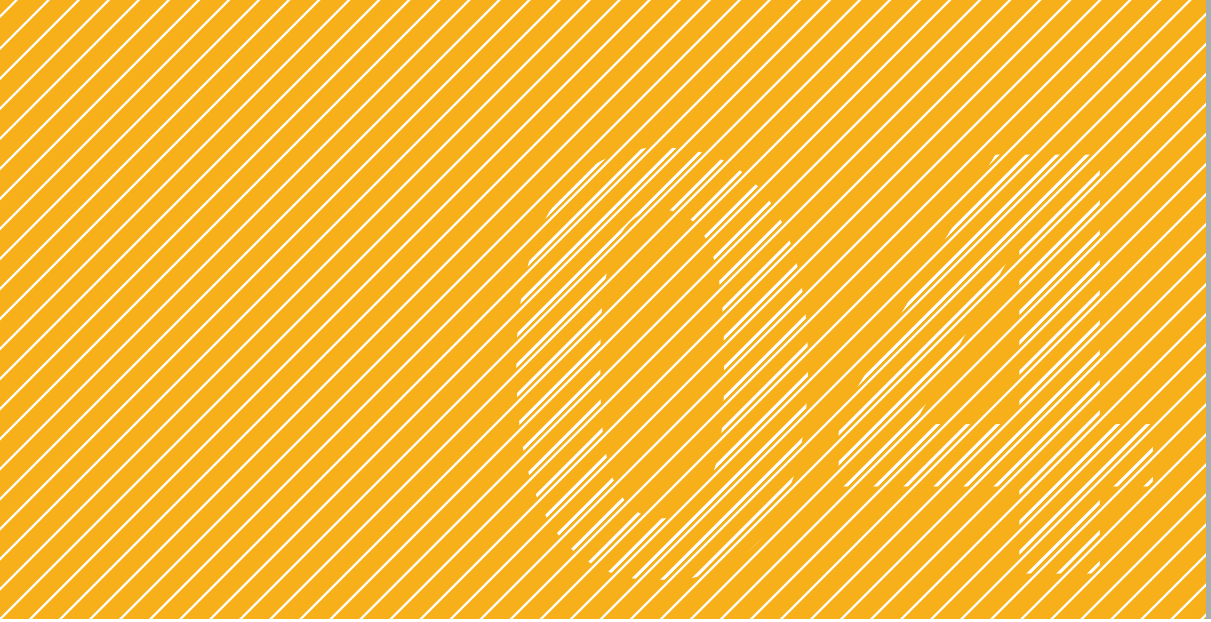


CAHIERS • DÉCEMBRE 2014

HASLERSTIFTUNG

LA PENSÉE INFORMATIQUE DANS LA FORMATION DES ENSEIGNANTS

ALEXANDER REPENNING

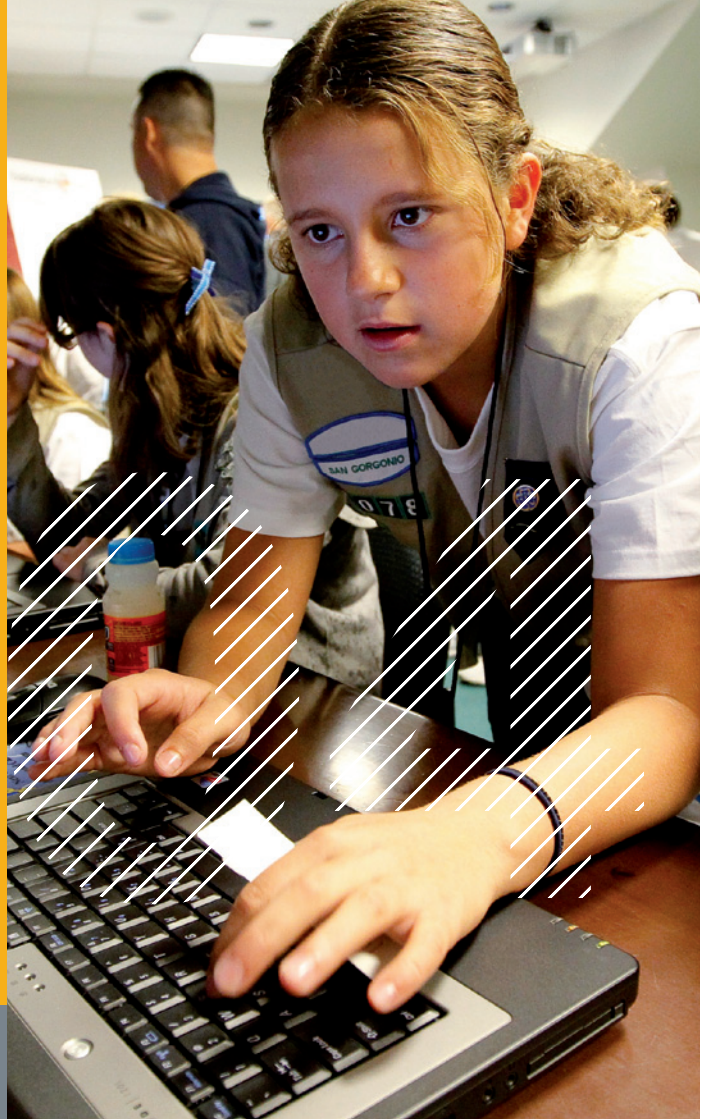


TABLE DES MATIÈRES

La pensée informatique dans la formation des enseignants 3

Pourquoi la pensée informatique ? 4

Comment fonctionne la pensée informatique ? 6

Pensée informatique ≠ programmation 9

Une stratégie de pensée informatique : Scalable game design 12

Un modèle scolaire 20

LA PENSÉE INFORMATIQUE DANS LA FORMATION DES ENSEIGNANTS

Le but de cette brochure est l'introduction axée sur la pratique de la notion de pensée informatique dans la formation des enseignants. Bien que les enfants soient nombreux à avoir une haute affinité pour la technologie informatique, à consommer des jeux vidéo ou communiquer avec leurs amis sur les réseaux sociaux, ils s'intéressent en général incroyablement peu à la programmation. Beaucoup d'élèves utilisent des mots cinglants pour parler de l'informatique : « La programmation, c'est pénible et ennuyeux. »

L'auteur de ce document a développé la méthode du *Scalable game design* il y a bon nombre d'années à l'Université Boulder, Colorado, pour initier les enfants et les adolescents à l'informatique. Le *Scalable game design* est une combinaison d'approches, d'outils et d'activités pédagogiques, avec lesquels les enseignant-e-s peuvent élaborer une formation à l'informatique avec leurs élèves de façon stimulante, afin de les familiariser délibérément avec la pensée informatique. La programmation doit devenir simple et passionnante !

