est déterminée par des épisodes localisées de résolution de problèmes et des ré-évaluations fréquentes du problème [7]." Une emphase sur l'exploration opportuniste semble particulièrement appropriée lorsqu'il s'agit de programmation par des novices [26]. D'où l'idée que l'exécution de programmes précède la conception de programmes ; i.e. de partir des différentes exécutions d'un même programme pour arriver à composer ce programme. Nous présentons le background et les travaux connexes dans la section 2, nous utilisons une analyse par facettes pour présenter la perspective du danseur dans la section 3, nous survolons des questions générales de la perspective de l'observateur dans la section 4, et nous concluons et évoquons les problèmes ouverts en section 5 .

2 BACKGROUND ET TRAVAUX CONNEXES

2.1 Scratch Jr

D'après Brennan et Resnick, créateurs du langage Scratch, lorsque des jeunes conçoivent des applications multimédia interactives avec Scratch, ils et elles mobilisent des concepts de la pensée informatique. Ces concepts sont communs à de nombreux projets Scratch et sont transférables dans d'autres contextes de programmation ou dans d'autres domaines ; ce sont les séquences, les boucles, le parallélisme, les conditionnelles, les opérateurs et les données [5]. Scratch Jr est un environnement de programmation inspiré de Scratch et retravaillé pour les enfants de maternelle. Les concepts sont plus simples : séquence, boucle, parallélisme, événement, message. Un script est une suite de blocs et appartient à un lutin (*sprite*).

Dans la figure 1, l'animal est la représentation d'un lutin et son script démarre avec le drapeau vert, finit avec un bloc rouge, et réalise une série de déplacements. Un lutin est comme un fil d'exécution (*thread*)) et les scripts, lorsqu'ils appartiennent à des lutins différents, expriment un programme concurrent. Nous avons choisi Scratch Jr par ce qu'il est destiné aux enfants qui ne savent pas lire ; et donc les blocs ont une représentation graphique qui est représentative de la signification du bloc et nous utilisons, autant que faire se peut, les éléments graphiques de Scratch Jr dans le langage corporel.

2.2 La méthode Moving and Making Strange

Loke et Robertson [19] propose une méthode de conception "Moving and Making Strange", une approche pour la conception d'interactions basées sur le mouvement qui reconnait le rôle central du corps et du mouvement dans la cognition vivante. La méthode de conception propose un ensemble de principes, perspectives, méthodes et outils pour concevoir et évaluer des interactions basées sur le mouvement avec la technologie. La méthode est structurée selon les trois perspectives du danseur, de l'observateur et de la machine. Un diagramme de sept activités liées entre elles est présenté dans la figure 3 de [19]. Les méthodes et outils sur lesquels s'appuie la perspective du danseur comprennent un ensemble de techniques pour expérimenter et ré-énacter le mouvement, les activités sont : 1. investigation du mouvement ; 2. invention et chorégraphie du mouvement ; 3. ré-énaction du mouvement. Le danseur peut aussi être en position d'observateur de ses propres mouvements, par exemple, pendant l'examen de mouvements enregistrés. Les méthodes et outils sur lesquels s'appuie la perspective de l'observateur travaillent avec un ensemble de représentations du mouvement observé, les activités sont : 4. description et documentation du mouvement ; 5. analyse visuelle et représentation du mouvement. La perspective de la machine se concentre sur la perception et l'interprétation du mouvement par l'ordinateur. Les méthodes et les outils sur lesquels s'appuie cette perspective facilitent un examen proche et détaillé de la correspondance et de l'interprétation des mouvements du corps comme entrées d'un dispositif technologique interactif, les activités sont : 6. exploration et mise en correspondance des interactions homme-machine ; 7. représentation des entrées en machine et interprétation des corps en mouvement.

2.3 Analyse par facette

L'analyse par facette examine les caractéristiques d'un problème et les regroupe en catégories pour créer un état des connaissances, en procédant du particulier vers le général. LaBarre la définit ainsi : "La technique d'analyse par facette repose sur l'utilisation d'une liste provisionnelle de catégories fondamentales qui sont composées de concepts, sujets ou termes qui permettent l'analyse d'un domaine donné ou d'un ensemble d'un objets en facettes : composants de base, attributs, caractéristiques et fonctions [17]." L'analyse par facette, et la classification par facettes qui lui est liée, sont communément associées à S. R. Ranganathan, un mathématicien et bibliothécaire indien. Ranganathan a développé une

