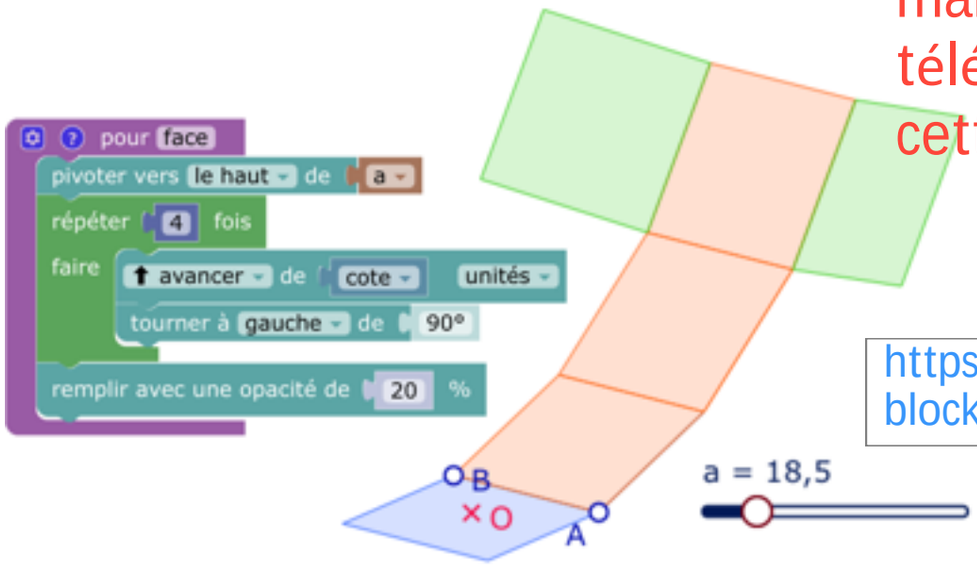
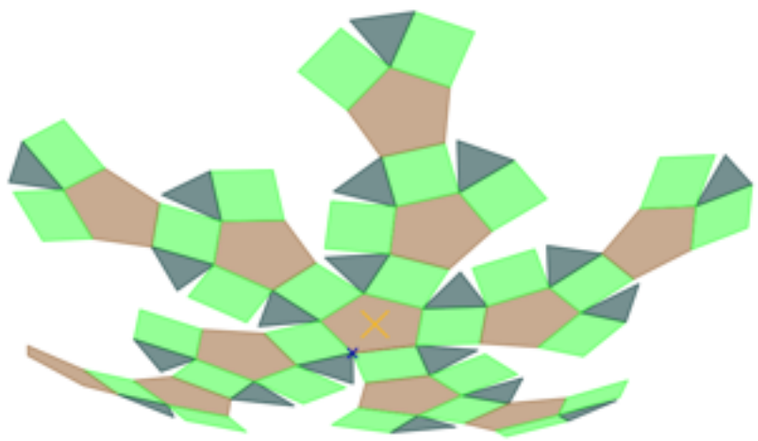
Blockly et tortue dynamique avec DGPad

Ce PDF est téléchargeable et les figures mentionnées sont manipulables en ligne et téléchargeables également à cette adresse

https://sites.google.com/view/blocklydgpad/

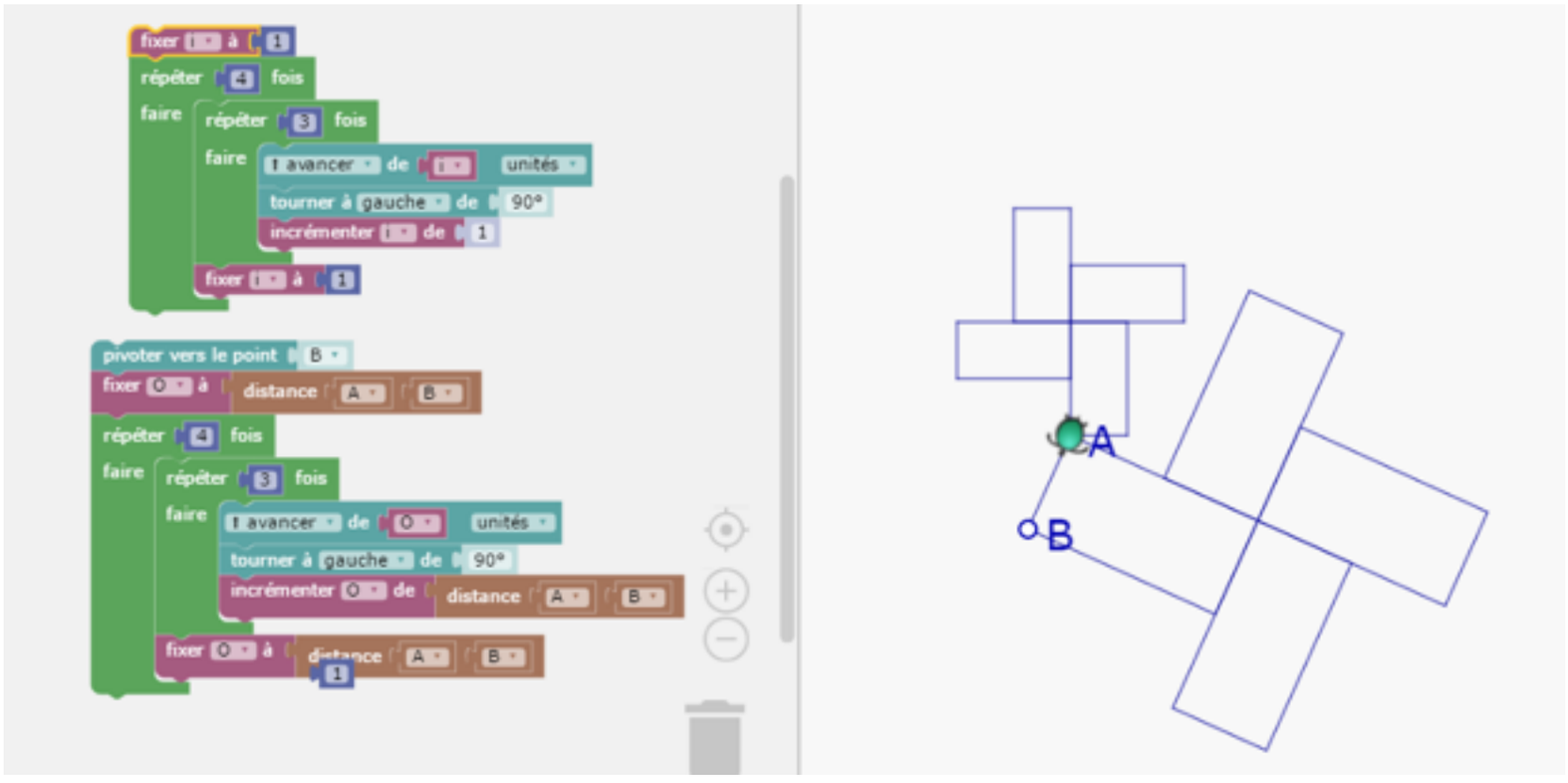




Présentation de l’ouvrage

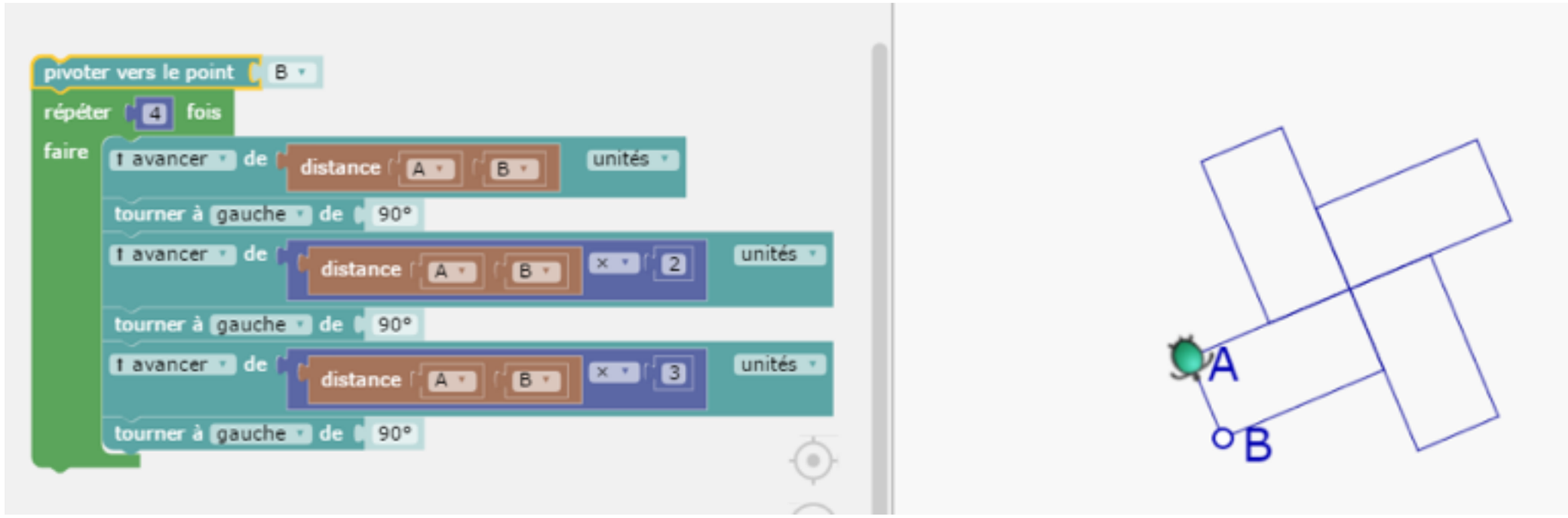
Pour l’enseignement du code conformément aux programmes de 2016 de collège, les enseignants disposent de nombreux outils possibles, riches et intéressants. En France, ce sont les professeurs de mathématiques qui enseignent le code, et c’est l’occasion de réfléchir aux changements de cadre que permet cette nouvelle mission des enseignants de mathématiques.

Par exemple il est surprenant de voir que chez certains élèves, la difficile question du nombre 1 comme neutre de la multiplication se retrouve aussi dans leur pratique de programmation, comme un invariant placé sous un nouvel éclairage. L’apprentissage de la programmation peut alors apporter un autre regard, plus opérationnel. En guise d’introduction à ce qui va être important dans ces pages, voici deux programmes d’élèves de 4°, sur le tracé de figures construites autour du cycle 1-2-3 (spirolatères, illustrés plusieurs fois dans ce livre). Une conception additive de la séquence 1-2-3 produit des solutions simples.
Une conception multiplicative de cette même séquence pose des difficultés.



Spirolatère de base, de cycle (1, 2, 3). Démarche additive : ici 2=1+1 et 3=2+1.
Même démarche pour une unité standard ou construite sur un segment [AB].

Dans le cas d'une démarche multiplicative, l'élève est amené à supprimer une boucle et construire une à une les traces de la tortue. Pour éviter cela, il aurait fallu multiplier par 1, ce qui n'est pas du tout naturel pour de nombreux élèves.



Le fait que 1 ne soit pas outillé, opérationnellement et réellement perçu, conceptuellement, comme neutre de la multiplication se traduit par la perte de la boucle interne du programme.

Dans cette situation, la présentation du bloc **compteur**, avec sa variable qui peut prendre la valeur 1 dans une multiplication, est d'abord une mise en œuvre efficace qui permet de retrouver, avec deux boucles imbriquées, une structure proche de la démarche additive. Elle est aussi l'occasion, pour l'enseignant de mathématiques, de faire le lien avec la représentation de l'unité dans des questions algébriques comme factoriser 3n+3 où le second 3 doit être perçu comme 3x1.

La pratique d'une programmation ludique, avec ici le projet de produire une figure dynamique, en manipulation directe, est une opportunité de faire vivre autrement les mathématiques, que ce soit l'arithmétique élémentaire, ou la géométrie pour laquelle nous allons faire vivre « par la tortue », autrement, les propriétés des quadrilatères.

De manière plus surprenante, parce qu'elle s'inscrit dans un environnement dynamique, la tortue va être une opportunité d'algébriser certains problèmes rencontrés et, par là même, les résoudre sans aucun calcul, simplement parce qu'ils ont été posés de manière algébrique. Même si un chapitre va y être consacré, cette attitude sera un peu partout présente.
C'est important car elle montre l'algèbre pour ce qu'elle est réellement : une façon de regarder un problème, une conceptualisation différente (de l'arithmétique) de la modélisation des situations

Cet ouvrage traite de la pratique de **Blockly**, implémenté dans un **environnement dynamique** de géométrie. Cette rencontre entre une programmation visuelle dynamique et un support d'objets en manipulation directe ouvre des perspectives d'une grande originalité dans plusieurs domaines – comme **l'algébrisation de la tortue**, déjà mentionnée.

Deux autres domaines tout aussi novateurs vont être abordés. En effet, la tortue dynamique plongée dans un **environnement 3D** renouvelle complètement le rapport que nous pouvons avoir à l'enseignement de la 3D, par des activités là encore ludiques, créatives, simples et porteuses de réflexions géométriques. Un chapitre y est consacré et, parce que l'auteur s'est aussi fait plaisir, une première galerie 3D est proposée.

Enfin, de par une implémentation bien particulière comme **comportement d'objet**, Blockly permet une **rupture maîtrisée du déterminisme** de la géométrie dynamique, Cela autorise de mettre en œuvre des activités qui ne sont pas possible sans cela (voir les derniers chapitres). Le logiciel CaRMetal permettait déjà cette rupture maitrisée du déterminisme. Mais ici, de plus, le tactile habille certaines de ces activités d'une dimension kinesthésique qui fait sens en pratique en classe aussi bien dans le domaine géométrique que dans le champ fonctionnel.

Dans un chapitre préliminaire, on présente l'interface de DGPad. En effet, un minimum d'appropriation des idées qui portent la spécificité de son interface (**outils infixés** au lieu d'être **préfixés**) permet d'être plus efficace ensuite dans la programmation.

Bonne découverte et bonne exploration de cette extraordinaire rencontre entre la programmation par blocs et la géométrie dynamique qu'est le logiciel DGPad.

Le logiciel DGPad

Ce livre se propose d'explorer les possibilités de l'implémentation de Blockly et de la tortue dans l'environnement de géométrie dynamique DGPad sous différents points de vue, technique, didactique, ou simplement mathématique. Avant cela, ce chapitre présente l'interface de DGPad. Sans être exhaustif, ni aucunement un manuel d'utilisation, il s'agit de faire un tour d'horizon de l'interface du logiciel pour être opérationnel dans l'utilisation de ce livre dynamique.

DGPad est une application de géométrie dynamique orientée tactile, mais disponible en WebApp sur ordinateur. Elle a l'avantage d'avoir un module 3D particulièrement facile à utiliser pour les élèves, et encore plus dans l'environnement de la tortue 3D.

Un nouveau concept d'interface - les palettes contextuelles

Vous avez déjà pratiqué au moins GeoGebra ou Cabri, ou encore CaRMetal : tous ces outils ont une interface classique pour leurs outils. On dit que les outils sont **préfixés** : on choisit l'outil, puis les objets associés. On ne s'en aperçoit pas toujours (à force, si quand même) car on a l'habitude et on va assez vite, mais cette interface est particulièrement gourmande en clics, surtout lorsqu'il y a des menus déroulants.

L'auteur de DGPad a réfléchi à une interface plus efficace nécessitant moins de contact (de « touché » sur tablette, ou clics de souris sur ordinateur). Il a choisi une approche radicalement nouvelle : celle des palettes contextuelles à la situation. Informatiquement, c'est le choix d'une programmation objet centrée sur les objets géométriques et non plus sur les outils. C'est comme si on venait réveiller un objet (point, droite, segment, polygone, cercle) et qu'il nous proposait tout ce qu'on peut faire avec lui. Les palettes sont donc différentes pour un point, une droite, un segment, un cercle …

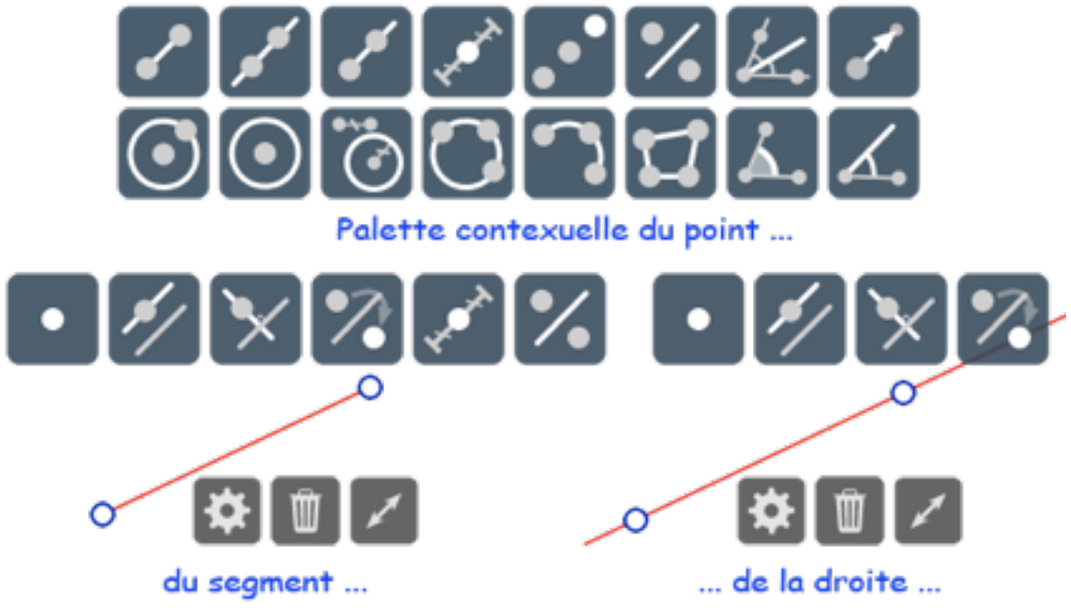


Tableau de bord - modes « consultation » et « standard »

Conceptuellement, on peut dire qu'on passe d'une interface où les outils étaient préfixés (tous les autres logiciels de géométrie dynamique), à une interface où ils sont **infixés**. C'est une nouvelle habitude à prendre, surtout pour les outils à trois entrées ou plus (cercle circonscrit, angle, polygone) : le pointeur tactile - le doigt - ne se comporte pas comme une souris. Quand on lève le doigt, le système ne sait pas où il va apparaître, alors qu'on sait toujours où est le pointeur de la souris, même si on ne clique pas dessus : on passe d'un pointeur continu (ordinateur) à un pointeur discontinu (tablette). L'interface doit en tenir compte pour forcer le lieu où l'on attend le pointeur pour conserver des constructions en anticipation - avec un « fil à la patte ».

Mode standard et mode consultation

Le tableau de bord est généralement sélectionné sur un mode donné. Or la plupart sont des interrupteurs que l'on peut désactiver. Quand aucun outil n'est sélectionné, on est en **mode consultation** : on peut agir sur la figure sans créer des points par inadvertance.