POURQUOI LA PENSÉE INFORMATIQUE ?

Qu'est-ce au juste que la pensée informatique (*Computational Thinking*) et qui en a besoin ? En matière de formation informatique, les avis sont partagés, car beaucoup ne savent pas bien ce que cette notion représente exactement. Les uns pensent que formation informatique signifie dactylographie ou utilisation de logiciels de traitement de textes ou de conception. D'autres assignent la formation informatique à l'informatique pure et dure, soit une formation très axée sur la programmation, qui donnera naissance à la prochaine génération de « super programmeurs » et de « méga fondateurs d'entreprises » tels que Bill Gates (Microsoft) ou Mark Zuckerberg (Facebook).

Désormais, le Plan d'études 21 prévoit une introduction de la formation informatique dès le primaire et les esprits s'échauffent : que doit-on enseigner et comment peut-on dans la pratique introduire de nouveaux contenus dans des emplois du temps déjà pleins ?

Quoi, quand et si

Les statistiques de la situation de l'emploi indiquent que la Suisse a effectivement besoin d'informaticiennes et d'informaticiens. Mais cela signifie-t-il obligatoirement que les écoles doivent enseigner l'informatique fondamentale dans le sens évoqué ci-dessus ? La recherche montre que même des pays comme les États-Unis, qui ont davantage d'expérience en matière de formation informatique, peinent à intégrer la programmation traditionnelle dans l'enseignement. Comme souvent, la solution repose sur un compromis.

La pensée informatique est une approche qui se concentre sciemment sur des concepts et des stratégies de résolution de problèmes de nature générale. Il s'agit entre autres de comprendre la relation entre processus séquentiels et processus parallèles. Cela parait compliqué, mais en fait ne l'est pas. La grand-mère qui veut faire un gâteau en est un exemple concret. Elle sait qu'elle peut déjà préparer le glaçage, pendant que le gâteau cuit dans le four.

Quoi, quand et si, c'est ça la pensée informatique. La teneur de la pensée informatique n'est pas seulement pertinente pour l'informatique, mais aussi pour d'autres sciences telles que les mathématiques ou les sciences naturelles, ou même les langues, l'art et le design. Par exemple en sciences naturelles, une personne qui réalise seule la simulation d'un écosystème par le biais de la programmation apprend la science comme un scientifique et non simplement par cœur. La Suisse a besoin de penseurs - de « penseurs informatiques » - et non de « cartes mémoire ».

Expérience venue des États-Unis

Tout cela à l'air très académique. Cela peut-il vraiment fonctionner ? Les États-Unis ont commencé il y de nombreuses années à analyser systématiquement la façon dont on pourrait encourager la pensée informatique dans les écoles. Grâce aux éléments du *Scalable game design*, les élèves commencent à construire des jeux vidéo de façon autonome dès la troisième classe, afin d'éveiller leur pensée informatique. Ils ne jouent pas, ils conçoivent, et apprennent ainsi la pensée informatique - par le biais de la programmation. Le plus stupéfiant n'est pas leur incroyable motivation, mais le degré de difficulté auquel les élèves osent s'attaquer volontairement. Ainsi des enfants de 10 ans veulent comprendre les équations de diffusion dont ils ont besoin pour intégrer l'intelligence artificielle dans leur jeu. La pilule ne doit pas être amère. Avec une bonne motivation, apprendre peut être amusant.

Les grandes études de ce type aux États-Unis ont montré non seulement que garçons et filles s'intéressent de la même façon à ce concept et sont capables de l'appliquer, mais aussi que même les enseignant-e-s sans connaissances préalables dans le domaine de l'informatique peuvent transmettre la pensée informatique. L'objectif est maintenant de mettre en œuvre un programme de promotion de la pensée informatique en Suisse.

COMMENT FONCTIONNE LA PENSÉE INFORMATIQUE ?

Jeanette Wing définit la pensée informatique de la façon suivante :

> **La pensée informatique est un processus de réflexion qui visualise tant la formulation d'un problème que la représentation de sa solution de façon à ce qu'elles puissent être exécutées par des humains ou des machines.**

Cette définition implique dans la plupart des cas un processus en trois étapes :

1. **Formulation du problème :** Dans sa forme la plus simple, une question précise basée sur l'analyse d'un problème.
2. **Représentation d'une solution :** une représentation reposant sur une combinaison de texte et de diagrammes, par exemple les instructions de montage d'un meuble, une recette ou un programme informatique.
3. **Exécution et évaluation de la représentation de la solution :** par exemple la réalisation d'une recette de gâteau et la vérification ultérieure de la qualité du gâteau.

Dans l'exemple de la grand-mère, la formulation du problème consiste à identifier les questions spécifiques qui se posent lors de la fabrication d'un gâteau. Cela comprend par exemple de s'apercevoir que le glaçage ne peut être versé qu'à la fin sur le gâteau cuit. La représentation d'une solution peut être la recette du gâteau, à savoir la description détaillée des étapes de travail parallèles ou successives dans le temps. La représentation de la solution peut être réalisée par la grand-mère elle-même ou son petit-enfant, pour qui la grand-mère a écrit la recette. La grand-mère et le petit-enfant peuvent vérifier ensuite la qualité du résultat en goûtant le gâteau.