théorie de l'*univers des sujets*, inspirée des travaux de Cantor et en relation avec l'idée d'infini : les sujets existent dans un espace multi-dimensionnel. Ranganathan a nommé facettes de tels ensembles infinis de sujets et il a décrit deux sortes de facettes : - sujets qui n'ont pas d'idées isolées (*isolates*), qui sont des sujets de base (i.e. Mathématiques) ; - qualification des sujets de base, qui sont appelées *isolates* (i.e. périodes ou pays). Ranganathan [24] a proposé au départ cinq catégories fondamentales - Personnalité, Matière, Energie, eSpace and Temps (PMEST). Hjorland [14] rapporte qu'il trouva que cinq sortes de facettes *isolate* sont nécessaires et suffisantes pour caractériser tous les documents produits et futurs : "- Personnalité représente les caractéristiques distinctes d'un sujet ; - Matière comprend les matériaux constituants qui peuvent composer un sujet ; - Energie est n'importe quelle action qui peuvent survenir pour le sujet ; - Espace est le composant géographique des localisations du sujet ; - Temps couvre les périodes associées au sujet [14].

2.4 Les Filles Qui...

"Les Filles Qui..." constituent un collectif d'étudiantes de notre université dont l'objectif est double : montrer l'exemple des sciences au féminin, et développer la pratique des sciences, de la technologie et de la programmation dans les écoles primaires de notre zone géographique. L'initiative des "filles qui..." opère sous le couvert de deux dispositifs français, "Accompagnement en Sciences et Technologies à l'École Primaire" - ASTEP (Éducation Nationale) et "Savanturiers - école de la recherche" (Centre de Recherche Interdisciplinaire, Paris). La plupart des "filles qui..." sont des étudiantes de licence (en Lettres ou en Sciences) qui animent des séances ASTEP d'apprentissage de la programmation avec Scratch Jr, Scratch ou mBlock (version de Scratch pour robots éducatifs mbot). Certaines "filles qui..." sont des étudiantes en master et en doctorat d'informatique qui accompagnent les projets Savanturiers comme mentors scientifiques. Lors de l'année scolaire 2017-2018, 9 classes ont été bénéficiaires de cours en Scratch Jr, 12 classes en Scratch, 5 classes en robots, soit 26 classes et 491 élèves de primaire au total. Nous avons acquis de l'expérience sur l'apprentissage des jeunes programmeurs novices et DYA a été conçu comme une tentative de réponse aux difficultés rencontrées par certains élèves.

2.5 Postulats initiaux

La danse et le corps sont mobilisés comme vecteur de compréhension des concepts de programmation véhiculés dans Scratch Jr. Nous avons à mettre en correspondance le temps, l'espace, les actions et les objets de Scratch Jr avec les choses qui peuvent être et se produire lorsqu'on danse. Nous avons commencé la conception avec les postulats suivants. Premièrement, le corps représente les différents blocs de Scratch Jr ; chaque bloc est associé à une forme différente du corps. Les blocs qui sont liés au même lutin partagent le même accessoire (i.e. un accessoire d'une certaine couleur ou un certain type de couvre-chef). Ainsi des couleurs ou des accessoires différents indiquent la présence de plusieurs lutins. Deuxièmement, la scène de danse est l'endroit où les enfants dansent leurs algorithmes, cela correspond en Scratch Jr à la partie de l'écran où les enfants construisent et éditent leurs scripts composés des blocs. Mais nous avons aussi besoin de représenter la scène de Scratch Jr, là où l'exécution des scripts se passe, i.e. les arrières-plans, les mouvements, les sorties. Dans Scratch Jr, cette scène est matérialisée par une partie de l'écran au-dessus de la zone d'édition des scripts. L'équivalent de l'écran de Scratch Jr, comprenant la zone d'affichage du programme et la zone d'exécution, sera projeté orthogonalement à la scène de danse. Il se pose deux problèmes d'orientation : les directions des mouvements et la verticalité/horizontalité. Pour le premier problème, nous avons considéré que le danseur danse comme si elle ou il était dans un jeu de tir à la première personne (*First Person Shooter - FPS*), c'est-à-dire en vue subjective où le danseur voit la scène à travers les yeux du lutin. Pour le deuxième problème, nous avons simplifié la question en considérant que lorsque le danseur voit les déplacements du lutin sur un écran vertical, elle ou il programme d'abord puis exécute ensuite son programme qui, précisément, s'exécute sur cet écran vertical où se passent, par exemple, les déplacements des lutins [1]. Mais si on décide de danser une exécution de programme, le danseur évolue sur une scène de danse et l'exécution se place sur un plan horizontal : elle est alors matérialisée par le danseur et peut être restituée au moyen d'un robot programmable [2]. Troisièmement, dans une version plus évoluée de DYA, la scène doit aussi représenter les variables (Scratch Jr n'utilise pas de variables et ce point sera donc traité ultérieurement mais on a fait une "provision de design" dans ce but). Quatrièmement, si plusieurs enfants dansent en même temps sur la scène, on fait l'hypothèse que le paradigme "naturel" de Scratch / Scratch Jr est employé : il y a une exécution concurrente des lutins, synchrone ou asynchrone cf. 4.1, avec la possibilité pour un lutin de tamponner un autre lutin ce qui peut déclencher la danse de ce deuxième lutin ainsi que la capacité de se synchroniser par échange de messages.