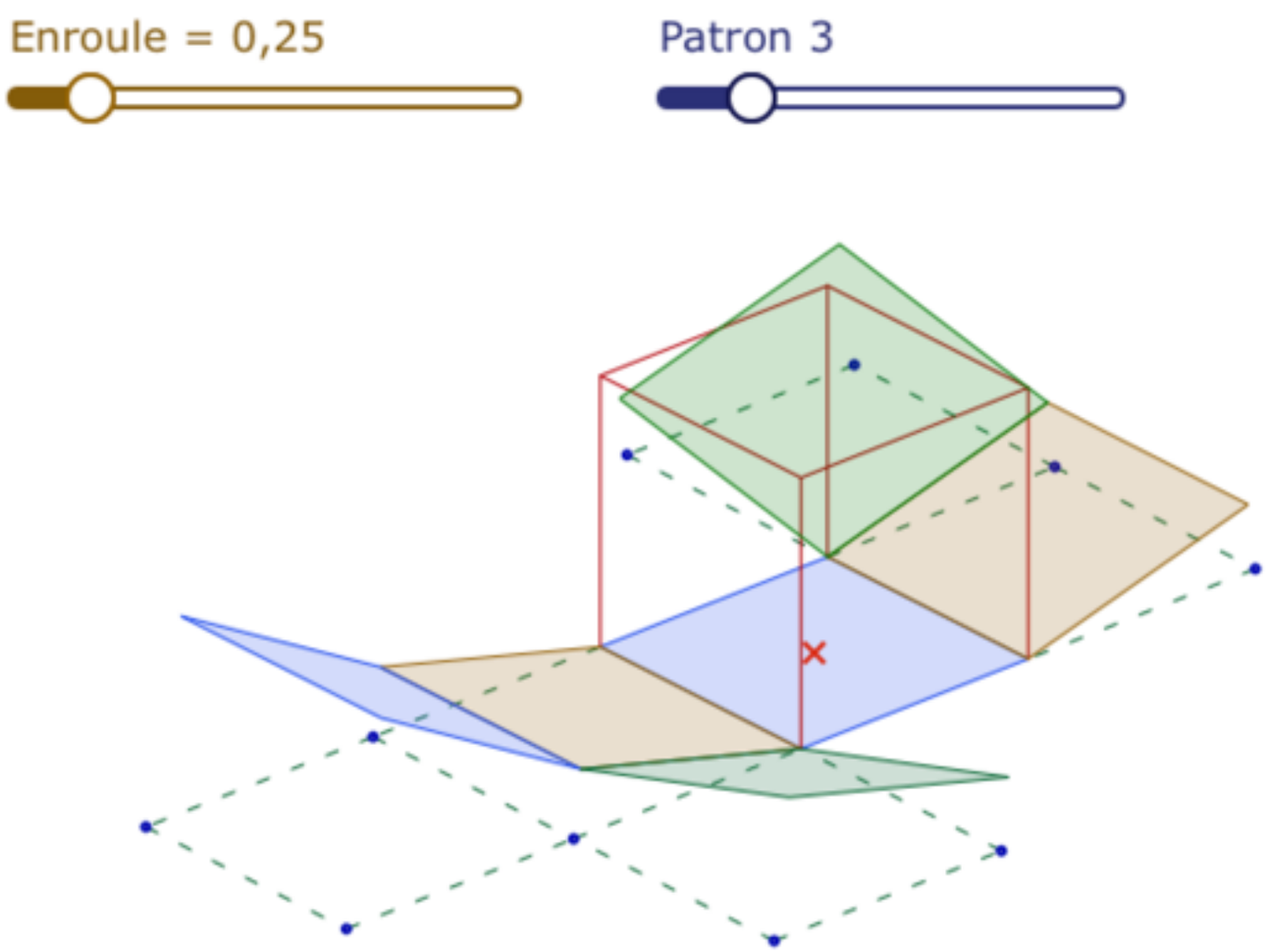
Le mode consultation est nécessaire car cette interface préfixée est très ouverte.

Dans ce mode, en particulier, le repère 3D est plus rapide et il se modifie avec un seul doigt (tablette, trackpad) ou clic-gauche-glisser (souris) au lieu de deux doigts (respectivement clic-droit-glisser). Quand on charge une figure, par défaut elle est en mode consultation. Un «touché » (un clic sur ordinateur) sur un item du tableau de bord, l'application passe en **mode standard**, c'est-à-dire en mode création d'objets. Toutefois, il faut choisir le **pointeur** usuel (la flèche à gauche) pour effectivement créer des objets.

En classe – surtout sur tablette – on peut apprendre aux élèves à passer d'un mode à l'autre rapidement pour manipuler une figure « en consultation », en particulier une figure 3D.

Dans la page suivante, on propose une figure 3D – **pliage des 11 patrons du cube** – que vous pouvez tester pour modifier les curseurs et faire tourner le repère, en passant du mode standard au mode consultation. On privilégiera ce dernier pour manipuler des figures riches sur tablette. C'est moins indispensable quand on a une souris.

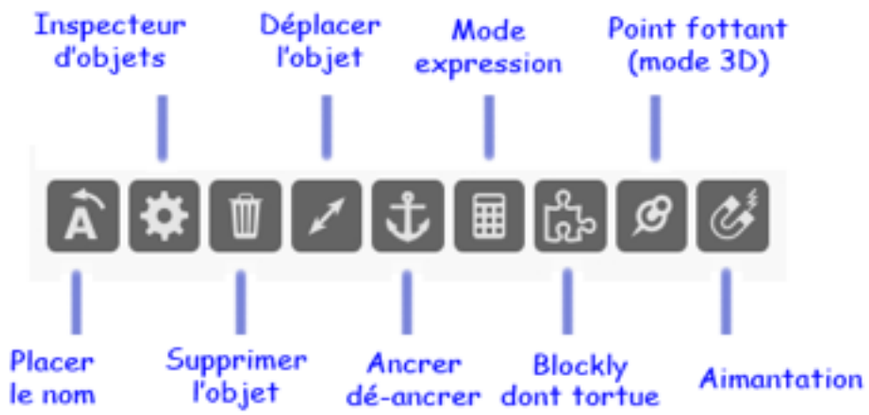
Figure dynamique 2.1 – Pliage des 11 patrons du cube



Manipuler la figure en mode consultation (aucun outil sélectionné) et en mode standard (la flèche sélectionnée).

La palette de comportements

La palette de comportement est elle aussi **contextuelle**. La plus importante est celle des points.



Cette palette de comportement contient :

• Des comportements spécifiques (d'où son nom) : ancrage et aimantation en 2D, point flottant en 3D.

• Des raccourcis du tableau de bord (inspecteur, suppression, calculatrice).

• Des réglages qui ne se font que sur la fenêtre elle-même (place du nom des points et déplacements d'objets superposés).

• L'accès à l'interface Blockly, et en particulier la tortue dynamique et les DGScripts : **l'interface Blockly est donc un comportement et, comme tel, ne crée pas d'objets**.

D'autres comportements sont ouverts à d'autres situations. Ainsi, si un point est **sur objet**, il peut être **animé** - le ressort, dernière icône à droite - avec une interface qui rend hommage à celle de Cabri, tout en étant plus moderne, en particulier parce que l'on peut continuer à créer des objets pendant des animations.

Les **expressions** disposent, elles aussi, d'un comportement spécifique – le **?** ci-dessous – qui permet de construire des figures pour les placer dans des quiz dynamiques avec les outils, du même auteur, les DocTools.

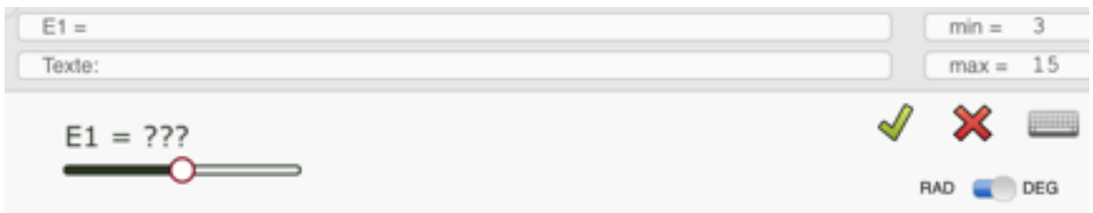
On notera que les expressions – dans certains contextes – peuvent avoir aussi le ressort d'animation.

Les curseurs

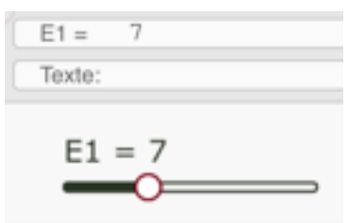
Même si on se concentre surtout sur la programmation, on utilisera souvent des curseurs. Les curseurs sont simplement des expressions qui ont des bornes.

Quand on ouvre le module expression (icône calculatrice), le clavier scientifique de DGPad s'ouvre et il faut d'abord faire un « touché » (un clic) dans la ligne d'expression – pas trop à gauche – pour l'activer.

Pour que cette expression soit reconnue comme curseur, il suffit de remplir les données **min** et **max** à droite. C'est l'occasion de faire remarquer à des élèves que, si la variable E1 est créée par ses bornes, elle n'est pas initialisée, d'où le retour **???** pour la valeur de la variable.

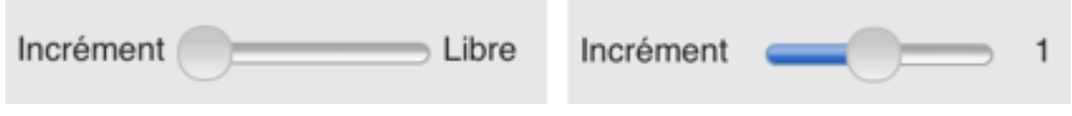


On termine par **initialiser** le curseur. Dans tous les cas, il est essentiel de **valider la création** de l'expression en sélectionnant le bouton vert.



On peut ensuite déplacer le curseur, en particulier car par défaut, la prochaine expression sera créée au même emplacement.

Comme tous les objets, les réglages spécifiques se font dans **l'inspecteur d'objets**, l'icône **roue** du tableau de bord. Certains curseurs peuvent représenter des paramètres entiers.
Il est important alors – surtout s'ils vont être la borne d'une boucle dans un bloc de programmation – de rendre la variation du curseur **discrète et entière**. Pour cela, il suffit de fixer l'incrément à 1 :

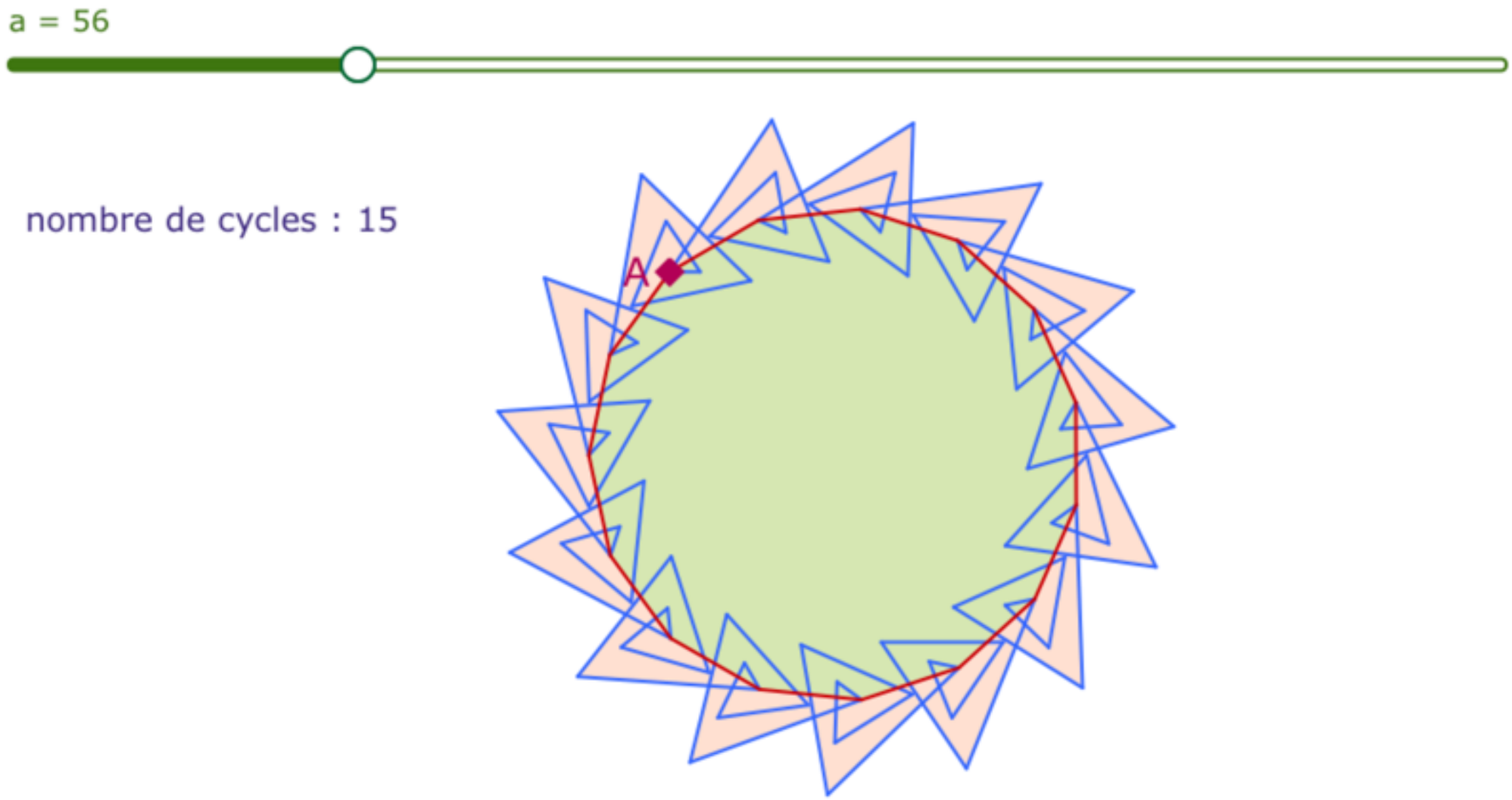


Cette incrémentation s'applique à d'autres objets, et en particulier aux points du plan : un incrément à 1 place un point sur la grille de base du logiciel; un incrément à 0,5 le place sur la demi-grille, ce qui est intéressant dans de nombreuses situation d'analyse par exemple.

On a donné ci-contre un exemple numérique élémentaire, mais les bornes d'un curseur peuvent être algébriques, en fonction d'autres variables de la figure, ainsi que des constantes, comme des multiples ou diviseurs de π par exemple.

Dans la page suivante, on propose d'explorer le concept de spirolatères (article Blockly sur le thème), constructions élémentaires basées sur l'itération d'un cycle de segments de longueurs entières consécutives, avec un angle constant entre deux segments consécutifs, itération jusqu'au retour au point de départ, quand c'est possible. On jouera donc sur le curseur angle pour observer la diversité des constructions.

Figure dynamique 2.2 – Spirolatère du 6-cycle standard (123456)
Le nombre de cycles, pour revenir au point A, dépend de l'angle a. La partie verte est le polygone des extrémités de cycles.



*On peut modifier l'angle **a**, déplacer le point **A**, et zoomer sur la figure avec deux doigts.*

Les listes

Les listes sont un objet important de DGPad, et plus encore depuis l'implémentation de Blockly. Déjà, un point est une liste à deux termes (point 2D) ou trois termes (point 3D).

Les listes générales peuvent être des listes de points ou de segments (ou encore de vecteurs, mais seulement avec Blockly). Si en général on choisit un type de liste lors de la création, on peut toujours le modifier à tout moment dans l'**inspecteur d'objets**.

La création de listes avant Blockly

Si on veut programmer une liste en JavaScript, on peut encore le faire - même si c'est devenu plus simple avec Blockly. Pour cela on utilise une **expression** que l'on appelle une « expression programme » qui est simplement un programme en JavaScript avec des simplifications d'écritures, comme *sqrt()* à la place de *Math.sqrt()* ou *π* à la place de *Math.PI()*.

Exemple de la modélisation du « cœur de tournesol »

On se propose d'ajouter une variable *angle* (un curseur) sur cette modélisation standard du cœur de tournesol donnée par les points P_k d'affixe $C\sqrt{k}e^{ik\theta}$, où l'angle θ est lié au nombre d'or φ par $\theta = \frac{2\pi}{1+\varphi}$ soit environ 137,5°. On remplace donc cet angle fixe (qui permet d'avoir des spirales liées au nombres de Fibonacci) par un curseur, nommé **ang** dans la suite. Dans une expression programme, on crée une liste par un programme classique JavaScript. Cette liste est renvoyée à DGPad (à la partie graphique) par la variable **placée après** le dernier point virgule (;) de cette expression programme. Ainsi, dans une expression, on rentre ce programme :

```
var tab=[];for (var k=1;k<=500;k++){var u=0.2*sqrt(k);var a=k*ang*π/180; var v=[u*cos(a),u*sin(a)];tab.push(v);};tab
```

L'expression devient une longue liste de 500 points - crochet de 2 coordonnées. Il apparaît alors deux icônes.

Que l'on choisisse les segments ou les points, on peut modifier ce choix et le régler finement, au dixième de pixel prés, dans l'inspecteur d'objets.

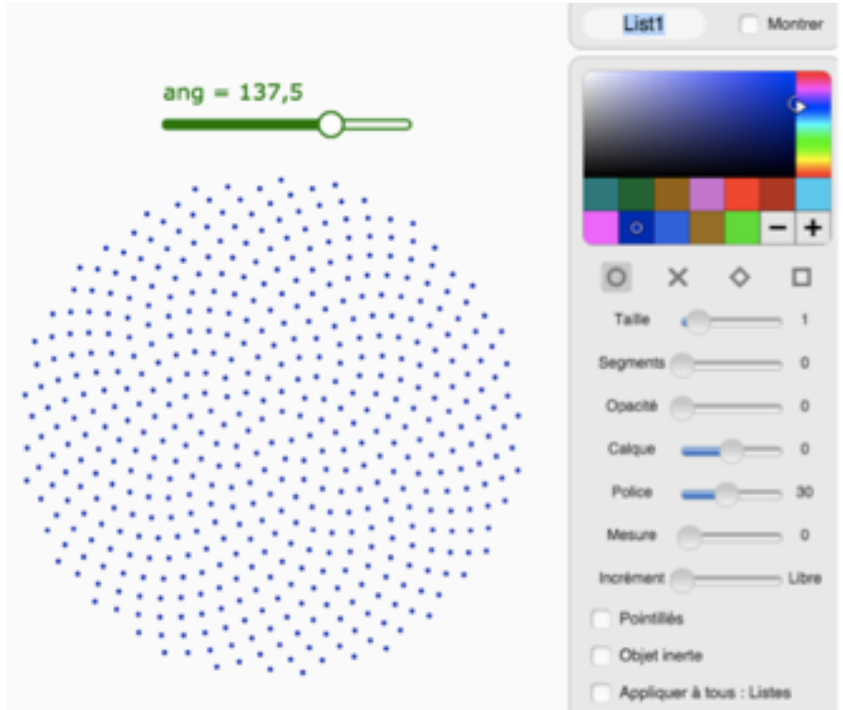
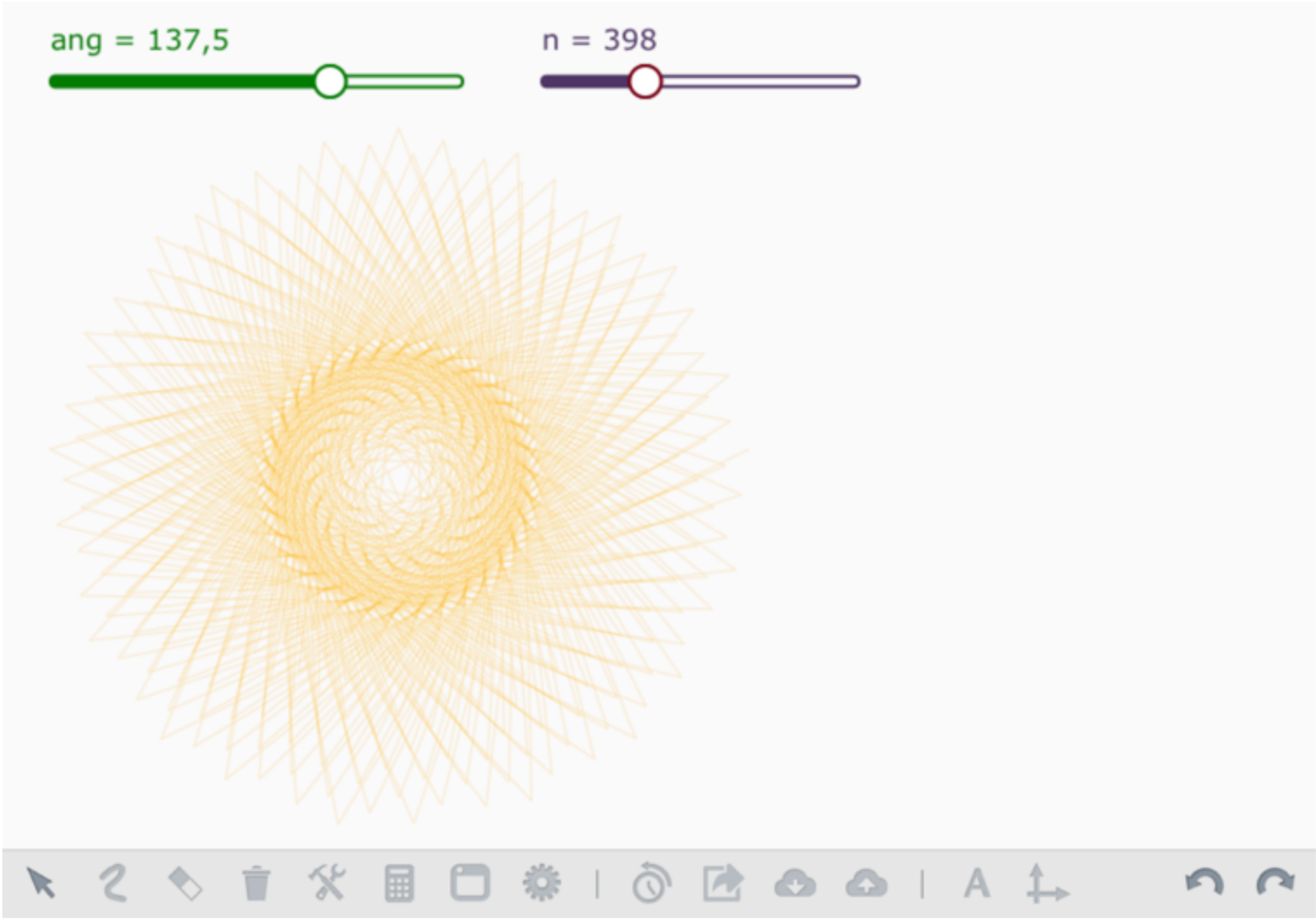