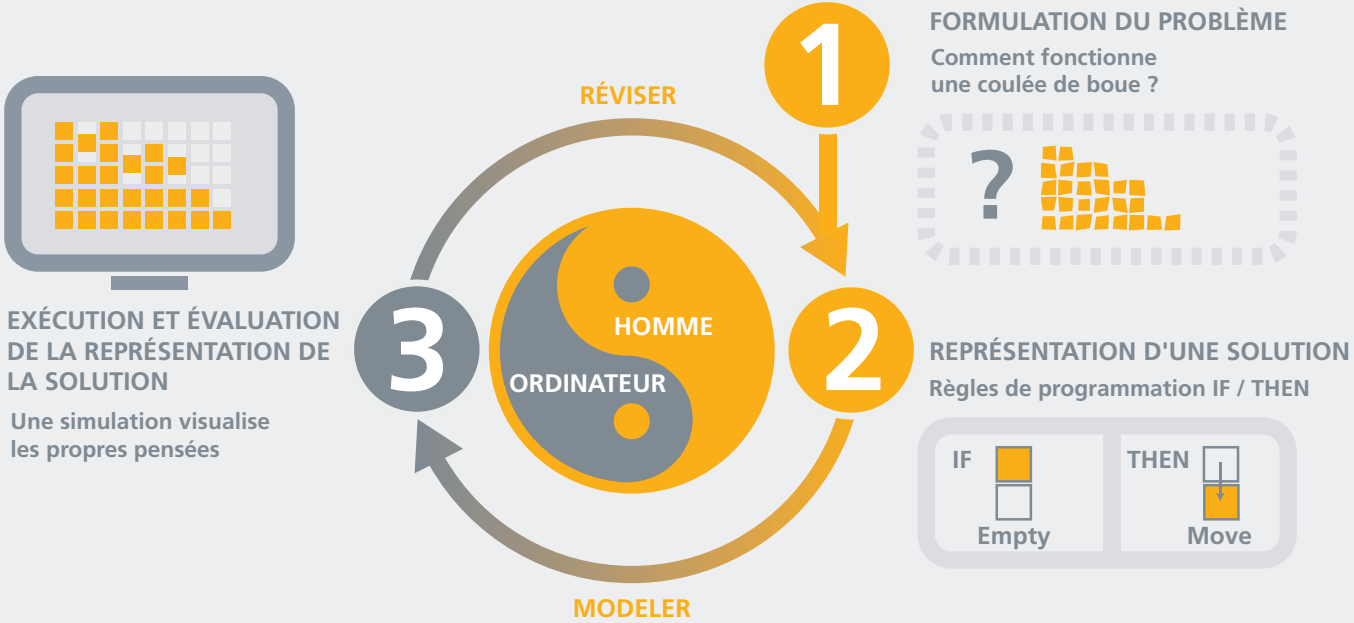
Pourquoi a-t-on besoin de la pensée informatique ? Les objectifs sont nombreux, mais ce document se concentre sur les deux aspects suivants :

- **Délégation et automatisation :** la représentation d'une solution peut être utilisée pour déléguer un processus de travail, soit à une autre personne, soit à un ordinateur. La recette de gâteau, si elle est suffisamment détaillée, précise et compréhensible, peut être réalisée par n'importe qui. Un programme informatique est lui-aussi la représentation d'une solution. L'enregistrement d'une

émission de télévision « programmé » par une utilisatrice constitue un exemple très simple d'automatisation.

- **Acquisition de connaissances :** ce processus unique et rectiligne en trois étapes ne fonctionne pas vraiment quand la solution du problème n'est pas évidente. C'est typiquement le cas quand il s'agit d'acquérir des connaissances. Dans le contexte de l'éducation, cette utilisation de la pensée informatique est cependant très intéressante. Le processus de pensée informatique général est itératif (voir figure page 8). Il est décrit ci-dessous à l'aide d'un exemple.

La pensée informatique offre une synthèse idéale des capacités de l'homme et de l'ordinateur dans l'acquisition active de connaissances. Le processus commence ① avec une question. Par exemple, « Comment fonctionne une coulée de boue ? » À ce moment X, nous ne connaissons pas encore la solution et nous ne pouvons par conséquent pas la représenter sous une forme définitive, c'est-à-dire d'un programme fini. Mais nous pouvons élaborer une première approche de solution. On peut par exemple se représenter une coulée de boue comme un amas de blocs de boue. Ces blocs peuvent être posés les uns sur les autres et les uns à côté des autres. Avec un outil de pensée informatique tel que *AgentSheets*, ces blocs peuvent être visualisés et représentés sous forme d'objets - appelés aussi agents - organisés dans un tableau. Dans l'étape ②, l'utilisateur ou l'utilisatrice formule par calcul les relations entre les agents. Dans le cas des *AgentSheets* ou d'outils similaires, ces relations sont exprimées sous forme de règles de programmation. La règle représente une approximation simple de la gravitation. Si (IF) rien ne se trouve sous un bloc de boue (*empty below*), alors (THEN) le bloc descend (*move down*). L'être humain a ainsi appréhendé une idée par le calcul. C'est maintenant la tâche de l'ordinateur ③ de visualiser les conséquences de cette idée. La tâche de l'ordinateur n'est pas de trouver l'idée, l'être humain le fait bien mieux. Mais grâce à ses capacités de calcul, l'ordinateur peut visualiser de tels concepts d'une manière qui est difficile, voire impossible pour l'être humain.



Comment la pensée informatique fait-elle le lien entre le mode de pensée humain et l'ordinateur ? Il ne s'agit pas de penser comme un ordinateur, la pensée informatique veut dire penser avec l'ordinateur. Cela signifie que la pensée informatique est une façon de penser qui utilise l'ordinateur comme instrument de soutien du processus de la pensée. On peut comparer l'ordinateur en tant qu'instrument à un microscope - il aide à rendre visibles des relations difficiles à comprendre et soutient ainsi le développement de la pensée informatique.