2.6 Conception incrémentale et appropriée

Scratch et Scratch Jr ont été développé par une succession de phases de recherche, de développement et d'observation et nous suivons la même approche pour la conception et le développement du langage corporel.

2.6.0 Essais préliminaires. Les gestes correspondant aux blocs ont été testés en maternelle pendant les séances des "filles qui...".

2.6.1 Proposition initiale. La première proposition du langage a été conçue avec l'aide du chorégraphe.

2.6.2 Premier essai en vrai grandeur. La première proposition sera testée pendant un stage d'été d'une semaine organisé par un centre socio-éducatif partenaire dans

1. la flèche haut fait se déplacer le lutin vers le haut, la flèche droite vers la droite, etc.
2. la flèche haut fait aller en avant, la flèche bas en arrière, etc.

un quartier prioritaire de notre ville. Le thème du stage est "Danse et programmation". Les jeunes enfants et teenagers apprennent Scratch Jr ou Scratch le matin, et dansent leurs programmes l'après-midi. Les algorithmes dansés sont traduit manuellement par une équipe d'étudiant.es de notre université. Le chorégraphe anime les sessions de danse.

2.6.3 Deuxième version Une seconde version du langage corporel est retravaillée à partir des observations issues du stage d'été et de l'état de développement du système DYA.

2.6.4 Deuxième essai en vrai grandeur. La seconde proposition sera utilisée à partir de Septembre dans un collège d'enseignement avec des jeunes présentant des difficultés d'apprentissage scolaire (section d'enseignement général et professionnel adapté – SEGPA). Les leçons de danse et de programmation auront lieu sur une période de 6 semaines, à raison d'une demi-journée par semaine. Le chorégraphe anime les sessions de danse. Selon la maturité du système DYA, les algorithmes dansés seront traduit automatiquement ou encore manuellement par les étudiant.es.

2.6.5 DYA 1.0 A partir de toute l'expérience acquise, nous construirons la version du langage qui sera utilisée lors de la prochaine année scolaire.

3 FACETTES DE LA PERSPECTIVE DU DANSEUR

Une danse est réalisée au travers d'un langage corporel d'exécution, et ça peut être aussi la trace d'exécution d'un programme - une chorégraphie - qui est écrit dans un langage de composition. Langage d'exécution et langage de composition sont liés mais servent des buts différents. Mayerhofer et al. [21] affirment que l'utilisation de traces provenant de modèles (et des traces en général) peuvent servir de modèle d'exécution ce qui permet de raisonner sur l'exécution, et facilite l'analyse dynamique et l'adaptation. L'utilisation ne diffère pas si la chorégraphie est écrite d'abord, puis dansée et son exécution enregistrée. Et dans le cas inverse, les danses sont exécutées, enregistrées et reconnues, et la chorégraphie (le programme) doit être inférée des exécutions.

Comme mentionné au préalable, la méthode "Moving and Making Strange" offre trois perspectives : l'exécution se place dans la perspective du danseur et la composition dans la perspective de l'observateur. Évidemment ce qui se passe dans ces deux perspectives a un impact sur la troisième, celle de la machine, car si, au bout du compte, le mouvement sert d'entrées de commande d'un système informatique, la machine doit être capable de détecter et d'interpréter correctement le mouvement. Considéré pour une analyse par facette, l'univers des sujets d'un danseur est différent de (bien que relié à) l'univers des sujets d'un observateur. Réaliser une analyse par facette procède généralement du particulier (les caractéristiques des sujets) vers le général (les questions communes à tous les univers) et, dans cette section, nous présentons l'analyse de l'univers du danseur dans l'ordre PMEST.

3.1 Personnalité et matière

Les sujets (les danseurs) peuvent être actifs ou passifs. Avec UML, les classes actives sont des classes qui représentent des flots indépendants de contrôle. La sémantique des classes actives spécifie que, quand une telle classe est instanciée, le nouvel objet commence l'exécution de son comportement en tant que conséquence de sa création [11]. Quand DYA est lu au singulier comme "Danse ton algorithme - Danse Your Algorithm", cela signifie que la machine notionelle est un système séquentiel : il y a un seul flot de contrôle, i.e. un seul sujet est actif à la fois. Quand DYA est lu au pluriel comme "Danse tes algorithmes - Danse Your Algorithms", cela évoque une machine notionelle concurrente. Cependant, dans nos premiers essais, chaque danseur exécute son programme "personnel" et personne n'a conçu d'exécution concurrente, et au cas où une interaction entre lutins existe(comme un lutin qui en tamponne un autre), c'est considéré comme l'exécution successive de deux machines séquentielles plutôt que l'exécution de deux lutins dans une machine concurrente.