Figure dynamique 2.3 – Liste par expression standard – algorithme du « cœur de tournesol »



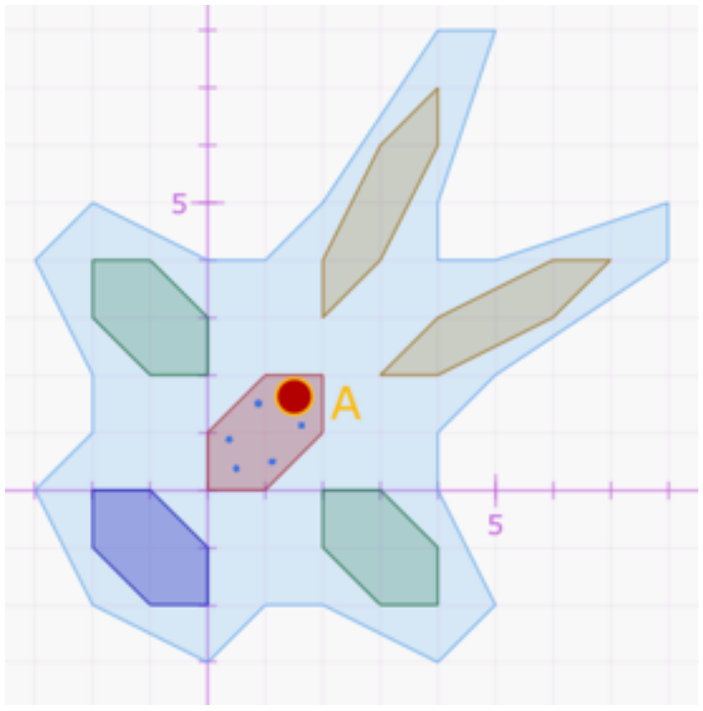
Activer l'inspecteur d'objet (roue) et modifier les tailles des points et des segments.

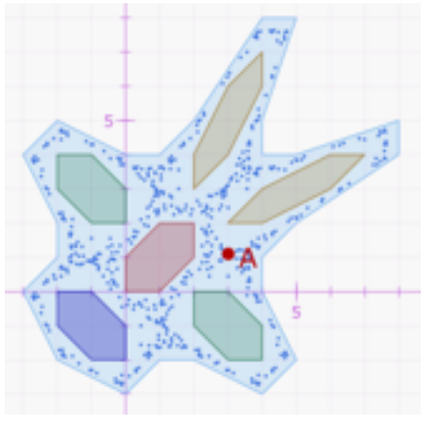
Autre exemple : l'algorithme du bonhomme de pain d'épice (Gingerbread Man)

Il s'agit de la suite de points $\begin{cases} x_{n+1} = 1 - y_n + |x_n| \\ y_{n+1} = x_n \end{cases}$

On se donne un point A de départ, comme condition initiale et on s'intéresse à l'évolution du système.

Le bonhomme est à l'envers, il a un cœur (rouge), une tête (bleue), deux bras (verts) et deux jambes (marrons). Tous sont à coordonnées entières
Le cœur est stable, les autres organes forment une orbite du point A quand il est dans l'un d'eux.

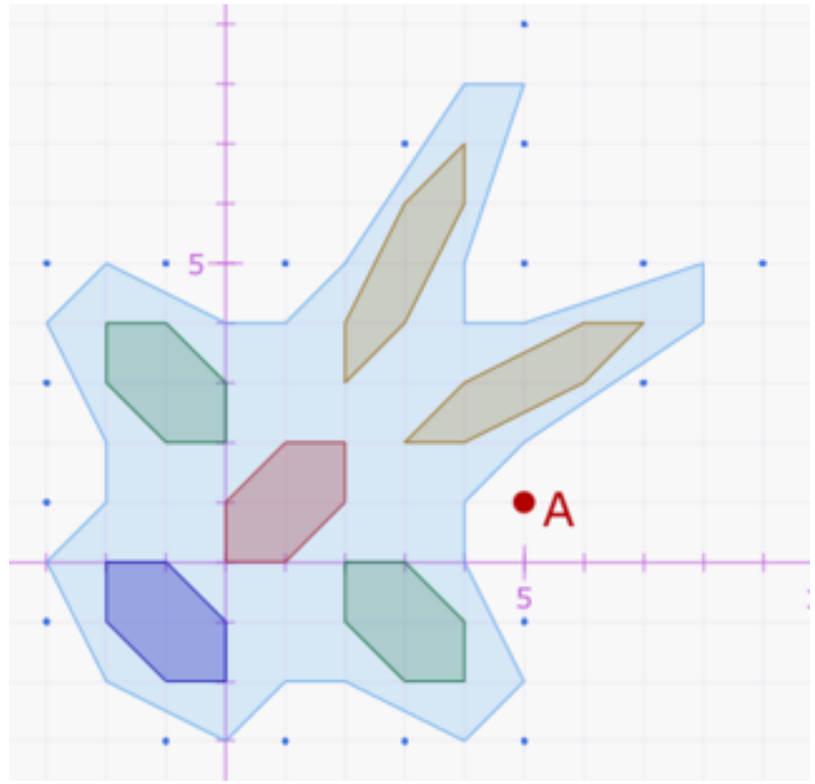


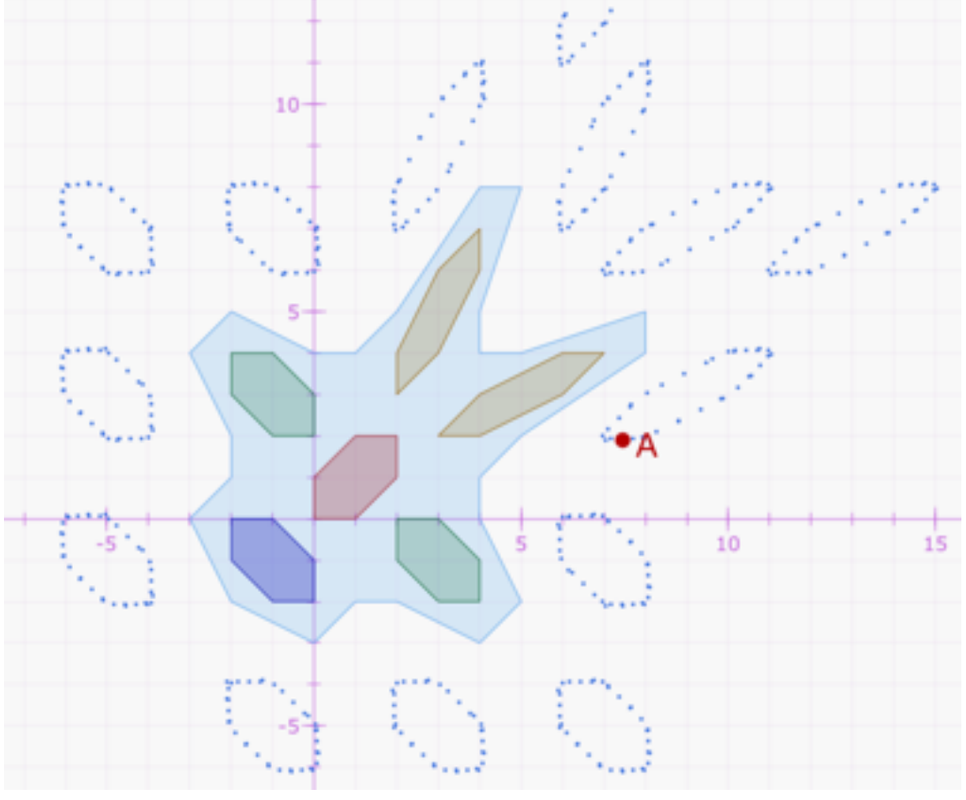


En dehors de ses membres, le bonhomme comme délimité sur ces illustrations est stable.

Dans la figure de la page suivante, penser à placer A au centre de la tête (-1,-1).

En dehors du bonhomme, de nombreuses situations sont stables ou forment elles aussi des orbites.
Par exemple placer A en le point (5,1), puis (6, 1). A est aimanté par ces points.





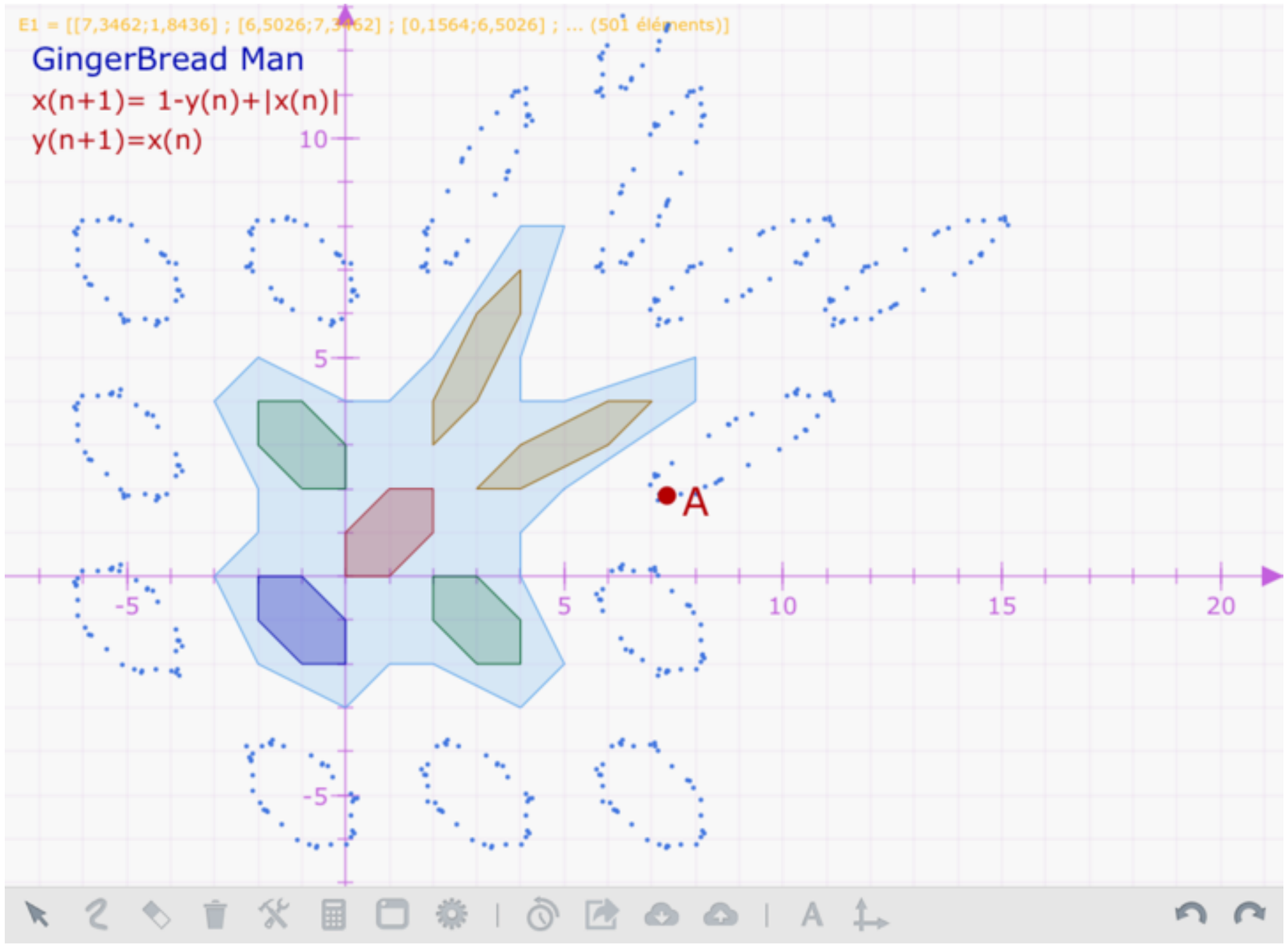
A est aimanté par divers points à coordonnées entières
de la figure.

Ne pas hésiter à régler le format de liste autrement, par segments, en particulier avec A en (-1, -1), en (6,1) et au voisinage de ces points.

Observer aussi le cas (immédiat) de A en (1, 1).

De nombreux travaux sur **GingerBread Man** sont disponibles sur le net.

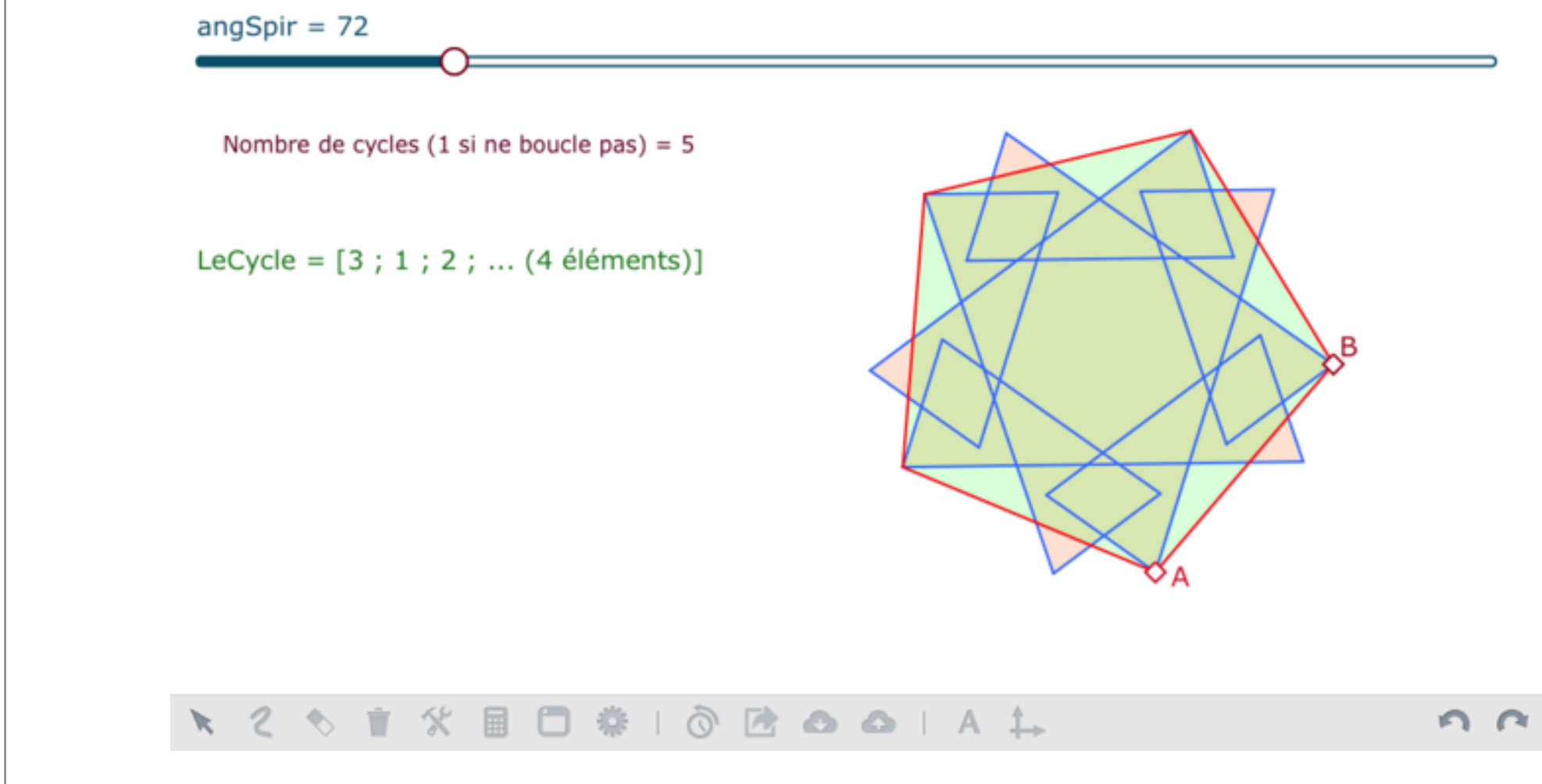
Figure dynamique 2.4 – Algorithme du bonhomme du pain d'épice



Exercice sur le passage d'une liste de points à une liste de segments avec l'inspecteur d'objets.

Modifier une liste : dans la figure suivante, sur les **spirolatères dynamiques**, un cycle quelconque, associé à un angle quelconque, va de **A** à **B**, côté prédéfini du polygone des extrémités. Vous pouvez **modifier la définition du cycle, agir sur A, B et l'angle.**

Figure dynamique 2.5 - Manipulation directe sur une liste paramètre dans le module expression



Touché prolongé (clic droit) sur la liste, choisir Expression dans la palette de comportement et modifier directement la liste dans l'expression, y compris sa taille. Le spirolatère s'adapte pour aller de A à B en un cycle (à l'ouverture 3141).

Un prochain chapitre traitera de la construction de listes spécifiquement depuis Blockly (ici fait avec la tortue).

Création d'objets - Régles pour une utilisation de base

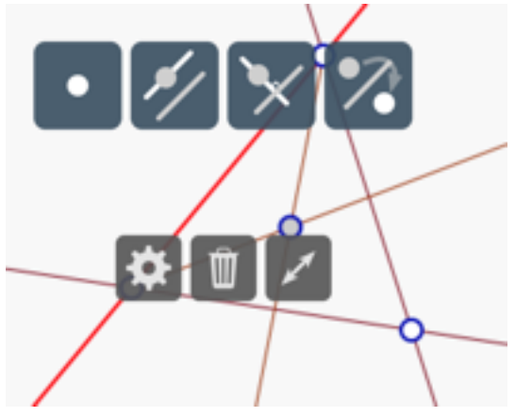
Même si ce livre est consacré essentiellement à la programmation, disons quelques mots sur les outils de base de création géométrique. En effet, l'interface du logiciel, entièrement pensée pour le tactile (mais utilisable sur ordinateur) diffère de celle des logiciels usuels de géométrie dynamique, essentiellement pour conserver l'anticipation des constructions (le «fil à la patte») y compris sur tablette alors qu'il n'y a pas de roll-over et donc que c'est a priori plus compliqué à installer.