La pensée informatique - Synthèse des capacités de l'homme et de l'ordinateur

PENSÉE INFORMATIQUE ≠ PROGRAMMATION

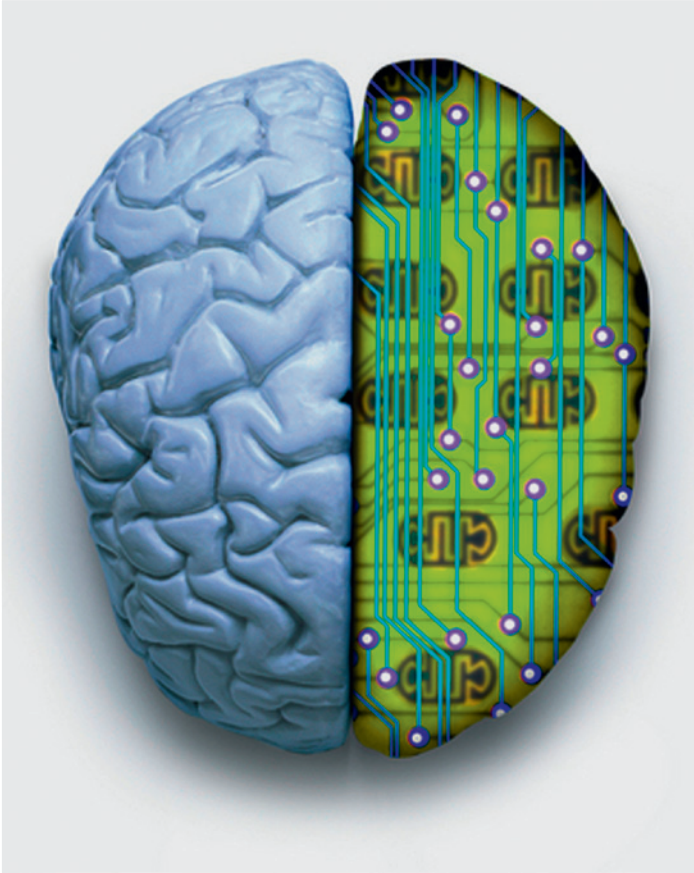
La pensée informatique n'est pas la même chose que la programmation. Elle repose bien plus sur la compréhension des concepts de programmation, qui sont applicables à différents langages. Par contre, chaque langage de programmation se distingue par sa syntaxe, que les utilisatrices et utilisateurs doivent maîtriser pour pouvoir écrire un programme fonctionnel. Les détails syntaxiques d'un langage de programmation ne sont cependant pas très importants pour la compréhension des concepts de programmation. Cela ne signifie pas que la connaissance de divers langages de programmation soit sans importance pour la pensée informatique, mais que la pensée informatique privilégie les approches pédagogiques, qui s'inspirent de l'acquisition de la langue maternelle. Lors de l'apprentissage de sa langue maternelle, le petit enfant se concentre sur une communication pertinente, axée sur ses buts, et non pas sur des règles grammaticales. De façon tout à fait analogue, la pensée informatique se concentre sur l'objectif, comme par exemple construire une simulation afin d'acquérir des connaissances. Qui simule un incendie de forêt s'intéresse par exemple à la relation entre la densité des arbres et la probabilité que la forêt brûle dans sa totalité, et non de savoir si une « boucle » (*instruction loop*) a été utilisée pour l'implémentation de cette simulation. Des compétences de programmation sont certes nécessaires et aussi souhaitées, mais, de même que le fait de suivre correctement des instructions de montage IKEA ne fait pas automatiquement de l'utilisateur un menuisier, les compétences de programmation ne sont pas comparables à la compréhension des schémas de pensée informatique.

La pensée informatique signifie se concentrer sur l'essentiel de manière élégante afin d'intéresser le public le plus large possible aux concepts informatiques. Mais c'est justement ce public qui perçoit habituellement la programmation comme « pénible et ennuyeuse ». Il n'est pas rare en effet que des programmes exprimant une idée relativement simple soient incroyablement longs et compliqués, au point que l'on ne reconnait pratiquement pas

l'idée dans le code. On pourrait ici faire plus avec moins, et en s'y prenant bien, même rendre la programmation « simple et intéressante ». Hitchcock a un jour défini le concept de drame dans les films comme une vie dont on a éliminé les moments ennuyeux. C'est tout à fait dans cet esprit que Repenning a construit ces 20 dernières années des outils que l'on peut appeler outils de la pensée informatique puisqu'ils mettent l'accent sur l'essentiel. Avec les *AgentCubes*, on peut fabriquer par exemple un jeu *Pac-Man* faisant intervenir l'intelligence artificielle avec juste 10 règles.

Un outil de pensée informatique est plus qu'un simple environnement de programmation. Il applique les principes de la pensée informatique au processus même de programmation. Un outil de pensée informatique doit visualiser de la manière le plus efficace possible les conséquences de sa propre pensée. Pour les débutants surtout, le décalage entre le moment où une erreur de programmation est faite et la manifestation de cette erreur peut être extrêmement frustrant. Plus ce décalage est long, plus il est probable que le programmeur ait pendant ce temps pris de nouvelles décisions et les ait intégrées dans le code. Plus le décalage augmente, plus il devient difficile de trouver l'origine de l'erreur. C'était on ne peut plus vrai dans la programmation à l'aide de cartes perforées. Souvent il fallait attendre des heures avant de voir le résultat du programme. Malheureusement, beaucoup d'environnements de programmation reposent aujourd'hui encore sur ces approches. Certes il n'est plus nécessaire de se battre avec une pile de cartes perforées, mais un programme doit être complet avant qu'on puisse le compiler (traduire) et puis enfin le lancer. La programmation en direct (*live*) va dans la bonne direction du fait qu'elle permet à ses utilisateurs de modifier un programme de manière incrémentielle, sans devoir relancer le programme depuis le début. Ainsi l'utilisateur peut voir et comprendre rapidement l'effet réel d'une modification. Le concept de la programmation

conversationnelle dans *AgentCubes* va encore plus loin, le programme faisant un pas dans le futur pour montrer à l'utilisateur ce qu'il va faire. Finalement, avec les outils de pensée informatique, il s'agit de trouver une symbiose judicieuse entre les capacités humaines et celles des ordinateurs. Avec les ordinateurs actuels justement, qui disposent d'une énorme puissance de calcul, le processus humain de programmation devrait enfin être soutenu activement.