Un sujet est un tout fait de parties constituantes (matière). Fondamentalement, un sujet a deux sortes de parties : instructions et données, donc il pourrait y avoir deux sortes de danseurs, les danseurs-instructions et les danseurs-données. Les gestes d'un danseur-instruction ou d'un ensemble de danseurs-instructions représente le programme, son exécution traverse les différentes instructions et le geste en cours peut matérialiser où en est le flot de contrôle, comme une sorte de compteur ordinal vivant. Quand les danseurs-instructions sont associées à des valeurs, ils peuvent être accompagnés de danseurs-valeurs ou porter eux-mêmes les valeurs. Dans la version actuelle de DYA et par simplicité, les danseurs représentent les valeurs dont ils ont besoin soit avec leurs doigts, soit en les écrivant sur une ardoise magique. Les valeurs sont donc - approximativement - des paramètres des instructions et peuvent être vues comme des parties d'un danseur-instruction (e.g. le nombre de tours d'une boucle ou la durée d'une action). Cependant, les décisions conernant la représentation des données sont toujours ouvertes. Dans le projet AlgoRythmics, "a technologically and artistically enhanced multi-sensory computer-programming education [16]", les danseurs jouent le rôle des données à trier. Chaque danseur porte une valeur sur ses vêtements. Dans un algorithme de tri, il y a deux "instructions" de base : 1. comparer deux nombres, 2. échanger deux nombres. Un pas de danse est associé à chaque "instruction". L'ensemble de la performance dansée permet de comprendre l'exécution du programme, par exemple dans le cas d'un tri à bulle, chaque traversée de l'ensemble des danseurs-données déplace au moins un danseur-donnée à sa position finale. Il est donc envisagé dans les versions ultérieures de DYA de représenter certaines données, dont les variables, par des danseurs.

3.2 Énergie

La facette Énergie traite de la manière dont le sujet change, est traité, évolue ... Dans des classifications par facette plus détaillées, la catégorie fondamentale Énergie donne lieu à plusieurs sous-catégories : Action, Opération, Processus [17]. La facette Énergie communique sur le sens et l'exécution des instructions. Scratch Jr est un langage de programmation pour des novices qui n'ont pas l'âge de lire, le jeu d'instructions est simple : valeurs constantes, événements de départ et d'arrêt, déplacements paramétrés, dire (sortie), apparence (grandir/rétrécir/disparaitre/apparaitre), son (enregistrer un son/jouer un son enregistré), attendre, répéter, répéter indéfiniment, envoyer/recevoir un message. Chaque instruction est associé à un geste de danse. La figure 3 représente la version dansée du programme Scratch Jr de la figure 2. Une difficulté de représentation concerne les mouvements. Dans Scratch Jr, un bloc de déplacement vers le haut/bas fait monter/descendre le lutin sur l'axe vertical et les blocs gauche/droite déplacent le lutin sur l'axe horizontal. Comme mentionné dans la section 2.5, les gestes capturés par la caméra sont projetés dans la partie inférieure de l'écran dédiée au programme et ces gestes représentent les mouvements du lutin sur sa scène d'exécution, située dans la partie supérieure de l'écran de projection. Au départ, les enfants se déplaçaient réellement sur la scène, qui est horizontale, mais ils étaient embarrassés et se trompaient dans les directions. Aussi nous avons décidé que les enfants ne se déplaceraient plus et qu'ils et elles auraient les mêmes verticalité et horizontalité que la scène d'exécution des lutins, un déplacement du lutin vers le haut/bas/droite/gauche correspond à un geste de l'enfant vers le haut/bas/droite/gauche. Une autre difficulté de représentation concerne la matérialisation du flôt de contrôle. Tamponner un autre danseur ou bien lui envoyer un message peut déclencher une réponse comportementale de l'autre lutin - une sorte d'appel de procédure sans retour. Dans certains cas, si la machine notionelle est séquentielle, le sujet de départ devient passif et le contrôle est transféré à l'autre sujet qui devient actiof, exécute sa propre danse, et éventuellement interagit avec le premier lutin et lui redonne le contrôle. On peut aussi considérer que la machine notionelle permet l'exécution concurrente de danseurs comme dans Scratch Jr.