La page suivante contient une feuille vide sur laquelle vous pouvez vous entrainer. Voici quelques consignes pratiques

Création d'objets à deux items

Quand on sélectionne un point, une palette de 16 items apparaît. On en choisit un – la droite par exemple, ou le segment – et on maintient le doigt sur l'outil pour le faire glisser à l'endroit où on veut créer – ou sélectionner – un autre point, afin de construite cette droite ou ce segment.

Ci-contre il s'agit de prendre une perpendiculaire à une droite donnée. On sélectionne la droite, on prend l'outil perpendiculaire et on maintient le doigt jusqu'au point d'où sera issue la perpendiculaire.



Intersection de deux droites

On touche l'intersection des deux droites, un item « point » (•) sur objet apparaît et reconnait l'intersection (deux droites sélectionnées).

Engagement direct

On peut créer des objets en créant ses constituants à la volée. Ci-contre on crée un cercle avec l'item « centre-point », le point étant construit pendant celle du cercle, reconnu comme intersection de deux droites (sur-lignées en jaune).

Objets à trois items

Pour prendre une bissectrice il faut 3 points. Quand on lâche le second point une icône apparaît (ci-contre), on reprend cette icône pour terminer la construction avec anticipation (la bissectrice est pré-tracée).

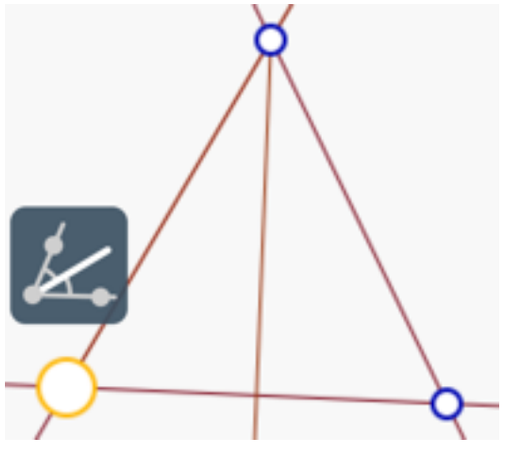
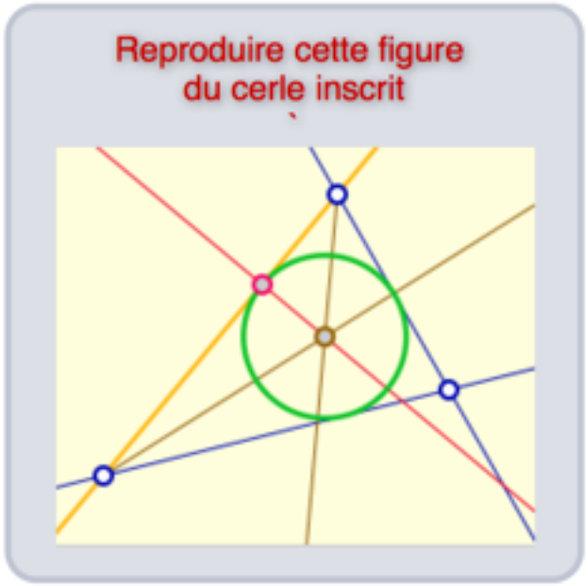


Figure dynamique 2.6 - Exercice d'apprentissage de l'interface de géométrie - Manipulation des divers outils



Pour modifier les couleurs, ouvrir l'inspecteur d'objet (la roue) et sélectionner les objets à modifier.
A noter qu'on peut aussi agir sur la figures dans le widget sur fond jaune.

Nommage des points

On peut envisager de nommer les points soit par anticipation, soit après la construction.

Nommer par anticipation

La fonctionnalité est active dès qu'on ouvre le panneau (icône A ci-dessous). On dispose alors de différentes options.

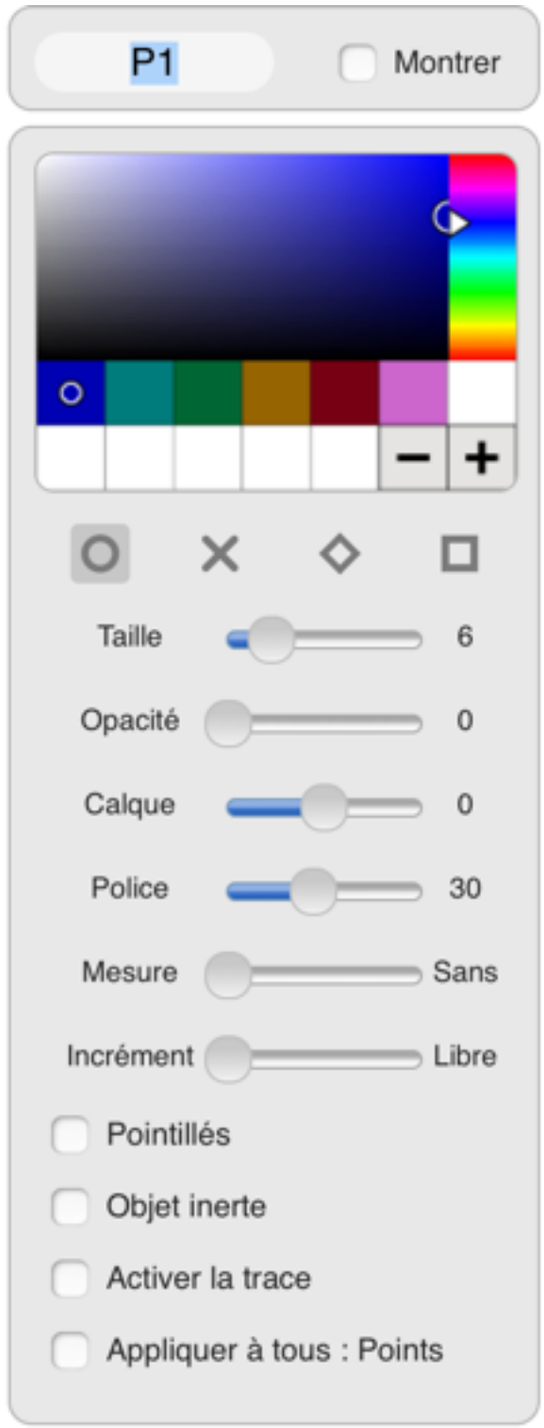
Nommer avec l'inspecteur d'objets

Sur tablette, il faut entrer les noms un par un, en validant par « Accéder » sur le clavier alphanumérique à chaque point, alors que sur ordinateur, le clavier ne monte pas sur la figure et un simple « retour chariot» suffit, ce qui permet de nommer plusieurs points à la suite.

La technique est finalement assez identique, mais elle est plus rapide sur ordinateur.

On voit, ci contre, les différents réglages possibles.

Au moment où ce livre a été rédigé, l'affichage des noms d'objets n'a été implémenté que pour les points. Mais les noms d'objets servent comme variable des expressions.



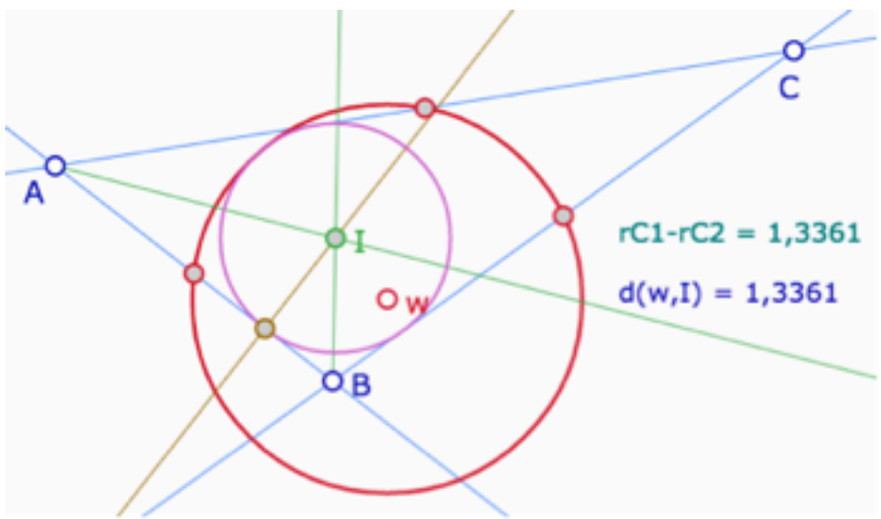
Les expressions

On a déjà vu des utilisations assez sophistiquées des expressions sous forme de programmes JavaScript mais, bien entendu, les expressions interviennent dans des situations plus élémentaires, comme l'utilisation de variables.

Le nom d'un objet comme variable algébrique

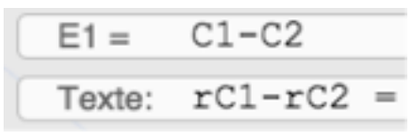
Comme tous les logiciels contemporains de géométrie dynamique, le nom d'un objet est une variable qui peut être utilisée dans une expression ou un programme. Ainsi, le nom d'un segment désigne sa longueur, celui d'un cercle son rayon, celui d'un polygone son aire, *algébrique*, dépendant de l'orientation du polygone.

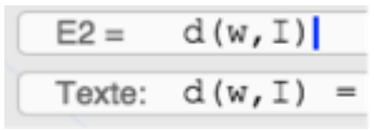
Dans la figure suivante, **I** est le centre du cercle inscrit, et **w** le centre du cercle passant par les trois milieux des côtés du triangle initial. On veut illustrer (ou conjecturer selon le contexte) que ces deux cercles sont tangents.



Remarque pratique : si vous vouliez finaliser cette construction sur la figure faite précédemment, il suffit d'utiliser l'item « cercle par 3 points » pour construire le cercle d'Euler. Alors le centre existe mais est caché. Il faut le faire apparaître, par l'outil gomme, et lui donner un nouveau nom.

On crée ensuite deux expressions, une est la différence des deux rayons des cercles (leurs noms) et l'autre la distance entre les deux centres.





Que cette approche illustre effectivement que les cercles sont tangents est un travail – en général non trivial – sur l'inégalité triangulaire.

Le module expression comme grapheur

De même que si une expression a des bornes elle est interprétée comme un curseur, de même si elle contient une variable standard comme x, y ou t, cette écriture est considérée comme une relation fonctionnelle . Le module ***expression*** devient un ***grapheur*** et propose de tracer la représentation graphique, avec une nouvelle icône de type parabole.

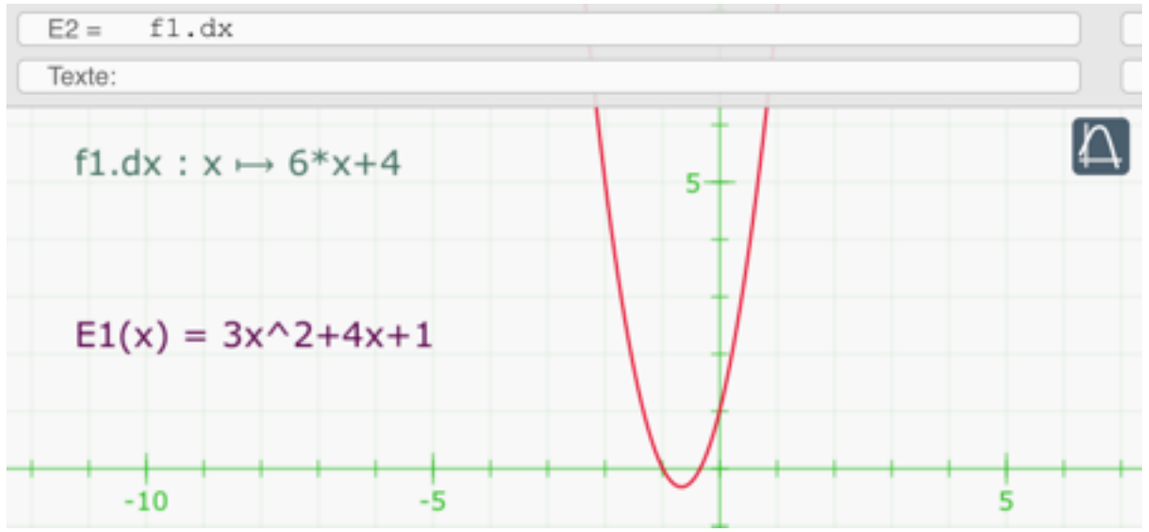
L'engagement direct de ce module peut être un vrai outil didactique en classe. En effet, son icône de tracé n'apparaît que si l'écriture a une syntaxe correcte comme ci-après.

On note l'écriture fonctionnelle de l'expression E1, avec E1(x). Quand on choisi l'icône, le tracé est effectué, l'expression disparaît mais elle n'est pas perdue, elle est incorporée au tracé graphique qui devient un objet géométrique. On peut alors mettre des bornes au tracé graphique.

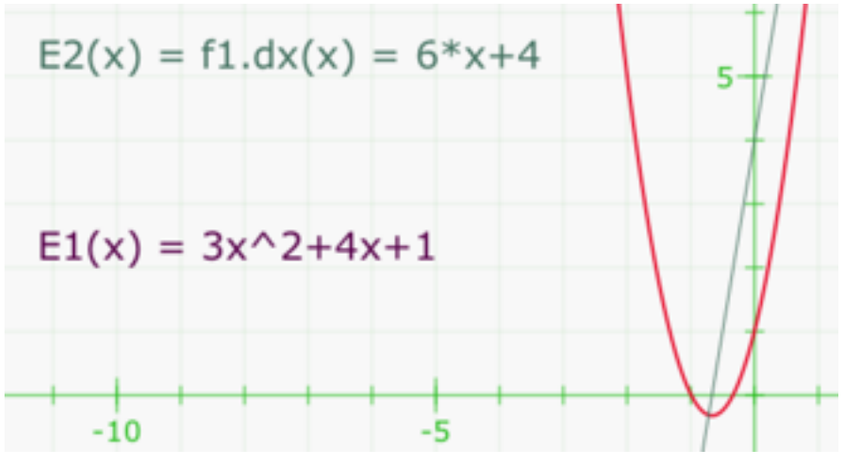
A propos du clavier de DGPad : sur ordinateur, on peut tout taper au clavier sauf la puissance : l'item **^ doit** être tapé depuis le clavier de DGPad.

Bien entendu, on peut incorporer toutes les autres variables numériques de la figure dans une expression fonctionnelle, dont des curseurs.

Sans entrer dans le détail (ce n'est pas l'objectif ici) signalons que plusieurs outils sont disponibles, par exemple l'outil **d** (au dessus de π) correspond à la dérivation.



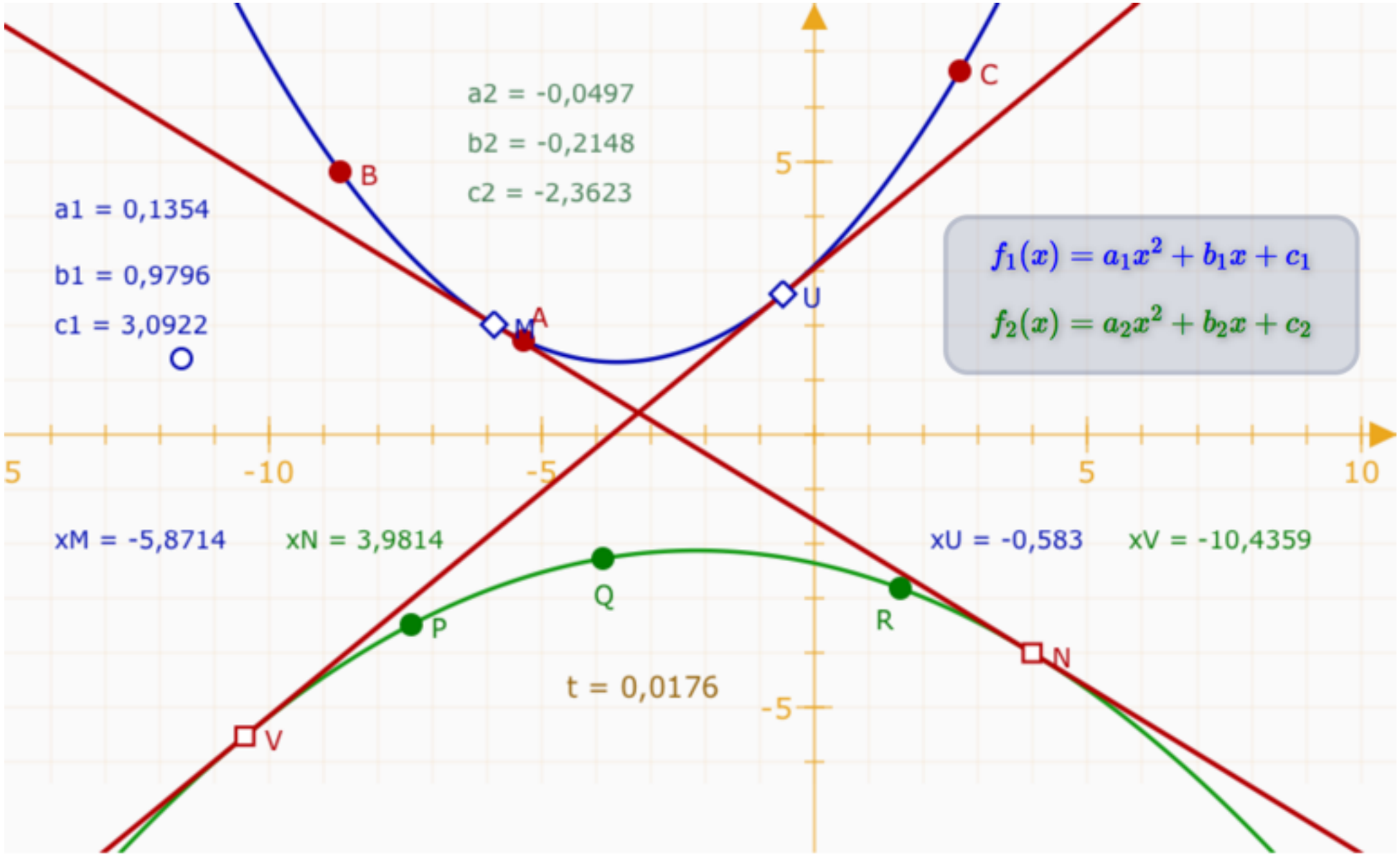
Et en validant, par l'icône graphique, l'expression E2 devient ce que l'on voit ci-contre.



Le choix de l'écriture **dx** vient de ce que l'on peut travailler sur plusieurs variables, mais nous ne développerons pas plus avant ici.

En géométrie dynamique, on met l'accent sur la manipulation directe. Ici on peut avoir envie d'agir directement sur les points. Ci-après deux figures sur ce thème. Tout d'abord, les tangentes communes à deux paraboles définies chacune par 3 points. Ensuite, à quelle condition ces deux tangentes peuvent-elles être confondues ?