Symbiose de l'homme et de l'ordinateur

UNE STRATÉGIE DE PENSÉE INFORMATIQUE : SCALABLE GAME DESIGN

Le projet de *Scalable game design*[1], sous la direction d'Alexander Repenning, étudie comment intégrer systématiquement la formation informatique dans les écoles en s'appuyant sur la pensée informatique. Le but est de motiver les élèves à l'informatique et de les former dans cette matière en utilisant la conception de jeux. L'approche a été expérimentée dans le cadre de grandes études sur la formation informatique aux États-Unis, avec plus de 10 000 participant-e-s. Le *Scalable game design* repose sur une stratégie en quatre points :

1. **Pénétration :** des modules pédagogiques faciles à adapter doivent être intégrés précocement dans des classes existantes, afin que de grands groupes divers d'enfants entrent en contact avec la pensée informatique. La formation informatique, programmation comprise, ne doit pas être limitée à l'encouragement des élèves doués, aux cours d'été ou aux options complémentaires. La pensée informatique peut commencer dès le jardin d'enfants - avec ou sans ordinateur. Le but est d'offrir à tous les élèves la possibilité de développer la pensée informatique. La pensée informatique peut être intégrée aux domaines les plus divers - dans les disciplines MINT (mathématiques, informatique, sciences naturelles et technique), mais aussi les langues, l'art et le design, la musique ou même le sport. Résultats : ce type de formation informatique est universellement applicable avec les enseignant-e-s et élèves d'un large éventail d'écoles. Aux États-Unis, des projets scolaires ont été réalisés aussi bien dans les zones rurales que dans les écoles de grandes villes - des projets pilotes ont même eu lieu dans des réserves indiennes. En moyenne, 45% de filles ont pris part à l'étude.

2. **Motivation :** l'idée de base est que la motivation est la précondition à l'acquisition de compétences. À l'inverse, l'appropriation de compétences n'entraine pas obligatoirement la motivation. Le *Scalable game*

1 www.scalablegamedesign.ch scalablegamedesign.cs.colorado.edu

design comprend une série de projets de jeux et de simulations soigneusement échelonnés. Au bout de peu de temps, les élèves construisent eux-mêmes des jeux d'ordinateur présentant différents niveaux de difficulté, qui évoluent par étapes de contenus de programmation simples à des contenus complexes. Des connaissances en programmation ne sont demandées à aucun moment. Résultats : bien que la plupart des élèves apprennent le Scalable game design dans des cours obligatoires, 74% des garçons et 64% des filles aux États-Unis souhaitent continuer.

3. **Compétences :** à l'aide d'outils assistés par ordinateur, les projets et programmes des élèves peuvent être analysés afin de mesurer objectivement les compétences. Les résultats comprennent des courbes d'apprentissage et le transfert de connaissances de différents schémas de pensée informatique - des jeux vidéo aux simulations scientifiques. Résultats : un outil d'évaluation appelé *Computational Thinking Pattern Analysis* (analyse des schémas de pensée informatique) a montré que les élèves peuvent reconnaitre comme concepts et mettre en œuvre des notions telles que collision et diffusion, non seulement pour créer des jeux mais aussi pour représenter des simulations.

4. **Pédagogie :** la formation des enseignants comprend entre autres de nouveaux concepts pédagogiques qui encouragent l'apprentissage progressif des contenus de programmation et permettent des acquis rapides même pour les personnes sans expérience en programmation. L'objectif est d'encourager une participation aussi large que possible des élèves et de concilier transmission de connaissances de programmation et travail individuel. Résultats : il s'est avéré que l'utilisation d'une pédagogie appropriée est le principal facteur permettant d'éveiller autant l'intérêt des filles que des garçons.

Scalable Game Design comme motivation

Fondements théoriques de l’apprentissage

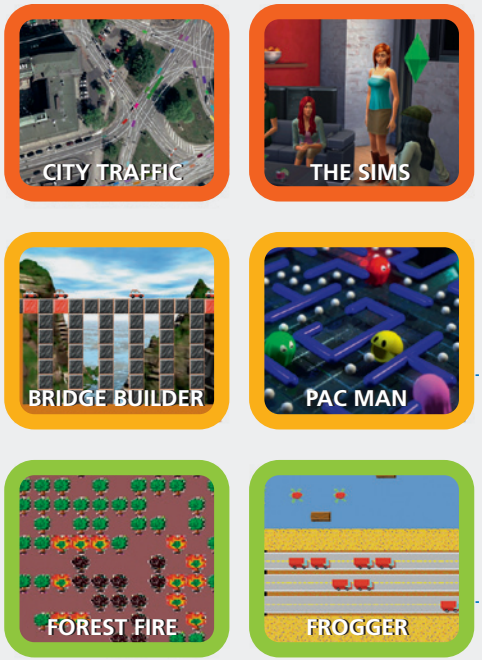
La stratégie du *Scalable game design* est résumée dans la figure (page 16) par le concept des *Zones of proximal flow* (zones de flux proximal). Selon ce concept, les individus sont particulièrement motivés lorsqu'ils se trouvent dans la zone appelée *Flow* [Csíkszentmihályi], où le défi et les compétences individuelles sont parfaitement équilibrés. Le tennis en est un bon exemple. Si un débutant jouait contre un professionnel tel que Roger Federer, la partie serait très déséquilibrée. Le débutant ne possède pratiquement aucune compétence en tennis. Jouer contre un professionnel représente un énorme défi. Le débutant n'est donc pas dans le *flow*, mais dans la zone d’anxiété (défi > compétences). Inversement, du point de vue du professionnel, le jeu contre un débutant est tout autre chose qu'un défi. Le professionnel se retrouve dans la zone d'ennui (défi > compétences).