3.3 Espace

La scène de danse matérialise la mémoire de l'ordinateur. Pour une machine notionelle représentant un ordinateur simple et simplifié, il y a principalement deux types de segment de mémoire : les segments de code (contenant du code source ou exécutable) et les segments de données (contenant des variables et des constantes). Tomorrow Corporation (https ://tomorrowcorporation.com/) est une petite entreprise de jeu qui a développé un jeu appelé "Human resource machine". Le joueur doit programmer un petit employé de bureau à l'aide de onze commandes (qui sont équivalentes à des instructions en assembleur). Il y a une boîte d'entrée et une boîte de sortie (les entrées-sorties) et quelques emplacements sur le sol pour stocker des affaires (la mémoire). L'employé de bureau peut porter une seule boîte à la fois (comme un accumulateur). Les boîtes (les données) affichent des lettres ou des chiffres. Human resource machine est un jeu de puzzle ; à chaque niveau, le boss donne un travail au joueur. Le joueur doit automatiser le travail en programmant l'employé de bureau pour qu'il exécute le travail.

Scratch Jr n'utilise pas de variables, et comme dans le jeu "Human resource machine", les danseurs portent des valeurs qui, au lieu d'être dans des boîtes, sont écrites sur une ardoise magique ou représentées sur leurs doigts. Cependant, avant de nous restreindre à Scratch Jr, nous avons travaillé sur la différence entre constantes et variables que nous évoquons ici (cela pourrait aussi avoir sa place dans la section 3.1 Matière). Une constante est une valeur subordonnée et les novices peuvent être tenu.es à l'écart des mécanismes requis pour gérer les constantes. Mais ils et elles ne peuvent ignorer qu'une variable est un emplacement dans la mémoire qui est associé à un nom (son identificateur) et qui contient des valeurs connues ou inconnues. Par conséquent, une variable devrait être une partie nommée de la scène d'exécution, et déclarer une variable (intentionnellement ou automatiquement) devrait réserver un emplacement sur la scène et lui attribuer un nom. Cependant le concept de variable est difficile à comprendre pour les novices. Corritore et Wiedenbeck [6] ont montré que les novices ont plus de difficulté avec les flots de données qu'avec les flots de contrôle. Quand nous allons traiter une version de DYA gérant des variables, nous envisageons d'emprunter la notion d'éléments exemplaires qu'on trouve dans le langage de requêtes Query-By-Example, QBE [30]. Un élément exemplaire (une valeur utilisée dans une requête) sert de variable alors qu'un élément constant est une valeur constante. Nous imaginons que les éléments exemplaires - jouant le rôle de variable - seront représentés par une sorte particulière de données (au moyen d'un signe distinctif) dont les valeurs peuvent changer pendant l'exécution.

3.4 Temps

Le temps est quand quelque chose survient. Le temps pourrait être compris comme le composant rythmique de la danse, comme il l'est pour la musique mais ce n'est pas une analogie appropriée en programmation. Le temps est une notion fondamentale dans une exécution car, à chaque moment du temps, il est possible de se trouver dans un état particulier de l'espace d'états (l'ensemble des états possibles), différent des autres états. Nous devons reconnaitre les différents moments temporels d'une danse parce qu'ils représentent le temps de la machine en exécution c.à.d. le moment où l'on change de l'exécution d'un bloc à l'exécution du bloc suivant. Quand la machine notionelle est un système séquentiel où il n'y a qu'un seul fil d'exécution (*thread of control*), il n'y a qu'un seul temps. Il nous semble alors que les questions sur le temps sont d'ordre technique et sont essentiellement liées à la capture du mouvement et à la reconnaissance des gestes. Nous avons besoin que la danse adhère à une base de temps canonique qui simplifie, en temps réel, la reconnaissance et la classification des gestes de danse [25]. Une machine notionelle

Figure 3: Gestes d'un programme DYA équivalent au programme Scratch Jr de la figure 2

concurrente devra s'intéresser à des questions de temps qu'on trouve traitées, par exemple, dans le domaine des langages de programmation réactifs synchrones.

4 FACETTES DE LA PERSPECTIVE DE L'OBSERVATEUR

La perspective de l'observateur fournit la vue du corps perçu de l'extérieur par une autre personne. Selon la méthode, il y a deux activités : (i) description et documentation du mouvement ; (ii) analyse visuelle et représentation des corps en mouvement. Ces deux activités se pratiquent plutôt de manière descendante et nous allons traiter quelques questions dans l'ordre inverse de l'ordre PMEST.