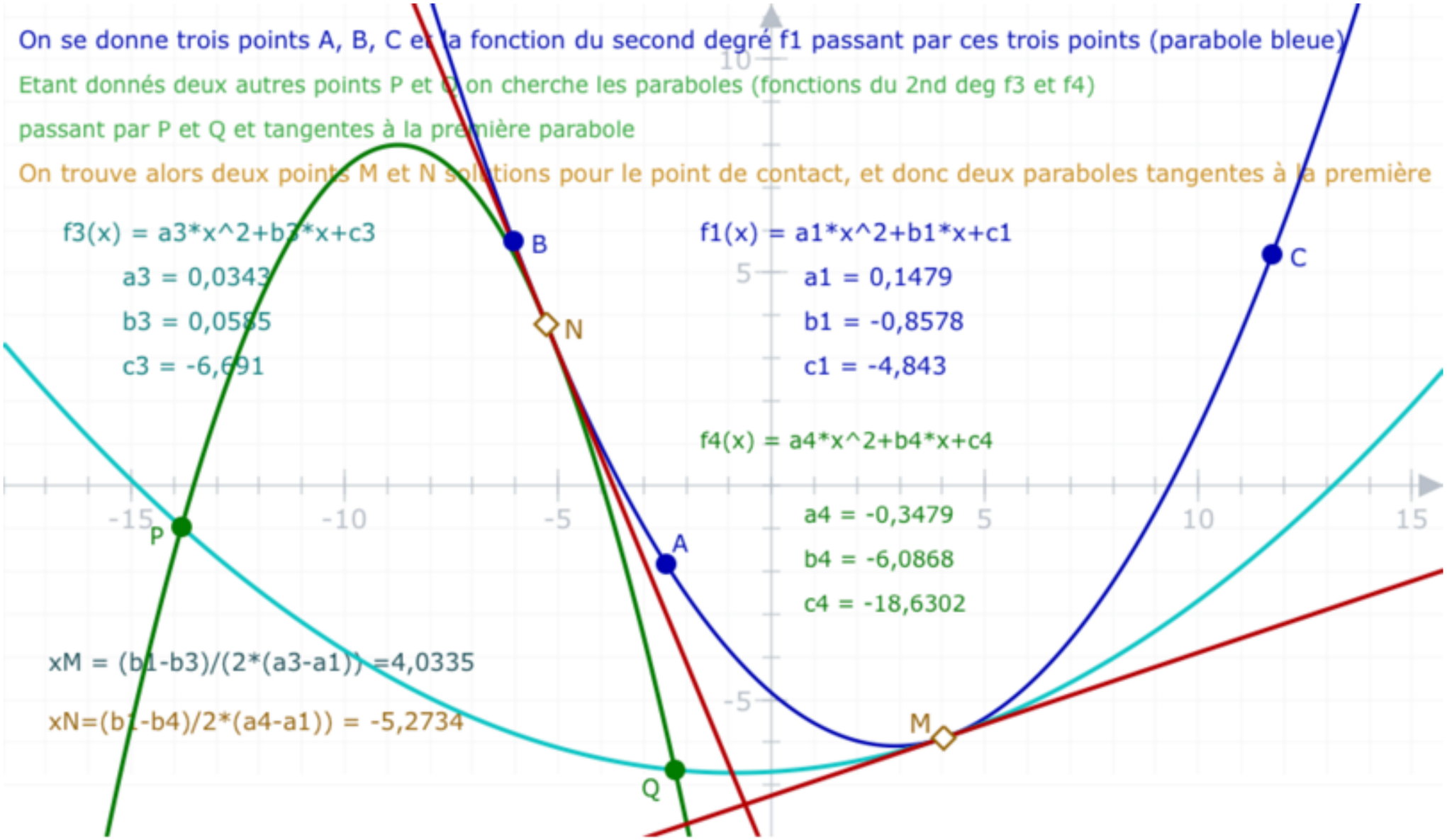
Figure dynamique 2.7 – Tangentes communes à deux paraboles définies par 3 points.



Les tangentes communes aux deux paraboles existent si et seulement si $t = a_1a_2((b_2 - b_1)^2 - 4(a_2 - a_1)(c_2 - c_1)$ *est positif.*

Dans la figure suivante on s'intéresse au cas où les deux droites rouges sont confondues : les paraboles sont tangentes. Dans ce cas un des trois points P, Q ou R disparait, R disons (remplacé par le point de contact cherché).

Figure dynamique 2.8 – Quelles sont les paraboles passant par P et Q tangentes à la parabole passant par A, B, C ?



Cette figure illustre que les problématiques de manipulation directe posent de nouvelles questions intéressantes à résoudre.

Utilisation des macros

La qualité d'un logiciel de géométrie dynamique peut s'évaluer sur plusieurs critères, comme la non modalité, la capacité à engager la création d'objets à l'intérieur de la création d'autres (engagement direct), et la qualité des micro-mondes qu'il permet de construire. Les premières réflexions et l'exigence de qualité de l'expérience utilisateur sur ces sujets ont été initiées par Jean-Marie Laborde, auteur de Cabri-géomètre dont l'auteur du présent ouvrage a été un étudiant, et celui de DGPad et CaRMetal très proche. Les qualités de micro mondes peuvent être très subtiles et CaRMetal a poussé loin cette qualité (comme l'accessibilité des objets constituant des macros dans d'autres macros par exemple). Même si DGPad est en deça de CaRMetal, pour des raisons techniques, les micro-mondes créés par DGPad restent d'une grande efficacité, même s'ils sont moins aisément accessibles qu'avec CaRMetal.

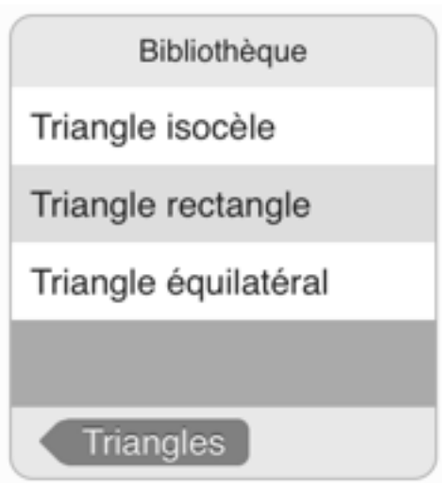
DGPad dispose de deux types de macros : celles, dites « de bibliothèque », intégrées dans le logiciel et celles dites « personnelles» créées par l'utilisateur.

Sur ce point, DGPad a encore innové dans la réalisation de ces macros - nous ne développerons pas plus avant car ce n'est pas nécessaire pour l'utilisation de Blockly – mais c'est aussi sur la gestion génériques de ces macros personnelles que DGPad n'a pas encore été assez développé.

Les macros de bibliothèque

Elles sont regroupées par thème dans des dossiers dont on voit la gestion de contenu ci-dessous :

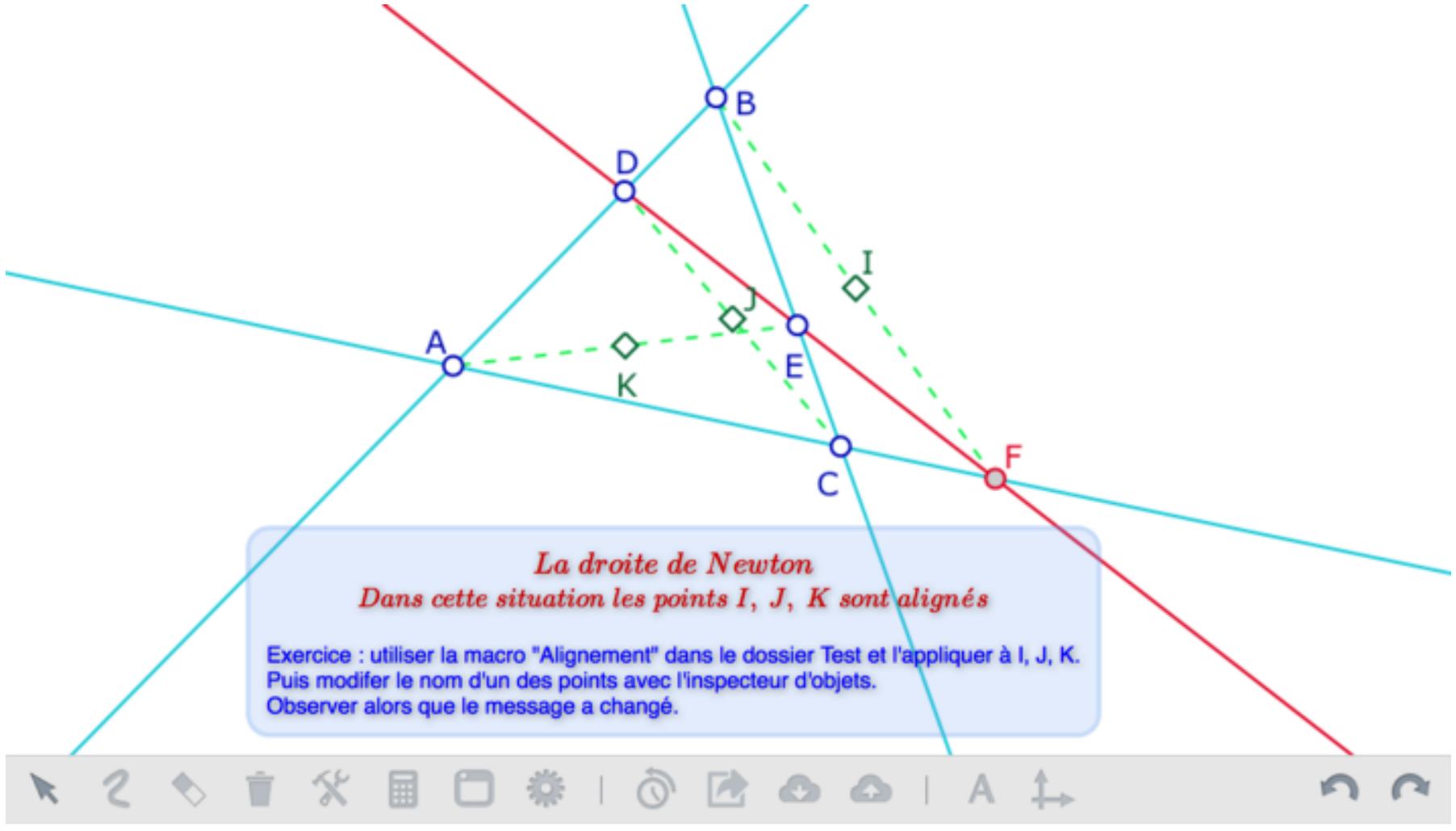
Polygones/Triangles/3 types



Dans les trois pages suivantes vous mettrez en œuvre certaines des possibilités des macros de DGPad.

Macros de bibliothèque - Dans cet exercice de manipulation, il s'agit d'illustrer que les macros de DGPad – parce que le logiciel est écrit en JavaScript – peuvent être dynamiques même sur les commentaires utilisant les noms des objets.

Figure dynamique 2.9 - Propriété des diagonales du quadrilatère complet.

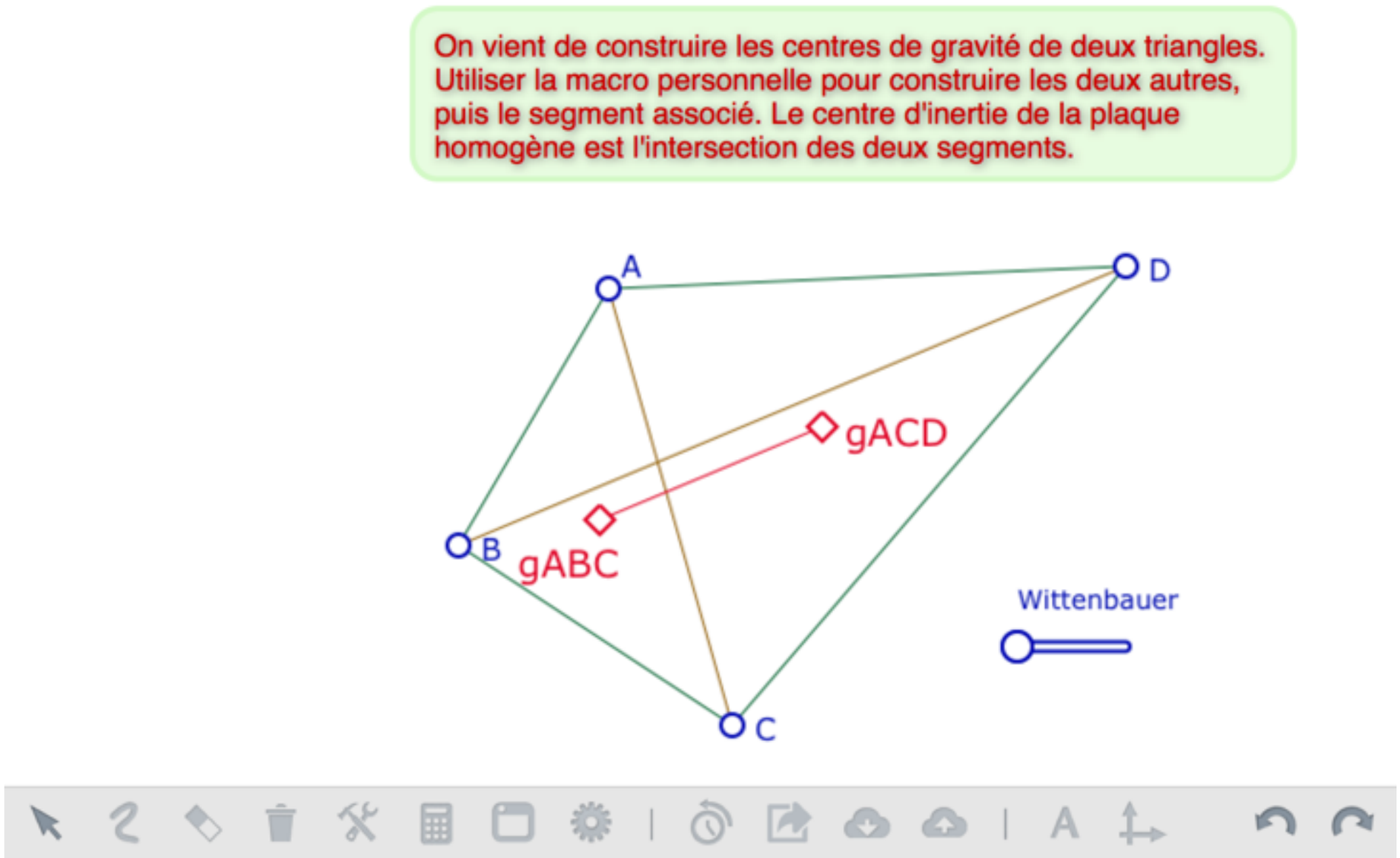


1. Activer les macros (icône marteau/clé à molette) - 2. Ensuite activer l'inspecteur (icône roue).

Mode standard ou consultation : ne pas oublier qu'en désélectionnant l'icône des macros, vous passez en mode consultation. Selon ce que vous voulez faire, il faut à nouveau activer le mode standard (sélectionner le pointeur ou l'inspecteur d'objets).

Utilisation d'une macro personnelle : il s'agit d'achever la figure pour illustrer que les macros personnelles peuvent elles aussi être réactives aux noms des objets utilisés, car les points créés vont être indicés des noms des points utilisés. Comme l'isobarycentre g_{MNP} des 3 points M, N et P est aussi le centre de gravité du triangle MNP en tant que plaque homogène, le centre de gravité de la plaque ABCD est à l'intersection des segments $[g_{ABC}g_{ACD}]$ et $[g_{ABD}g_{BCD}]$.
C'est aussi le centre du parallélogramme construit sur les tiers des segments (Wittenbauer).

Figure dynamique 2.10 – Centre d'inertie d'une plaque homogène - Théorème de Wittenbauer.

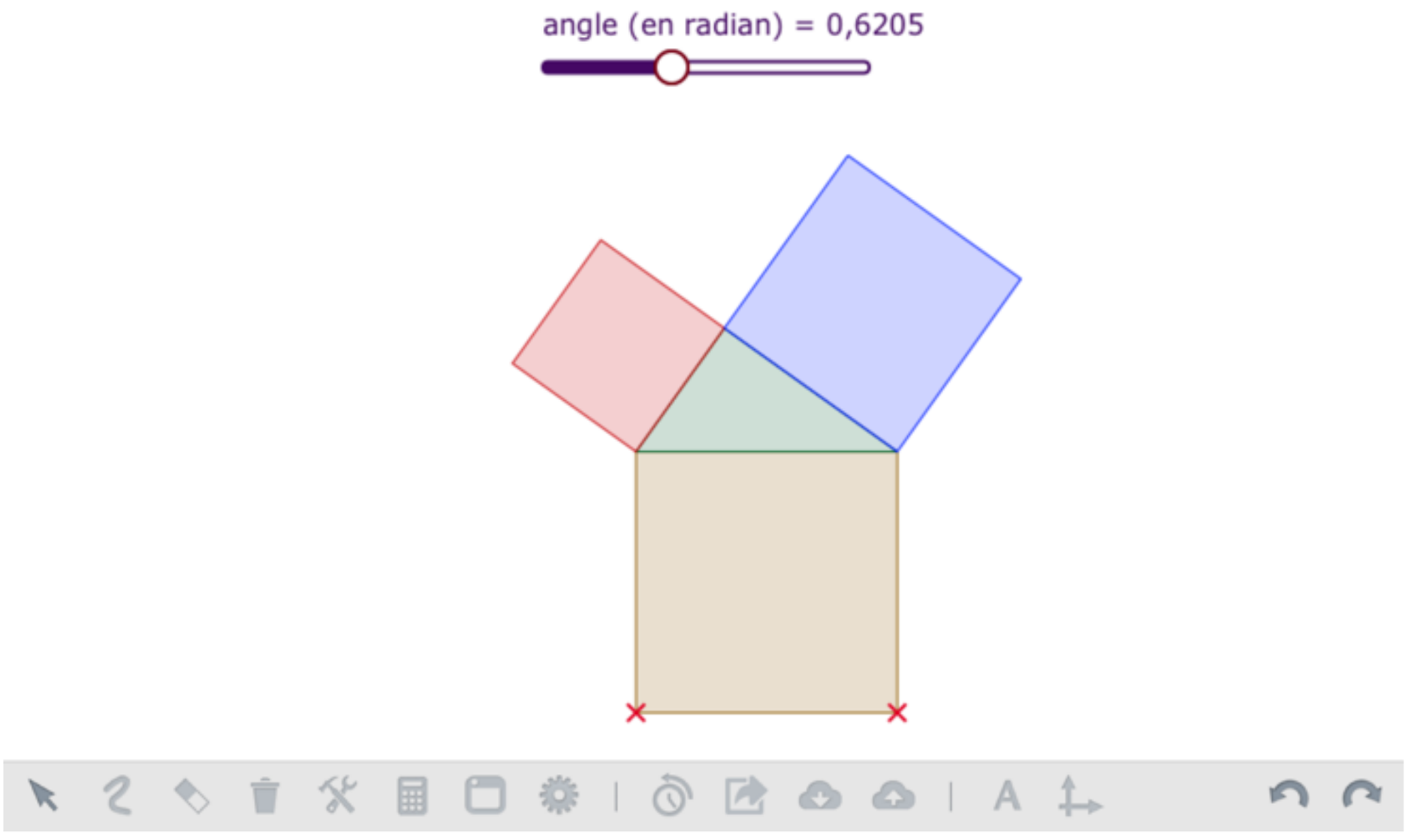


Construire les centres de gravité des triangles ABD et BCD.

Autre exemple d'utilisation d'une macro personnelle

On notera que dans cette macro, la donnée de l'angle est une donnée implicite, il n'y a pas à la montrer.
Penser à modifier l'angle pendant les applications successives de la macro.

Figure dynamique 2.11 – Fractale de Pythagore – Macro en un touché



Activer la macro personnelle et l'appliquer itérativement aux carrés rouges et bleus sur les côtés des triangles rectangles.