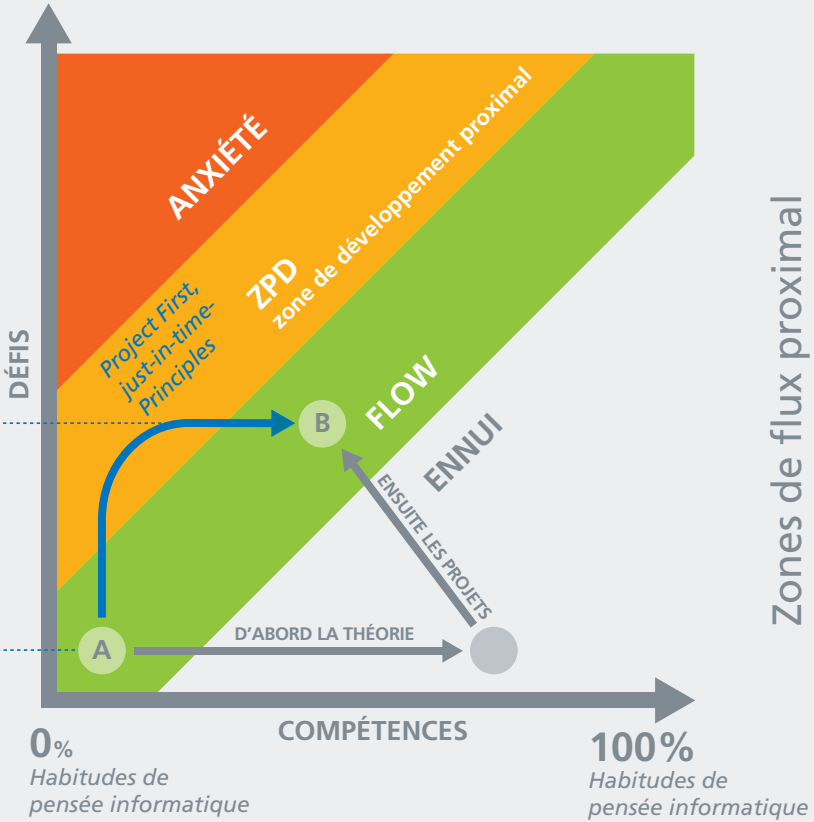
Vygotsky, qui a fait beaucoup de recherche sur le thème de l'apprentissage social, a créé la notion de *Zone of proximal development* (zone de développement proximal).

D'après Vygotsky, un apprenant se retrouve dans la zone de développement proximal lorsqu'il arrive à la limite de sa propre compréhension. Si l'enseignant aide l'élève juste à ce point, c'est là qu'il apprend le mieux. Les *zones of proximal flow* présentent une combinaison entre flow et zone de développement proximal. Elles définissent différentes zones d'apprentissage. La question est maintenant de savoir comment on apprend. Un apprenant au point A, avec peu de compétences et confronté à un faible défi, souhaite arriver au point B. Habituellement on suit un chemin indirect (ligne bleu) : on transmet d'abord les bases à l'élève, par exemple dans le domaine de l'algèbre linéaire, sans référence aux défis axés sur le projet. Ainsi les élèves apprennent comment manier les matrices, sans en comprendre les applications concrètes. Beaucoup plus tard seulement, probablement pendant leurs études, ils seront confrontés aux applications de l'algèbre linéaire, peut-être dans le cadre d'un cours d'infographie. Ils auront dû malheureusement faire un détour par la zone de l'ennui pendant de longues années.

L'approche de *Project First, just-in-time Principles*, utilisée dans le *Scalable game design*, est fondamentalement différente. Elle commence directement par un projet de jeu, sans que les élèves aient besoin de quelques compétences de programmation que ce soit. Elle emmène les élèves très rapidement à la limite de leurs compétences, mais les pilote consciencieusement dans la zone du développement proximal avec le soutien pédagogique adéquat et les ramène dans la *zone de flow* pour une expérience d'apprentissage optimale.

AgentSheet & AgentCubes





Le programme du *Scalable game design* comprend deux parcours progressifs qui soumettent les élèves pas à pas à de nouveaux défis. Le premier est axé sur les jeux et commence par des jeux simples du *style Arcade 1980*, comme *Frogger*. Des jeux plus difficiles tels que *Pac-Man* introduisent déjà des concepts d'intelligence artificielle, par exemple la façon dont les fantômes trouvent le chemin les menant à *Pac-Man* dans le labyrinthe. Dans des jeux encore plus exigeants, des modèles psychologiques tels que la pyramide des besoins de Maslow sont même mis en œuvre. La deuxième met l'accent sur les simulations. Les différents niveaux de défi de ces jeux et simulations peuvent être conçus comme des profils de compétences, basés sur des schémas de pensée informatique. Ces schémas décrivent typiquement l'interaction entre les objets à un niveau donné, indépendamment des détails d'un quelconque langage de programmation, mais aussi d'applications telles que jeux vidéo ou simulations. La collision entre deux objets constitue un exemple simple d'un tel schéma de pensée informatique. Dans le jeu *Frogger,* quand la grenouille essaie de traverser la route fréquentée, la collision entre la grenouille et une voiture signifie la mort pour la grenouille. Ce même schéma de pensée informatique est très fréquent aussi dans les simulations. Ainsi par exemple, la collision entre molécules est très importante en chimie et la collision entre êtres humains et virus contagieux qui se rencontrent peut être utilisée pour simuler le concept d'une épidémie.

Schémas de pensée informatique

L'utilisation explicite de schémas de pensée informatique est très importante pour une transmission du savoir qui doit comprendre le transfert par la pensée informatique. En règle générale, le transfert de connaissances ne se fait pas tout seul, même si du point de vue de la personne enseignante une analogie d'idées existe. Donc, si un ou une élève est en mesure de construire un jeu, cela est loin de signifier qu'il ou elle pourrait créer une simulation, même si les concepts fondamentaux du jeu et de la simulation sont similaires.

La liste des schémas de pensée informatique comporte des interactions relativement simples entre les objets.