4.1 Temps

Dans la version actuelle de DYA, nous faisons le postulat que la machine notionelle est séquentielle. Cependant nous percevons notre monde comme un ensemble de sujets distincts distribués dans le temps et dans l'espace. Nous pourrions imaginer que la machine notionelle est comme notre monde et que les danses en font de même. alors nous aurions un modèle idéalisé de calcul dans lequel des fils d'exécutions concurrents manipulent un ensemble d'objets partagés. Supposons que la danse soit exécutée par plusieurs sujets actifs en même temps, mais que l'ensemble de la scène soit capturé par une seule caméra. Dans ce cas, nous avons une sorte d'effet stroboscopique, à chaque moment de capture, chaque sujet (et ses différentes parties) peuvent être reconnus en train de faire un geste différent du moment précédent. Cependant le temps de tous les sujets est synchronisé : tous les sujets partagent les mêmes moments de reconnaissance. Si nous pouvons affecter une caméra à chaque sujet (ou à un petit groupe de sujets), les différents sujets peuvent évoluer sur des bases de temps différentes : ils sont asynchrones.

4.2 Espace

Dans DYA , nous postulons que la scène entière (l'espace) est partagée par tous les danseurs. Nous avons un modèle de calcul où la danse représente la mémoire (partagée) de l'ordinateur. Mais nous pourrions diviser la scène en différentes parties, maintenir chaque sujet ou groupe de sujets dans des parties différentes de la scène, et fournir des primitives de communication avec les autres sujets ou groupes de sujets : signaux, variables globales ou partagées. Même avec une seule caméra, on peut avoir un système avec une mémoire distribuée.

4.3 Énergie

La catégorie Énergie est sans doute celle où il y a le plus de différence entre un langage d'exécution et un langage de composition (de programmation). Nous avons beaucoup réduit l'écart et les difficultés afférentes dans la version actuelle de DYA qui est basée sur Scratch Jr, un langage où la divergence entre exécution et composition est petite. En effet, sauf pour les boucles, l'exécution d'un bloc de Scratch Jr peut être relié de manière univoque avec la définition de ce même bloc. C'est d'ailleurs le parti pris de cet article, plutôt que d'analyser le problème sous les deux angles de l'exécution et de la programmation, nous l'avons analysé sous l'angle du danseur et de l'observateur, ce qui nous a permis d'échapper à des incompatibilités sémantiques. Cependant si on doit enrichir le langage avec toutes les constructions classiques de la programmation, ces divergences devront être résolues. Une alternative n'existe pas en exécution : un chemin d'une alternative est un chemin comme un autre et si un chemin doit être reconnu comme appartenant à une alternative, cela veut dire qu'on a su reconnaitre deux exécutions comme faisant partie du même programme mais différant seulement sur une partie de chemin qui sera alors associé à une alternative. Les boucles n'existent pas, per se, dans une exécution mais les cycles existent et peuvent être reconnus et associés à une boucle. Il sera peut-être fructueux d'explorer d'autres paradigmes de programmation pouir faciliter les allers-retours entre exécution et conception de programmes. Les langages Événement-Condition-Action appartiennent à un paradigme intuitif et puissant pour programmer des systèmes réactifs. Les systèmes ECA reçoivent des entrées (principalement sous la forme d'événements) en provenance de l'environnement externe et réagissent en exécutant des actions qui modifient les informations stockées dans le système (actions internes) ou inflience l'environnement du sytème (actions externes) [1]. Les diagrammes états-transitions (*statechart*) sont aussi une alternative de programmation pour novices, ils décrivent les états d'un système et les transitions possibles dans une forme modulaire qui permet le groupement, la concurrence et le raffinement, et encourage des capacités de 'zoom' pour déplacer le point de vue entre différents niveaux d'abstraction [13].

4.4 Personnalité et matière

Les parties d'un programme sont principalement les structures de contrôles et les procédures. Les parties des données sont gérées par les types et les structures de données. Dans DYA, la gestuelle des boucles est basée sur une signalisation début / fin : un geste signale qu'on entre dans la boucle, un autre geste indique qu'on sort de la boucle. Pour les boucles imbriquées, la structure d'imbrication n'est pas triviale à reconnaitre du point de vue d'un observateur. Les types complexes comme les listes pourraient être représentés avec une organisation dédiée de danseurs-données et de danseurs-instructions représentant les opérations primitives sur ces types.