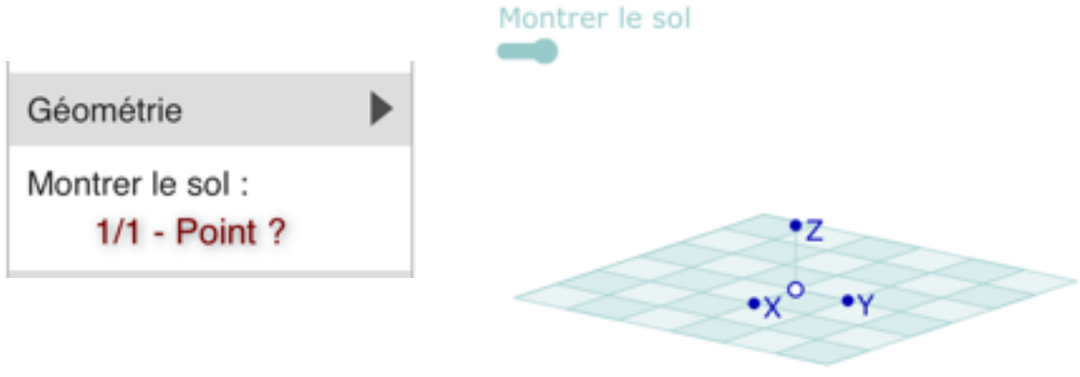
Mise en place de l'environnement 3D

Dans le cadre d'une programmation de la tortue ou de Blockly en général, c'est peut-être le seul moment où vous aurez à utiliser l'interface des macros car l'environnement 3D s'installe par macro. C'est le premier dossier des macros.

DGPad n'est pas un logiciel de géométrie 3D au sens usuel. Il permet des constructions 3D dynamiques, mais sans gestion automatique des parties cachées, c'est plutôt un logiciel de géométrie du point en 3D.
En revanche, il a des spécificités propres : plusieurs des outils de la palette des points fonctionnent en 3D, les droites et les segments, mais aussi le «cercle passant par 3 points». D'autres outils sont dans le dossier **Géométrie** que l'on voit ci-dessus. Mais ils ne serviront pas dans le cadre de Blockly.

Pour installer l'environnement 3D, on commence par prendre un repère 3D en désignant un point comme origine du repère.

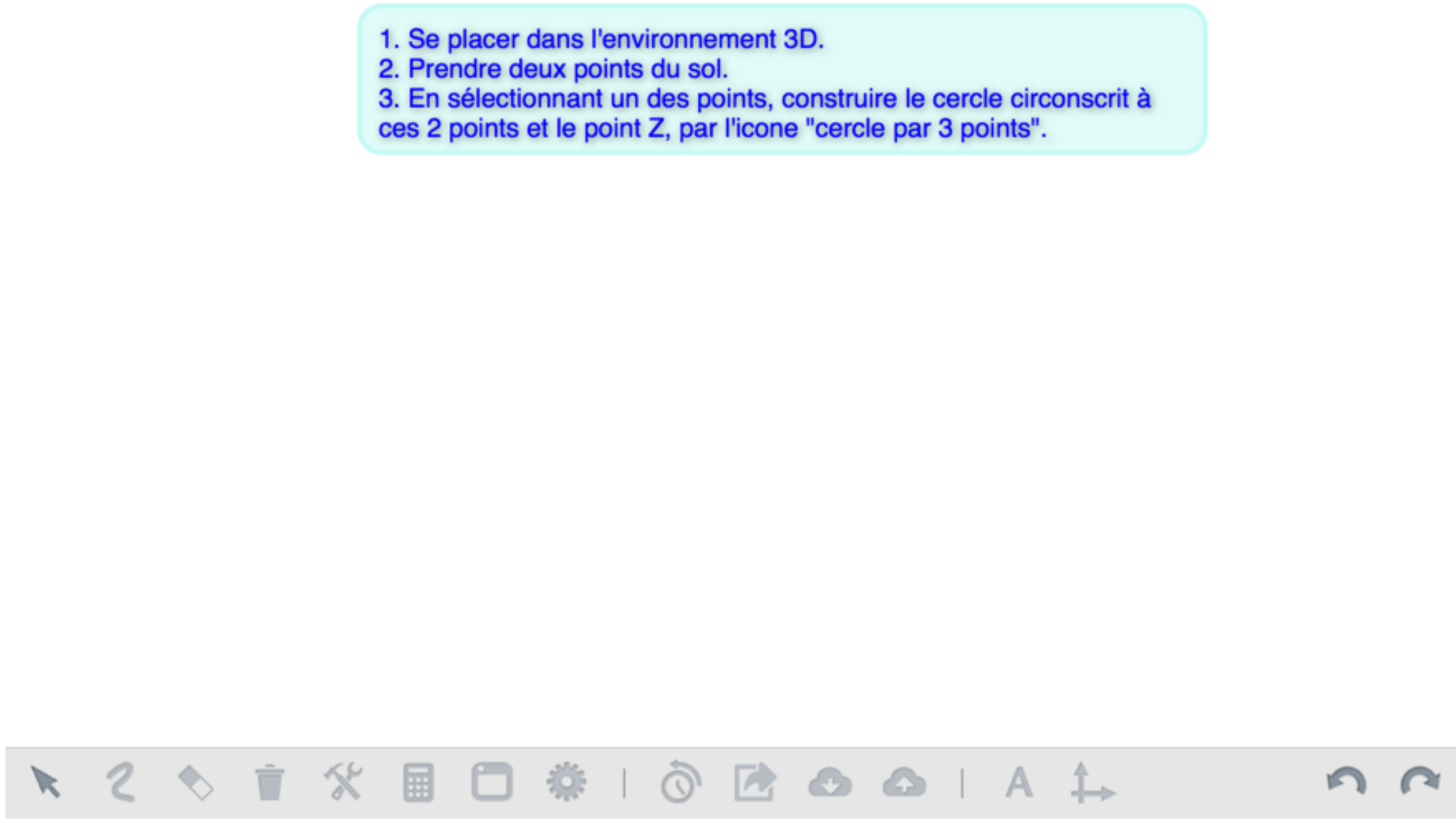
Puis on choisit (mais pas toujours nécessairement) l'outil **Montrer le sol** qui permet de placer facilement des points dans le plan (xOy). Montrer l'origine du repère.



La page suivante vous propose d'installer l'environnement 3D et de placer deux points sur le sol. Pour utiliser Blockly, ce sera suffisant. Les pages suivantes sont des illustrations des possibilités du logiciel.

Prendre des points sur le sol : toucher le sol (ses bords sont surlignés). Prendre l'icône du **point** déjà rencontrée précédemment. Un point sur le contour apparait. Lâcher le doigt – ou la souris – **à l'intérieur du sol**, à l'endroit où vous voulez créer le point.

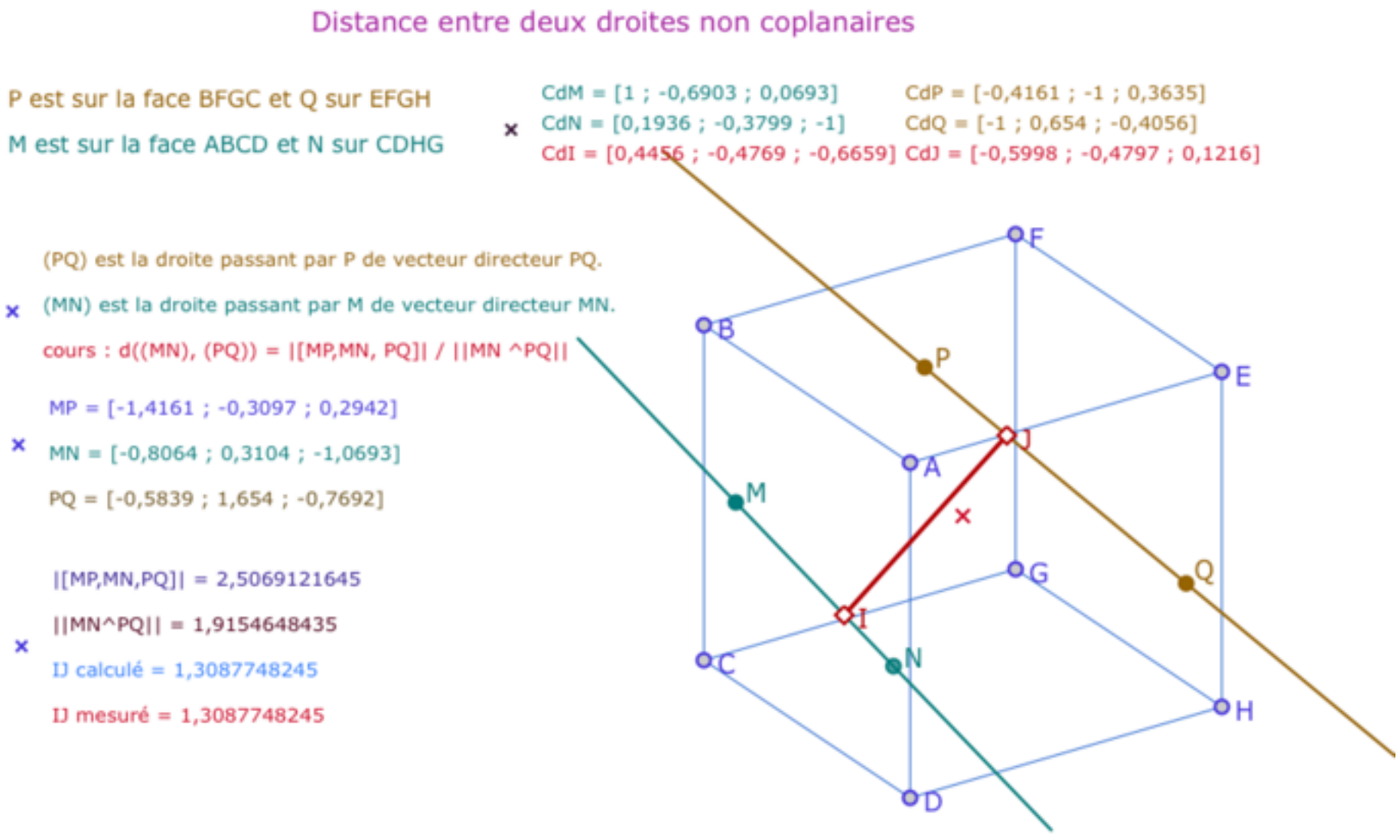
Figure dynamique 2.12 – Installation de l'environnement 3D



... et construction d'un premier cercle circonscrit 3D...

L'outil distance 3D : la fonction **d(,)** du clavier DGPad s'adapte à l'environnement 2D ou 3D. Ci-dessous, la distance IJ entre les droites est calculée par les formules, puis mesurée par le logiciel (qui traite les données autrement, par vibration du trièdre).

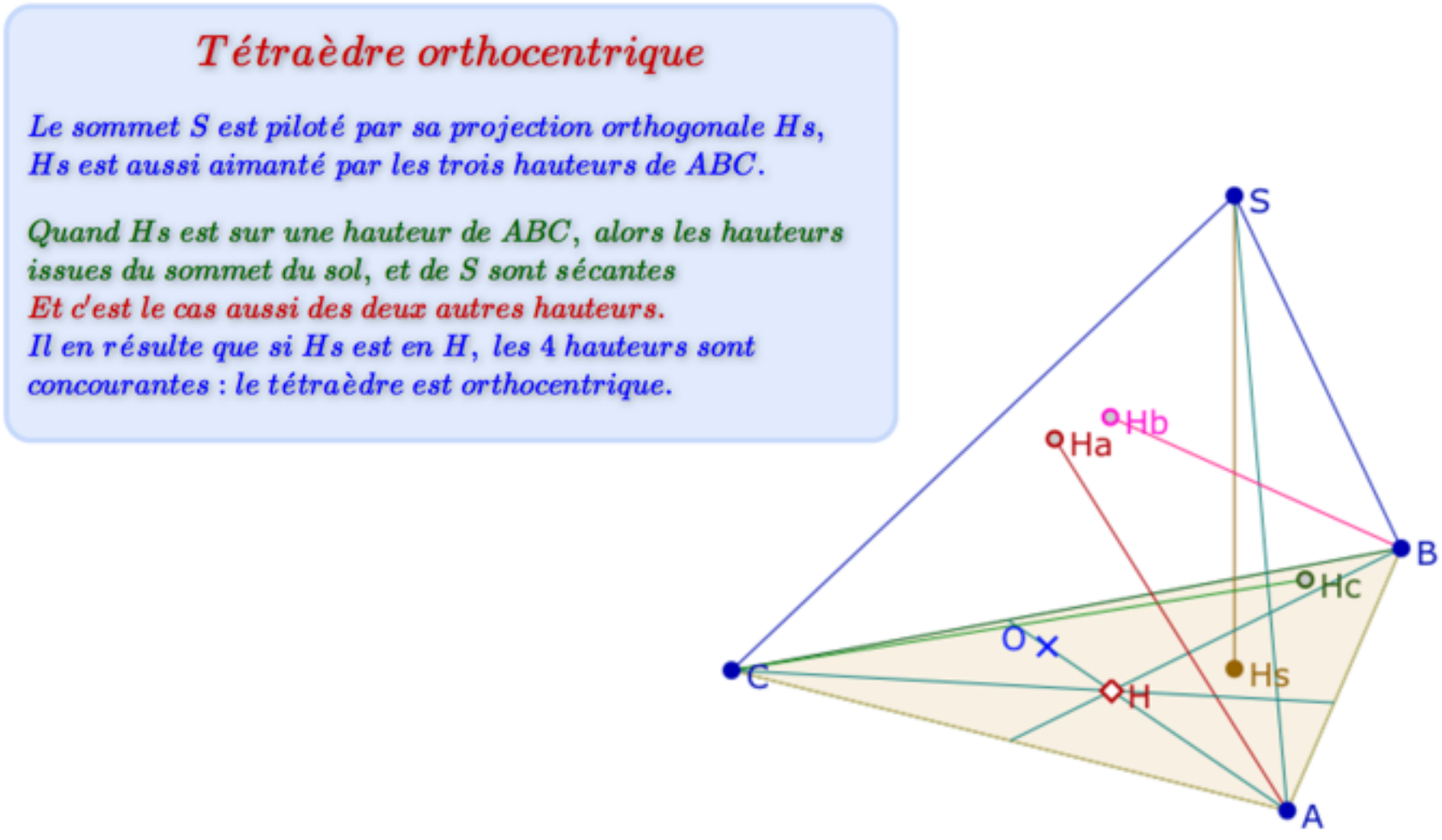
Figure dynamique 2.13 - Perpendiculaire commune et distance entre deux droites non coplanaires



Tourner le trièdre en faisant glisser un doigt sur l'écran. Déplacer les 4 points dans leurs faces respectives.

Utilisation didactique de l'aimantation en 3D (aide visuelle à la compréhension d'une preuve)

Figure dynamique 2.14 - Un tétraèdre est orthocentrique *ssi* un sommet se projète sur l'orthocentre de la face opposée



Placer Hs sur une des hauteurs, tourner la figure (au doigt) pour voir les 4 hauteurs deux à deux sécantes.
Tester sur une autre hauteur. Puis placer Hs sur H. On déplace la figure par l'origine du repère O.

Introduction à la tortue de DGPad

La tortue de DGPad est une partie de l'implémentation de Blockly qui, parce qu'elle est placée dans un environnement dynamique, a la particularité de réagir en temps réel. C'est un choix de l'auteur du logiciel, Erick Hakenholz, que de conformer la trace de la tortue instantanément à toutes les modifications de la figure. Cela en fait **un outil inédit**, d'une performance surprenante, avec des possibilités d'utilisation tout à fait originales et encore à explorer. Ces possibilités sont disponibles en maintenant une programmation aussi aisée que dans les autres environnement de programmation par bloc.

La tortue dynamique de DGPad rend la programmation 3D dynamique d'une telle aisance qu'on pourrait repenser les curricula du cycle 4 du collège sur l'enseignement de l'espace avec un tel outil.

Par contre, dans cet environnement, il n'est pas question de voir la tortue effectuer un trajet ou de la ralentir, il n'y a rien qui ressemble à du kinesthésique. Cela surprend, en général, au début, les utilisateurs d'autres environnements comme Scratch ou GeoTortue par exemple, mais le temps réel est un autre paradigme d'une très grande richesse. Comme tel, il a ses propres méthodologies et développe des démarches spécifiques comme on va le voir sur les exemples.

La tortue est une (petite) **partie de Blockly**, et nous verrons dans un chapitre spécifique les possibilités qu'offre Blockly - comme programmé par Eric Hakenholz - pour la géométrie dynamique. Il permet de donner à DGPad des capacités de rupture programmée du déterminisme des figures que seul CaRMetal possédait déjà, de par la programmation initiale de son noyau CaR par Renée Grothmann.

On l'a déjà dit, mais rappelons une réalité essentielle : Blockly est un **comportement d'objet**, cela signifie qu'il ne crée pas de nouvel objet.

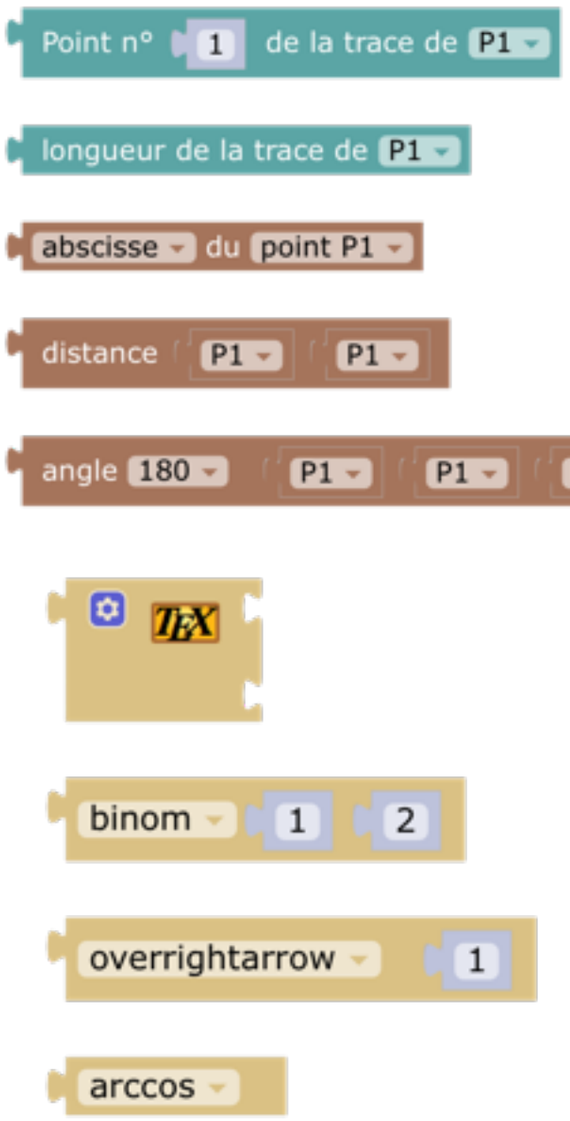
Ce sera flagrant dans l'utilisation standard de Blockly. D'une certaine façon c'est moins clair avec **la trace de la tortue** qui a un statut spécial car bien qu'elle soit un objet de DGPad, par défaut c'est un objet inerte, ce que l'on peut modifier dans l'inspecteur d'objet.

La tortue s'applique essentiellement sur des points. Créer un point, lui appliquer Blockly. L'interface s'ouvre avec 5 onglets, l'onglet **Déplacé** étant activé.

Activer l'onglet **Tortue** rajoute deux rubriques à l'interface de Blockly (Tortue et Texte) et place la tortue sur le point dans la direction (Ox).

Rubriques du Blockly de DGPad - Interface générale

Même si l'interface est bien entendu standard, les blocs mis en œuvre sont liés aux outils de DGPad. Ainsi, **Aspect** reprend les réglages de l'inspecteur d'objet et en ajoute d'autres qui existaient mais n'étaient que programmable. Elle en crée un nouveau, l'aspect **flèche,** pour créer des listes de vecteurs, en particulier des champs de vecteurs pour les systèmes différentiels.