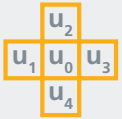
- **Entrer en collision :** deux objets ou plus se percutent, p. ex. dans le jeu *Frogger*, la grenouille peut se faire écraser par une voiture.
- **Pousser/faire glisser :** une personne pousse une brouette.
- **Tirer :** un train tire plusieurs wagons.
- **Générer :** quelque chose de nouveau apparait, autrement dit est généré. Le distributeur généré des billets.
- **Absorber :** un aspirateur absorbe les miettes sur le sol.
- **Transporter :** une planche à roulettes transporte un adolescent.

Il existe en outre des schémas de pensée informatique plus complexes, applicables dans des jeux et simulations. La diffusion par exemple est un schéma de pensée informatique très puissant. En 1952 déjà, le célébré mathématicien et informaticien Alan Turing a défini la diffusion comme étant le déplacement d'agents [chimiques] (particules) de régions à forte concentration vers des régions à concentration plus faible. Il utilisait l'ordinateur pour calculer la diffusion. Dans une grille à deux dimensions, on peut exprimer la diffusion par l'équation suivante :

$$u_{0,t+1} = u_{0,t} + D\sum_{i=1}^{n}(u_{i,t} - u_{0,t})$$

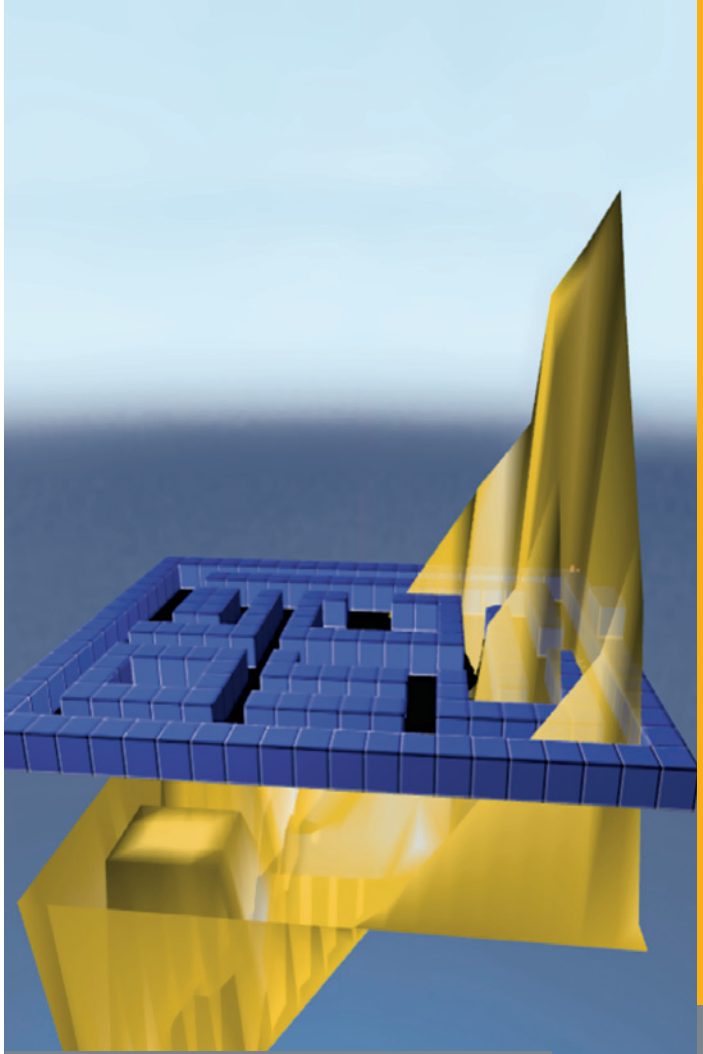
DANS LEQUEL:

- n = nombre d'agents voisins utilisé comme entrée pour l'équation de diffusion
- $u_{0,t}$ = valeur de la diffusion de l'agent centre
- $u_{i,t}$ = valeur de diffusion de l'agent voisin(i>0)
- D = coefficient de diffusion [0..0.5]



Ces concepts, au premier abord très complexes, peuvent être rendus accessibles et compréhensibles pour les élèves, afin qu'ils soient en mesure de résoudre des problèmes compliqués de façon élégante. Dans un jeu *Pac-Man*, la diffusion peut être utilisée pour permettre aux fantômes de trouver *Pac-Man* partout dans le labyrinthe en faisant appel à l'intelligence artificielle. La diffusion peut expliquer dans cet exemple comment *Pac-Man* diffuse une

odeur que les fantômes suivent. Des enfants de 10 ans sont déjà capables de l'appliquer. Cela signifie que, d'une part, l'influence de la motivation est fortement sous-estimée: pour l'élaboration de jeux électroniques, la motivation est souvent si grande que de enfants qui ne s'intéressent ni aux ordinateurs ni à la programmation montrent soudain des aptitudes mathématiques analytiques que personne n'aurait jugées possibles. La raison en est qu'ils résolvent des problèmes qui les intéressent vraiment et identifient ainsi les mathématiques comme un outil fantastique. D'autre part, un outil de pensée informatique tel qu'*AgentCubes* joue un rôle décisif : il facilite la résolution du problème par la visualisation. Dans *AgentCubes*, on peut par exemple insérer des tracés en 3D qui représentent la distribution de l'odeur dans le monde du jeu comme valeur de la diffusion. Un pic au-dessus de *Pac-Man* montre dans ce cas la répartition de la diffusion. L'utilisateur peut alors comprendre l'algorithme de recherche comme un concept qui sera utilisé par les fantômes pour trouver *Pac-Man* (fig. à droite).



AgentCubes

UN MODÈLE SCOLAIRE

Comment associer la pensée informatique au modèle scolaire suisse ? L'accent est mis avant tout sur le primaire. La pensée informatique doit débuter ici en tant que compétence clé. Du point de vue de la motivation aussi, la formation au niveau du primaire est le point crucial. Vers l'âge de 11 ans, beaucoup d'enfants déjà décrètent, à tort, qu'ils ne sont pas faits pour les mathématiques ou l'informatique. Qui a fait cette constatation un jour, ne sera plus guère disposé par la suite à aborder ce sujet.