5 PERSPECTIVES ET CONCLUSION

DYA est un effort et une tentative pour faciliter l'apprentissage de la programmation pour de jeunes novices ; cependant pour certain.es élèves, l'énaction des constructions de programmation par le mouvement facilite la compréhension des concepts et l'apprentissage ultérieur de Scratch Jr. Par exemple, dans quelques classes, la plupart des enfants n'ont pas bien compris la séance sur les échanges de messages ; et une simulation où les enfants se sont écrits et ont échangé de "vrais" messages a permis à l'ensemble de la classe de comprendre le paradigme d'échange de messages. Un autre exemple concerne les déplacements dans un labyrinthe où on utilise au préalable des déplacements physiques dans un labyrinthe tracé sur le sol de la classe. En conséquence, nous avons entrepris le projet DYA et nous avons présenté dans cet article les choix préliminaires de conception du langage et leur impact sur la machine notionelle sous-jacente ainsi que des alternatives qui pourraient conduire à des machines complètement différentes.

Bien que la réalisation du système automatisé soit en cours, ce qui reste à faire pour la première version du langage ressort de l'ingénierie de développement de système, et à notre avis, cet article fait le tour des questions de recherche et des solutions proposées pour la première version du langage. Les essais d'ergonomie du langage sont déjà en cours, en utilisant des étudiant.es pour traduire les programmes dansés par les enfants en Scratch Jr, ce qui permet d'avoir déjà un feedback sur la pertinence des gestes et des choix de conception. Comme indiqué en introduction de cet article, la machine notionelle doit être simple, et on doit fournir au novice des méthodes d'observation de la machine en action. L'angle de résolution de notre recherche, qui en fait son originalité, est de ne pas essayer de résoudre le problème d'impédance entre exécution de programmes et conception de programmes, mais plutôt d'examiner la dualité qu'il y a entre danser ses programmes et observer la danse de programmes, notamment sa propre danse enregistrée, analysée et reconnue par le système. Ce sont les trois perspectives de la méthode Moving and Making Strange, méthode qui nous a fourni un ensemble d'activités dans chaque perspective.

La mise au point du langage DYA opère de manière incrémentale, par essais et retours d'expérience successifs. Deux points durs ont été laissé de côté : les structures conditionnelles et les variables. Ce sont sensiblement les mêmes choix faits en Scratch Jr, ces points absents sont fournis en Scratch (pour des enfants plus grands). Scratch est d'ailleurs vu par ses auteurs comme un marchepied vers la "vraie" programmation, car les concepts computationnels (qui correspondent aux blocs de Scratch) sont communs à de nombreux langages de programmation [5]. L'idée est de fournir des machines notionelles qui s'imbriquent telles des poupées russes et c'est cet objectif incrémental que nous poursuivons dans DYA.

Plusieurs approches pour la reconnaissance de gestes sont basées sur la quantification de concepts par une analyse Laban du mouvement [3, 18, 23]. L'utilisation fréquente dans la communication humaine d'exemples illustratifs est une application de la théorie d'inférence inductive ; c'est souvent utilisé pour expliquer tout ou partie de la spécification d'un algorithme ou d'une procédure [2, 9, 27]. On a à apprendre la grammaire d'un programme inconnu à partir d'exemples. L'idée générale est que l'information est présentée à l'apprenant.e qui met à jouir ses hypothèses après chaque morceau d'information. Au bout d'un certain moement, l'apprenant.e aura trouvé le concept correct et n'en bougera plus [8]. La capacité d'exécuter des modèles conceptuels a été considérée comme vitale pour la validation de leurs propriétés dynamiques. Gulla [12] met en avant une approche qui utilise des explications pour améliorer la validation de modèles exécutables, cette approche utilise un composant qui peut expliquer le raisonnement interne du modèle de même que les interactions avec l'utilisateur. Des idées des langages de programmation basés sur les prototypes sont présentes dans GME (Generic Modelling Environment) ce qui en fait un outil appréciable pour la construction de programmes à partir d'instances d'exécution [15].

REMERCIEMENTS

La première version du langage DYA a été financée par la Fondation Blaise Pascal (bénéficiaire du Programme d'Investissements d'Avenir - PIA) et par un aide de notre laboratoire au titre des projets transversaux. Nous remercions Lisa pour le croquis du système DYA utilisé dans le teaser.

RÉFÉRENCES

[1] José Júlio Alferes, Federico Banti, and Antonio Brogi. 2006. An event-condition-action logic programming language. In *European Workshop on Logics in Artificial Intelligence*. Springer, 29–42.

[2] Dana Angluin. 1987. Learning regular sets from queries and counterexamples. *Information and computation* 75, 2 (1987), 87–106.

[3] Andreas Aristidou, Efstathios Stavrakis, Panayiotis Charalambous, Yiorgos Chrysanthou, and Stephania Loizidou Himona. 2015. Folk dance evaluation using laban movement analysis. *Journal on Computing and Cultural Heritage (JOCCH)* 8, 4 (2015), 20.

[4] Benedict du Boulay, Tim O'Shea, and John Monk. 1999. The black box inside the glass box : presenting computing concepts to novices. *International Journal of Human-Computer Studies* 51, 2 (1999), 265 – 277.