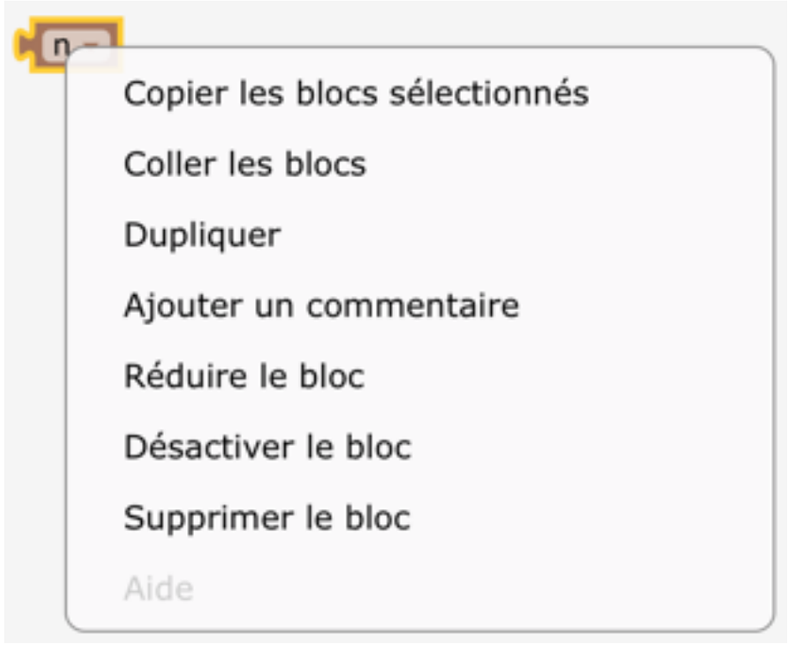
Pour pouvoir être interfacés dynamiquement avec la figure, deux items importants de la tortue, sont disponibles dans la rubrique **Expressions**, pour des utilisations plus techniques et, en général, non scolaires.
Classiquement dans la programmation par blocs, de nombreuses options de bloc sont dans des **menus déroulants** comme ici :

La rubrique **Textes**, qui n'existe que pour la Tortue, permet d'écrire aussi sur la trace de la tortue en LaTeX. La version retenue est le **KA-TeX**, de la Khan Academy. Les items à un paramètre sont pointés sur ***overrightarrow*** qui n'existait pas dans le KA-TeX, et que Eric a écrit pour la tortue. C'est aussi une façon d'écrire du **TeX** partout dans la figure.

Les manipulations de base des outils (généralité Blockly)

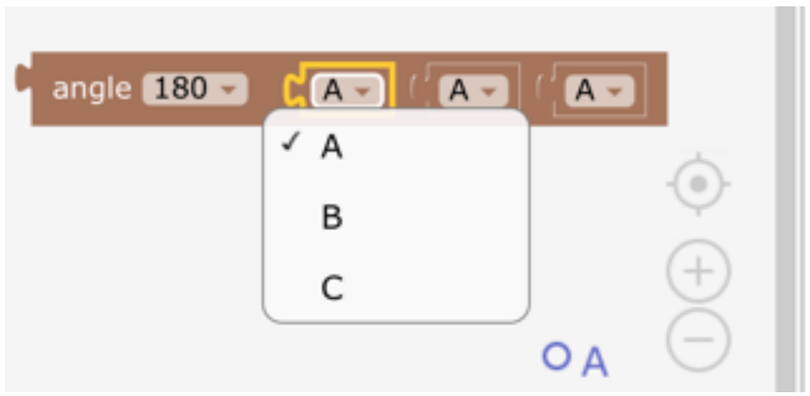
Un **touché long** (un clic droit à la souris sur ordinateur) sur un bloc permet de nombreuses opérations, dont le **dupliquer**, très pratique dans l'espace de travail.

Quand on copie un bloc, il copie toutes les lignes suivantes, ce qui permet de copier et coller toute une partie du code. Il faut donc séparer les blocs pour ne copier qu'une partie du code.



Le paramétrage des blocs par manipulation directe de la figure

Une spécificité des blocs de DGPad est dans l'interaction dans les deux sens entre le code et la figure : on peut paramétrer le code depuis la figure.
Dans l'exemple suivant, on peut choisir les points par un pop-up ou en **sélectionnant les points sur la figure** (il faut avoir sélectionné la part du bloc concerné).



○ C

○ A

○ B

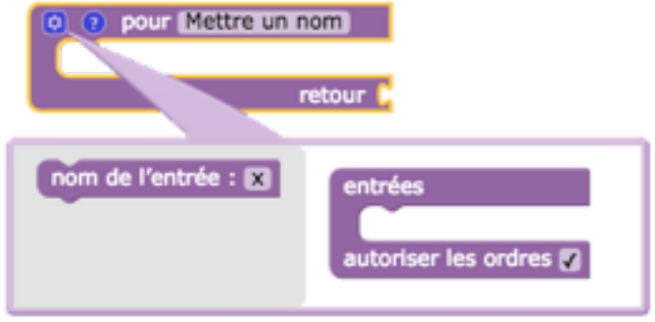
○ B

Ci-dessus, avant de cliquer sur B, ci-dessous, après

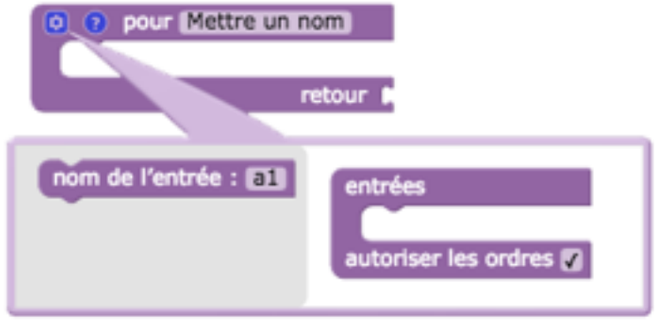
○ B

Mise en œuvre des fonctions

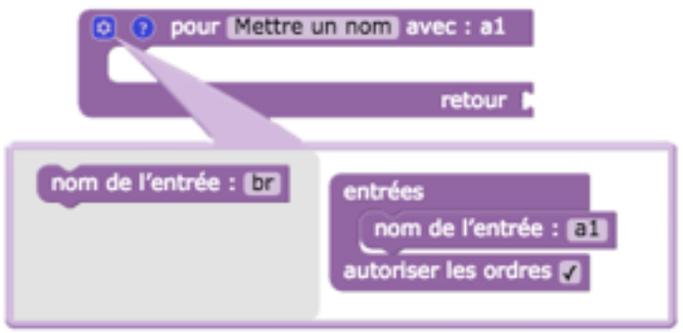
Souvent, les fonctions ne seront utilisées, en classe, que comme des structurations en étapes d'une démarche, en particulier pour communiquer. Mais pour des constructions plus sophistiquées, on utilisera naturellement des fonctions avec paramètres, comme ceci :



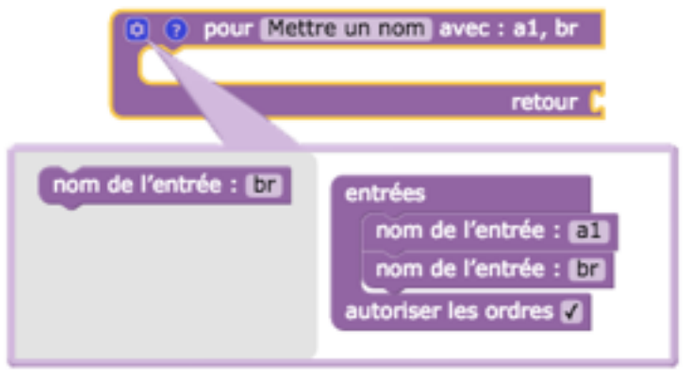
1. Activer le paramétrage



2. Nommer le premier paramètre



3. Glisser le bloc du nom dans les entrées



4. Poursuivre pour d'autres paramètres si nécessaire

Une spécificité du Blockly de DGPad

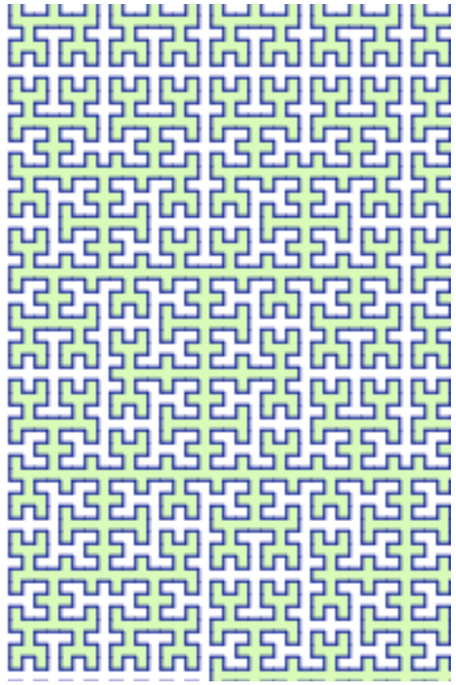
Eric Hakenholz a particulièrement travaillé les variables locales pour que les fonctions récursives puissent figurer dans le retour de leur définition comme dans **pgcdrec** ci-dessous.

En testant sur des sites en ligne en Blockly, vous pouvez vérifier que cela ne fonctionne pas par défaut.

$Nombre\ de\ segments\ =\ 4^n = 262144$

n = 9

taille = 5



Quelle taille maximale
pour la trace de la tortue ?

Ci-dessus en 2D, la courbe de Hilbert.

Ci-contre un arbre récursif classique.

Bien sûr, à des tailles comme celles-ci, les figures ne sont plus dynamiques, mais elles se construisent, on peut faire des copies d'écran.

En bref, on peut obtenir un peu plus de 250 000 segments en 2D et plus de 100 000 segments en 3D.

taille de départ en pixels : 206

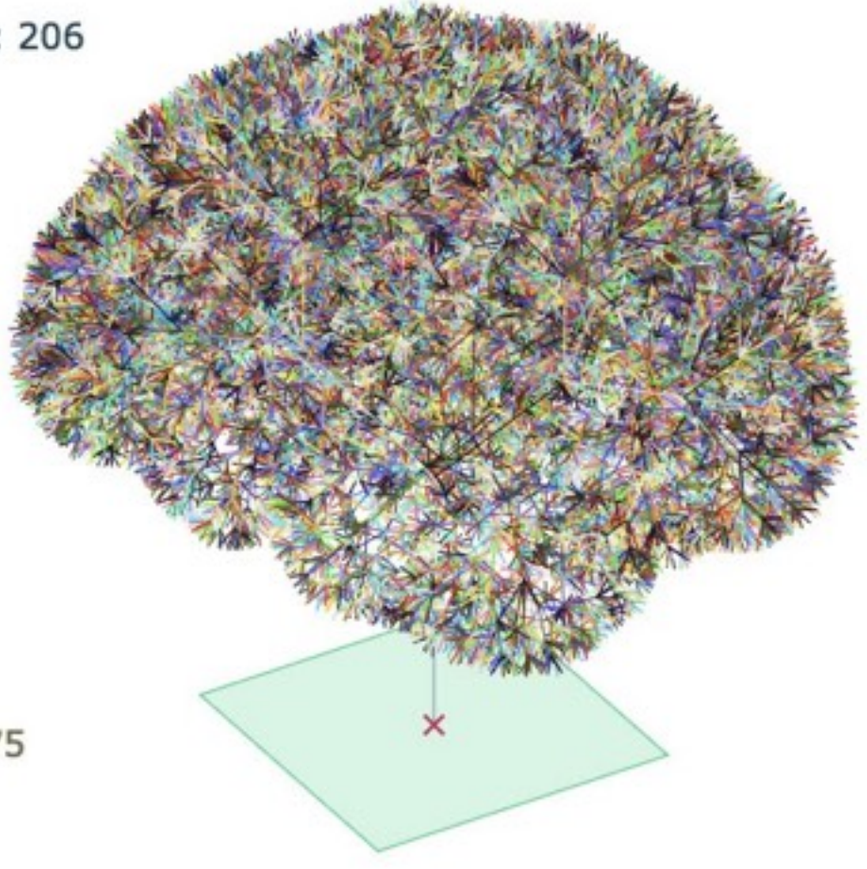
angle 1 : 24

angle 2 : 42,5

angle 3 : 43

coef = 0,61

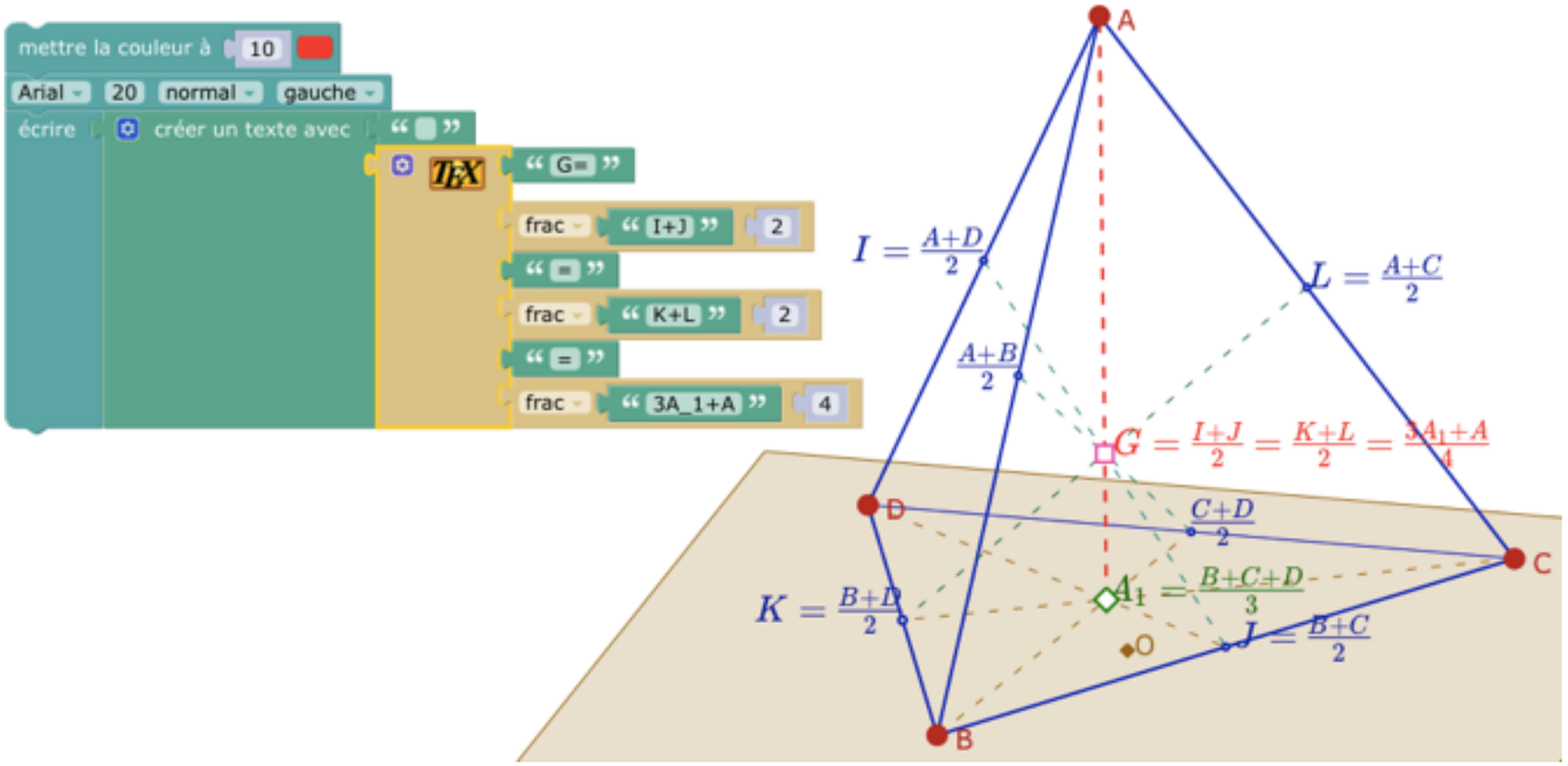
nombre de points : 111975



Le TeX pour un alias de nom

Comme on peut mettre une trace tortue sur chaque point, on peut l'utiliser pour des informations plus riches.

Heureusement que le copier-coller fonctionne bien...



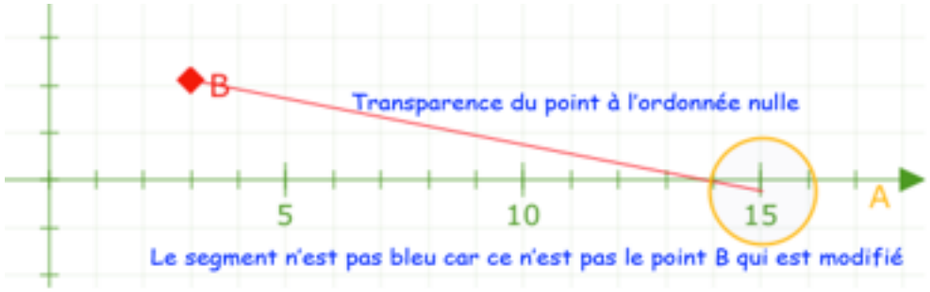
Aspects dynamiques
Les onglets Appuyé - Relaché

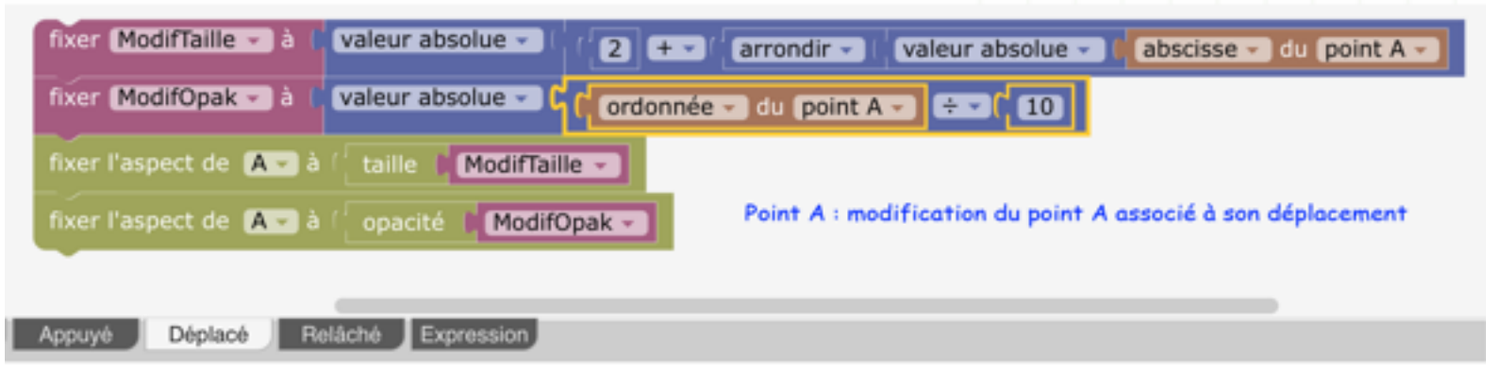
Rapidement quelques exemples d'utilisation des onglets **Appuyé** et **Relaché** pour illustrer la mise en œuvre.

Les comportements dans **Appuyé** et **Relaché** ne sont **pas des «tant que»**, mais ce sont des actions qui sont accomplies. Ainsi si on met le segment à 6 pixels dans le *Appuyé*, cela ne signifie pas qu'il revient à 1 pixel tout seul quand on lâche le point.
Pour parler plus techniquement, **ce n'est pas «on mouse up»**.

Et donc, il faut réinitialiser le segment à 1 pixel dans **Relaché** si on le modifie dans **Appuyé**.

La page suivante permet de tester cela. C'est aussi une page d'exploration pour modifier les codes des points A ou B.