Écoles primaires

Quand on catégorise la pensée informatique, comme décrit plus haut, en formulation du problème, représentation d'une solution et exécution de la représentation de la solution, on constate que ces concepts, ou du moins des concepts très similaires, existent déjà dans les plans d'études et sont mis en œuvre avec succès. Ceci est important du fait que les enseignant-e-s du primaire ne sont pas des spécialistes et doivent à de rares exceptions près enseigner toutes les matières, de la lecture au sport. Par conséquent le concept d'éducation informatique doit être inclusif et non exclusif. L'éducation informatique au primaire ne doit donc pas être isolée, mais au contraire intégrable aux autres disciplines, afin qu'elle ne soit pas considérée par le personnel enseignant comme une concurrence vis-à-vis des autres matières. Il s'agit donc d'un principe d'enseignement « transversal ». En tant que compétence clé du 21ème siècle, au même titre que la lecture et l'écriture, la pensée informatique peut être utilisée en soutien des domaines les plus divers :

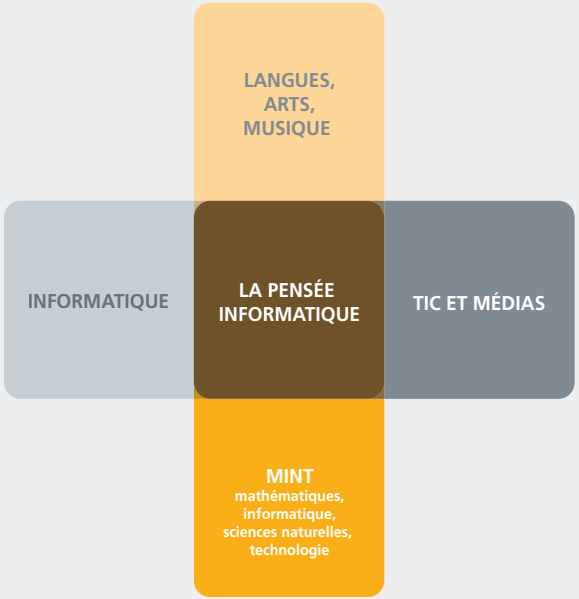
- **Langues/arts/musique/sport :** le processus de conception d'un jeu utilise une analyse grammaticale de la description de jeux ou de simulations portant sur les noms et les verbes, afin de comprendre les objets et leurs interactions. Ceci est très utile dans les cours de langues, y compris pour les langues étrangères. La création d'objets de jeux et de simulations en 2D ou 3D comporte des éléments importants pour les cours du domaine artistique. Par la combinaison de la synthèse sonore assistée par ordinateur (p. ex. avec MIDI) et de concepts de programmation, nait un concept synergique de pensée musicale informatique.

Dans le sport, des jeux comme *Frogger* peuvent être vécu sans l'utilisation d'ordinateurs (Notion Informatique *Unplugged*).

- **MINT (mathématiques, informatique, sciences naturelles, technologie) :** des jeux simples associent déjà des concepts mathématiques importants à des concepts de jeu : dans un jeu *Frogger*, une grenouille doit traverser une route fréquentée sans entrer en collision avec les voitures. Le degré de difficulté du jeu dépend directement de la fréquence et de la probabilité avec lesquelles de nouveaux véhicules apparaissent. Les simulations, comme par exemple la simulation d'un incendie de forêt ou d'une coulée de boue, permettent aux élèves d'expérimenter les principes des sciences naturelles au lieu de les apprendre simplement par cœur.

- **Informatique :** la pensée informatique est compatible aussi avec l'enseignement traditionnel de la programmation. Les schémas de pensée informatique peuvent être appliqués par les élèves dans les langages de programmation les plus divers. *AgentSheets* et *AgentCubes* en ligne jettent un pont vers la programmation impérative en proposant une fonction que l'utilisateur peut employer pour voir le code source (Java ou JavaScript).

- **TIC et médias :** au cours de l'élaboration de jeux et de simulations dans le contexte de la pensée informatique, de nombreux points de l'éducation aux médias, notamment le rapport aux médias, sont abordés. Grâce à la programmation, les enfants ont notamment l'opportunité de prendre leurs distances avec la société de consommation en créant eux-mêmes. L'idée que le volet pensée informatique des TIC et des médias comprend explicitement la programmation et ne se limite pas à la dactylographie ou au traitement de texte est très importante.



Secondaire 1

Le secondaire 1 doit prendre la suite de l'école primaire en ce qui concerne la pensée informatique. La compréhension de la pyramide des besoins de Maslow par exemple permet de réaliser des jeux très exigeants tels que les Sims. L'accent devrait être mis sur les simulations, qui sont utilisables surtout dans les sciences naturelles. De plus la base d'outils employés en sciences naturelles doit être considérablement étendue. Les tableurs, qui sont typiquement produits par simulations autoprogrammées, peuvent p. ex. être utilisés comme outils pour l'analyse et la visualisation de grandes quantités de données. Le concept global à ce niveau doit encore être inclusif, complété par des cours axés sur l'informatique donnés par des spécialistes.

Secondaire 2

Beaucoup d'écoles du secondaire 2 proposent déjà l'informatique en tant qu'option complémentaire enseignée par des personnes qualifiées possédant des connaissances informatiques formelles. La participation, surtout des filles, est cependant très faible. Tant qu'une éducation informatique généralisée aux niveaux primaire et secondaire n'existe pas, les écoles du secondaire 2 doivent offrir des cours d'informatique obligatoires, adaptés à la grande majorité des élèves. Ces cours doivent se concentrer sur la motivation, avec des approches passionnantes et neutres en termes de genre, comme par exemple la conception de jeux.

Les auteurs de ces documents étayent leurs propres arguments qui ne reflètent pas nécessairement l'opinion de celle de la Fondation Hasler.

HASLERSTIFTUNG

Hasler Stiftung
Hirschengraben 6
CH-3011 Bern
Tel. +41 31 381 41 41
Fax +41 31 381 67 00

www.haslerstiftung.ch
www.fit-in-it.ch