[5] Karen Brennan and Mitchel Resnick. 2012. New frameworks for studying and assessing the development of computational thinking.

In *Proceedings of the 2012 annual meeting of the American Educational Research Association, Vancouver, Canada*. American Educational Research Association, Vancouver, Canada, 1–25.

[6] Cynthia L Corritore and Susan Wiedenbeck. 1991. What do novices learn during program comprehension ? *International Journal of Human-Computer Interaction* 3, 2 (1991), 199–222.

[7] Simon P Davies. 1993. Models and theories of programming strategy. *International Journal of Man-Machine Studies* 39, 2 (1993), 237–267.

[8] Colin De la Higuera. 2010. *Grammatical inference : learning automata and grammars*. Cambridge University Press.

[9] E Mark Gold and others. 1967. Language identification in the limit. *Information and control* 10, 5 (1967), 447–474.

[10] TRG Green. 1991. Programming languages as information structures. In *Psychology of programming*. Elsevier, New York, U.S.A., 117–137.

[11] Object Management Group. 2015. Unified Modeling Language 2.5. OMG. (May 2015). https://www.omg.org/spec/UML/2.5/

[12] Jon Atle Gulla and Geir Willumsen. 1993. Using explanations to improve the validation of executable models. In *International Conference on Advanced Information Systems Engineering*. Springer, 118–142.

[13] David Harel. 1987. Statecharts : A visual formalism for complex systems. *Science of computer programming* 8, 3 (1987), 231–274.

[14] Birger Hjørland. 2013. Facet analysis : The logical approach to knowledge organization. *Information processing & management* 49, 2 (2013), 545–557.

[15] Gábor Karsai, Miklos Maroti, Ákos Lédeczi, Jeff Gray, and Janos Sztipanovits. 2004. Composition and cloning in modeling and meta-modeling. *IEEE Transactions on Control Systems Technology* 12, 2 (2004), 263–278.

[16] Zoltan Katai and Laszlo Toth. 2010. Technologically and artistically enhanced multi-sensory computer-programming education. *Teaching and teacher education* 26, 2 (2010), 244–251.

[17] Kathryn La Barre. 2010. Facet analysis. *Annual Review of Information Science and Technology* 44, 1 (2010), 243–284.

[18] Rudolf Laban. 1994. La maîtrise du mouvement. *Arles, Actes Sud* (1994), 48.

[19] Lian Loke and Toni Robertson. 2013. Moving and making strange : An embodied approach to movement-based interaction design. *ACM Transactions on Computer-Human Interaction (TOCHI)* 20, 1 (2013), 7.

[20] Richard E Mayer, Jennifer L Dyck, and William Vilberg. 1986. Learning to program and learning to think : what's the connection ? *Commun. ACM* 29, 7 (1986), 605–610.

[21] Tanja Mayerhofer, Philip Langer, and Gerti Kappel. 2012. A runtime model for fUML. In *Proceedings of the 7th Workshop on Models@ run. time*. ACM, Innsbruck, Austria, 53–58.

[22] Lars Nyberg. 1993. *The enactment effect : Studies of a memory phenomenon*. Ph.D. Dissertation. Umeå Universitet.

[23] Bernstein Ran, Shafir Tal, Tsachor Rachelle, Studd Karen, and Schuster Assaf. 2015. Multitask learning for Laban movement analysis. In *Proceedings of the 2nd International Workshop on Movement and Computing*. ACM, 37–44.

[24] Shiyali Ramamrita Ranganathan. 1963. *Colon classification : basic classification*. Vol. 26. Asia Pub. House, New York, U.S.A.

[25] Michalis Raptis, Darko Kirovski, and Hugues Hoppe. 2011. Real-time classification of dance gestures from skeleton animation. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics symposium on computer animation*. ACM, 147–156.

[26] Anthony Robins, Janet Rountree, and Nathan Rountree. 2003. Learning and teaching programming : A review and discussion. *Computer science education* 13, 2 (2003), 137–172.

[27] Ray J Solomonoff. 1964. A formal theory of inductive inference. Part I. *Information and control* 7, 1 (1964), 1–22.

[28] Willemien Visser. 1990. More or less following a plan during design : opportunistic deviations in specification. *International journal of man-machine studies* 33, 3 (1990), 247–278.

[29] Leon E Winslow. 1996. Programming pedagogy—a psychological overview. *ACM Sigcse Bulletin* 28, 3 (1996), 17–22.

[30] Moshe M. Zloof. 1977. Query-by-example : A data base language. *IBM systems Journal* 16, 4 (1977), 324–343.