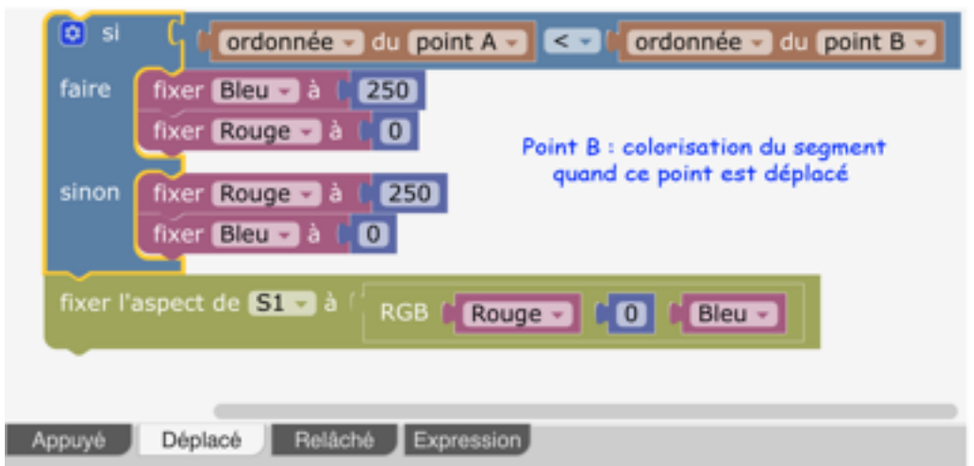


Figure dynamique 3.1 – Tester les aspects – Jouer avec les onglets Appuyé et Relâché sur cet exemple élémentaire

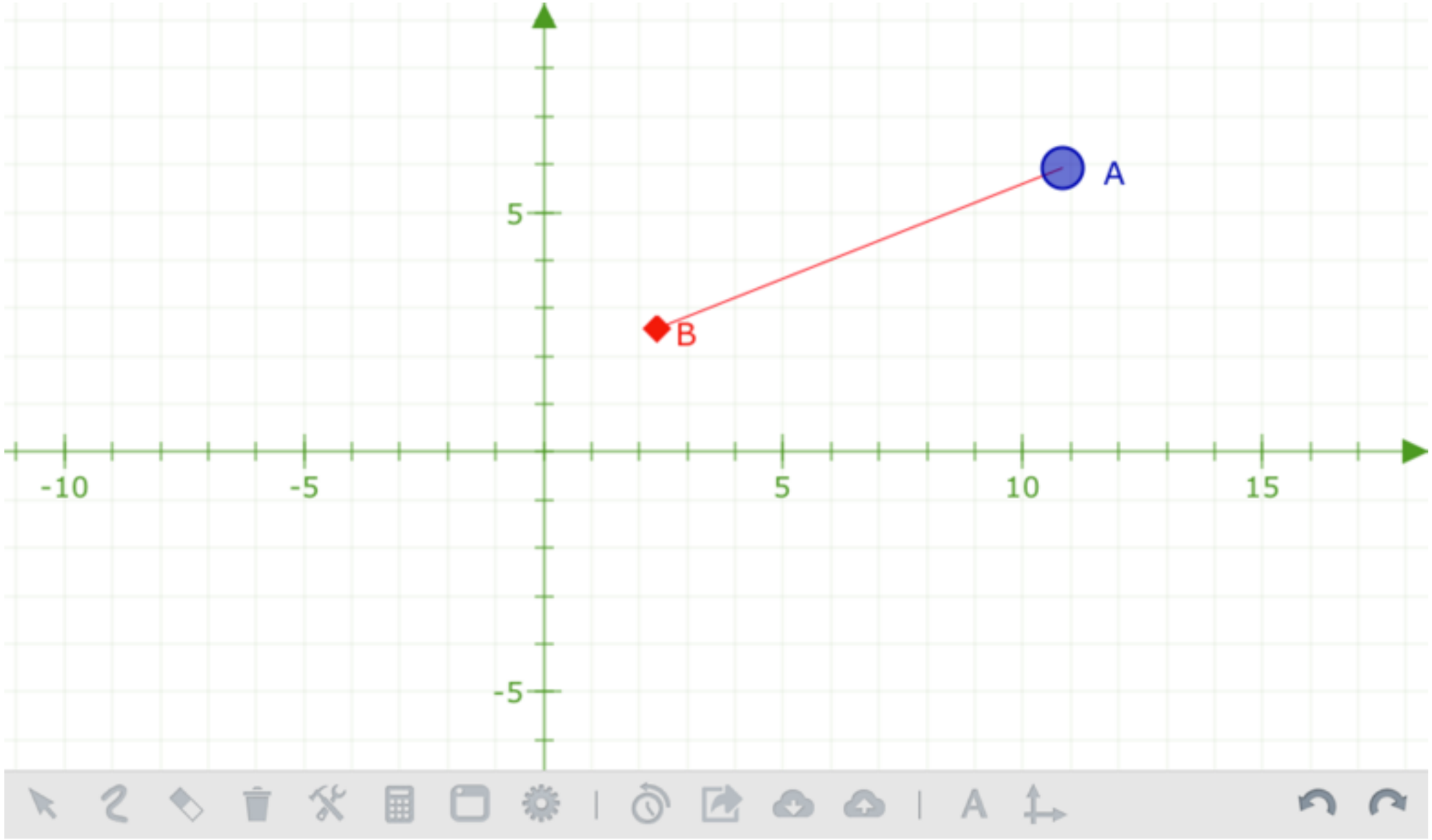
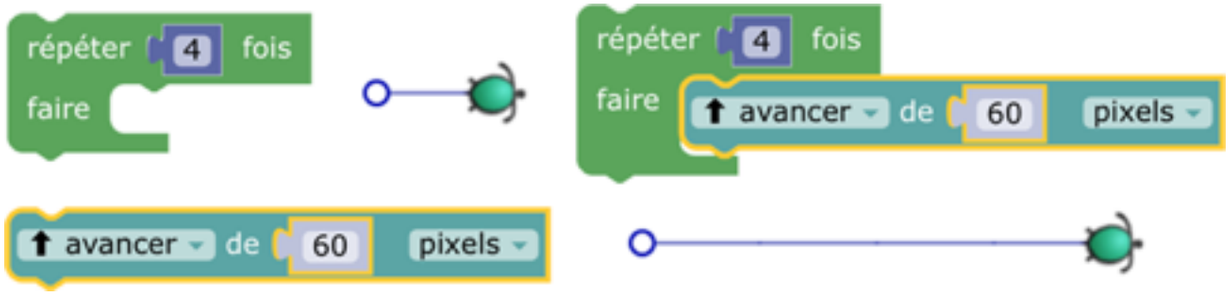
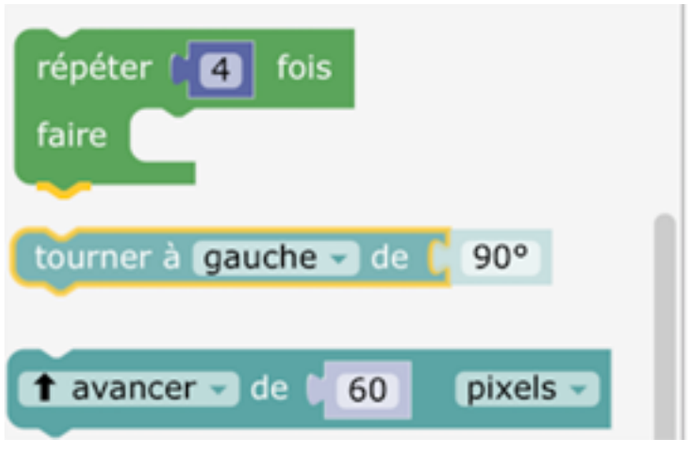


*Figure en **mode consultation** (aucun outil sélectionné) à l'ouverture : déplacer les points A et B pour voir les effets.*
*Il faut passer en **mode standard** (pointeur de gauche sélectionné) pour **accéder au code Blockly des points** et les modifier ou les enrichir.*

Blocs, interface et réactivité de la tortue

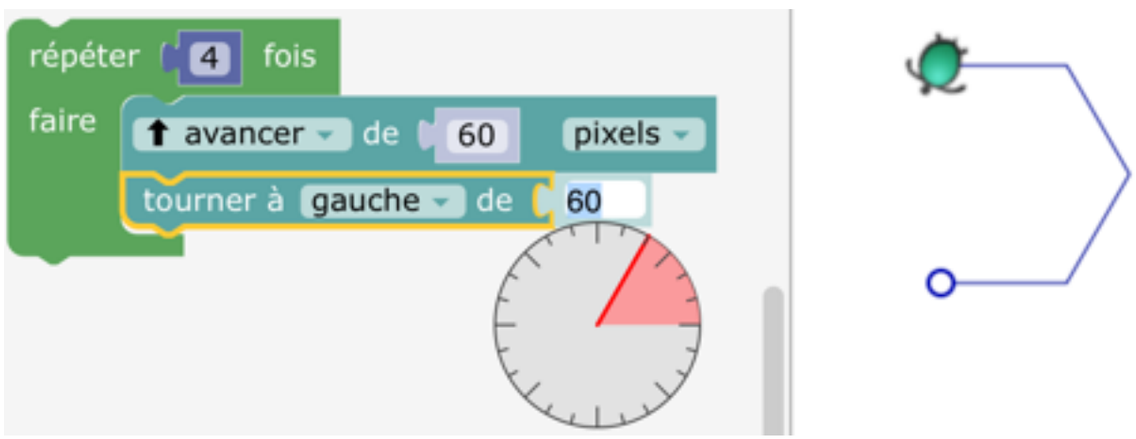
Voyons, sur quelques exemples, ce que l'on entend par **réaction en temps réel de la tortue**. Dans l'illustration de droite, en glissant le bloc **avancer** dans la boucle, la tortue avance instantanément.

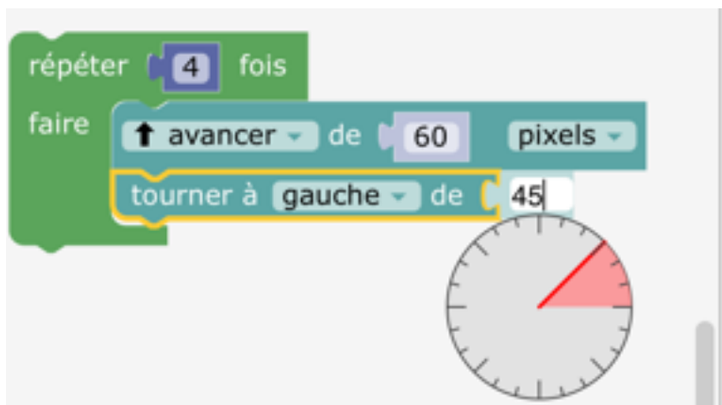


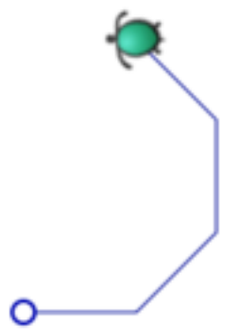


Ci-contre, à gauche, on voit l'interface de Blockly sans montage : on place le point plutôt à droite, en dehors de la page de travail de Blockly qui est de toute façon en transparence. Un bloc se dépose dans l'espace de travail soit en le faisant glisser, soit simplement en le sélectionnant par un touché (un «touch»). Sur l'illustration ci-contre, on voit qu'en glissant le bloc **tourner**, avant même de le déposer, la trace de la tortue se modifie.

Richesse de la réactivité de la tortue : ayant fait le carré usuel, on peut sélectionner l'angle. Il apparaît une roue modifiable, voici ce que donne, sur l'interface réelle, l'action sur la roue, qui a un pas de 15°. On voit bien le potentiel d'exploration « en acte ».



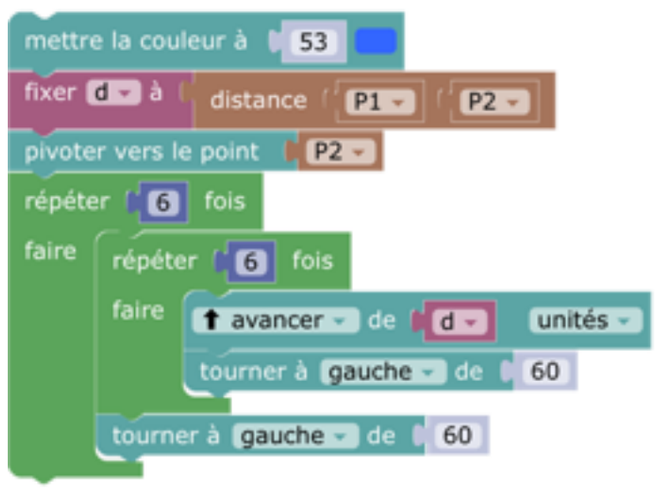


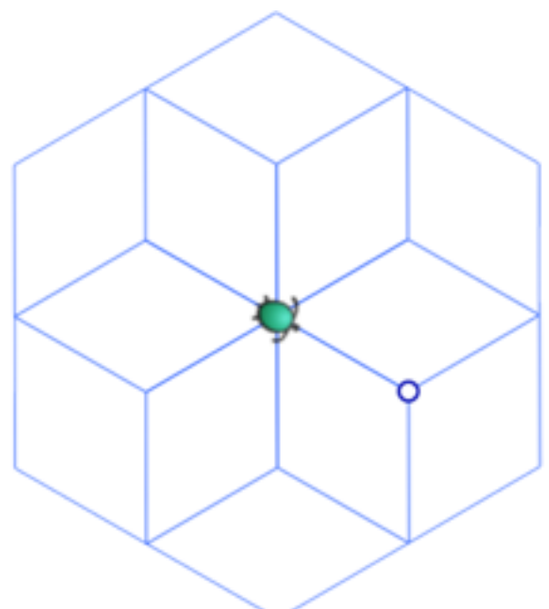


Bien entendu, en faisant cela, les élèves peuvent ensuite modifier le nombre de segments dans la boucle **répéter** pour finir les polygones réguliers. Rapidement se pose la question du rapport entre le nombre de côtés et l'angle dont tourne la tortue.

Première activité proposée dans la page suivante : passage du numérique à l'algébrique.

Dans cette activité, on a commencé par construire 6 hexagones en rotation de 60° (à droite). Pour bien voir ce qu'il se passe et questionner les élèves, on ajoute un remplissage à 20% de chaque hexagone, et on demande aux élèves d'expliquer les nuances de couleurs : pourquoi des parties sont-elles plus foncées ?

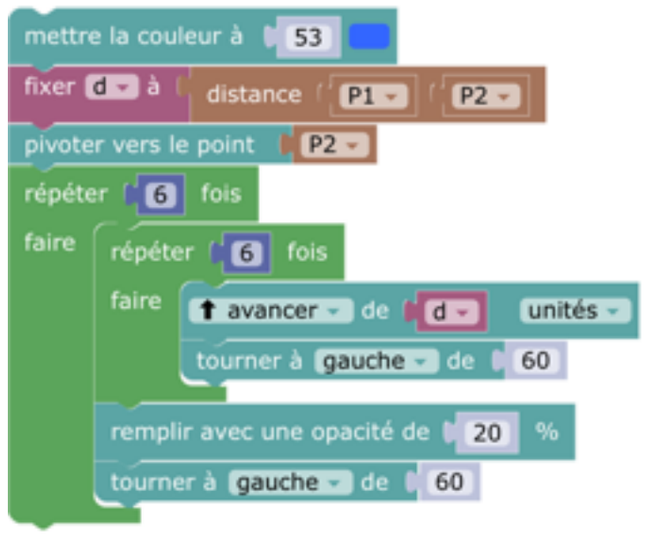


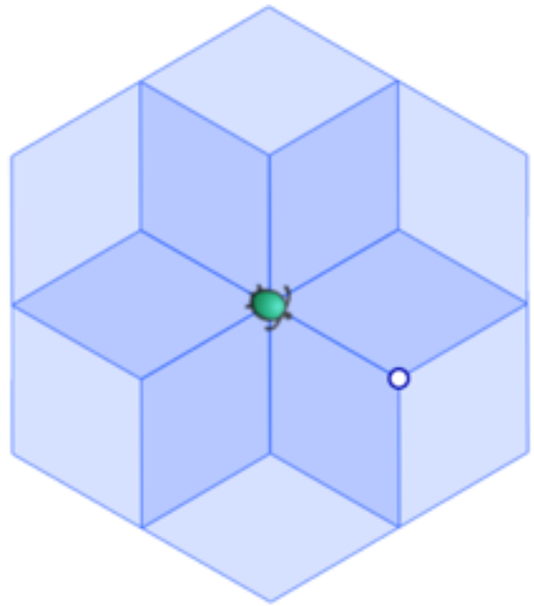


Modification 1 : dans la page suivante, la figure est faite, et il y a un curseur **n**. L'idée est de remplacer l'hexagone par un polygone à **n** côtés. Pour cela, activer Blockly, le code est dans le point central.

Puis sélectionner dans **Expressions** le bloc des objets, et remplacer **P1** par **n**.

Penser ensuite à dupliquer deux fois (appui long) le bloc **n** puisqu'il en faut deux pour les limites des boucles et un autre pour préparer la rotation de la tortue, expression que l'on dupliquera également puisqu'on l'utilise aussi deux fois.





Pour n= 8, par exemple, on peut à nouveau demander aux élèves pourquoi il y a 3 niveaux de couleurs.

Modification 2 : on peut ensuite doubler le nombre d'itération de la *boucle extérieure* et diviser par deux l'angle de la rotation qui passe d'un polygone à l'autre. Une autre modification est proposée plus loin.

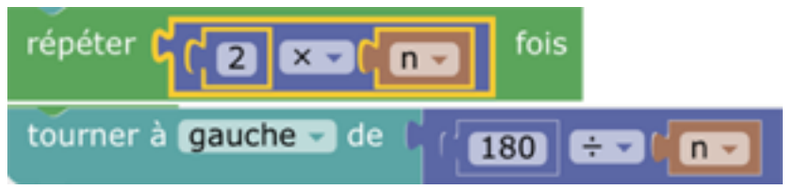
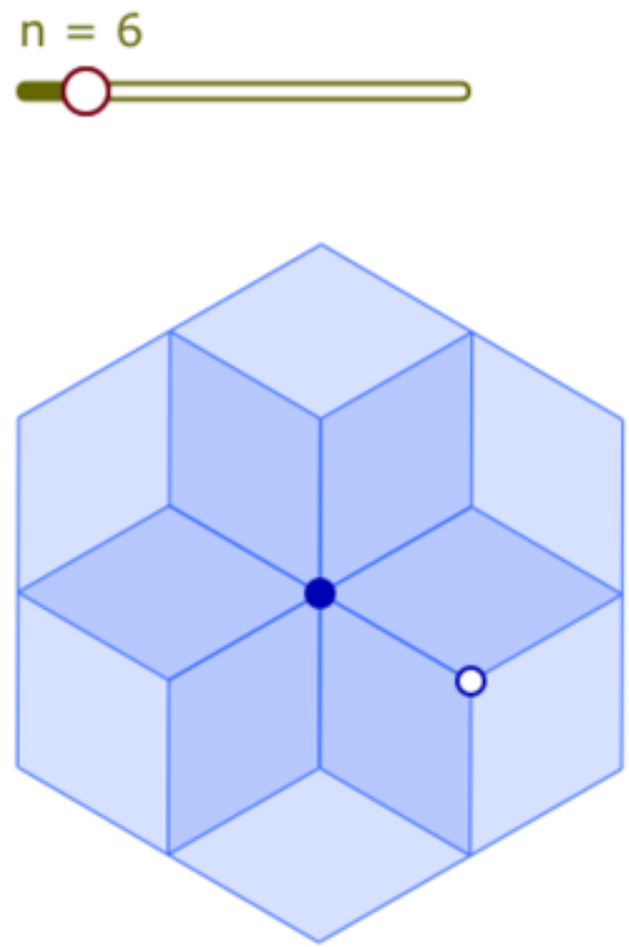


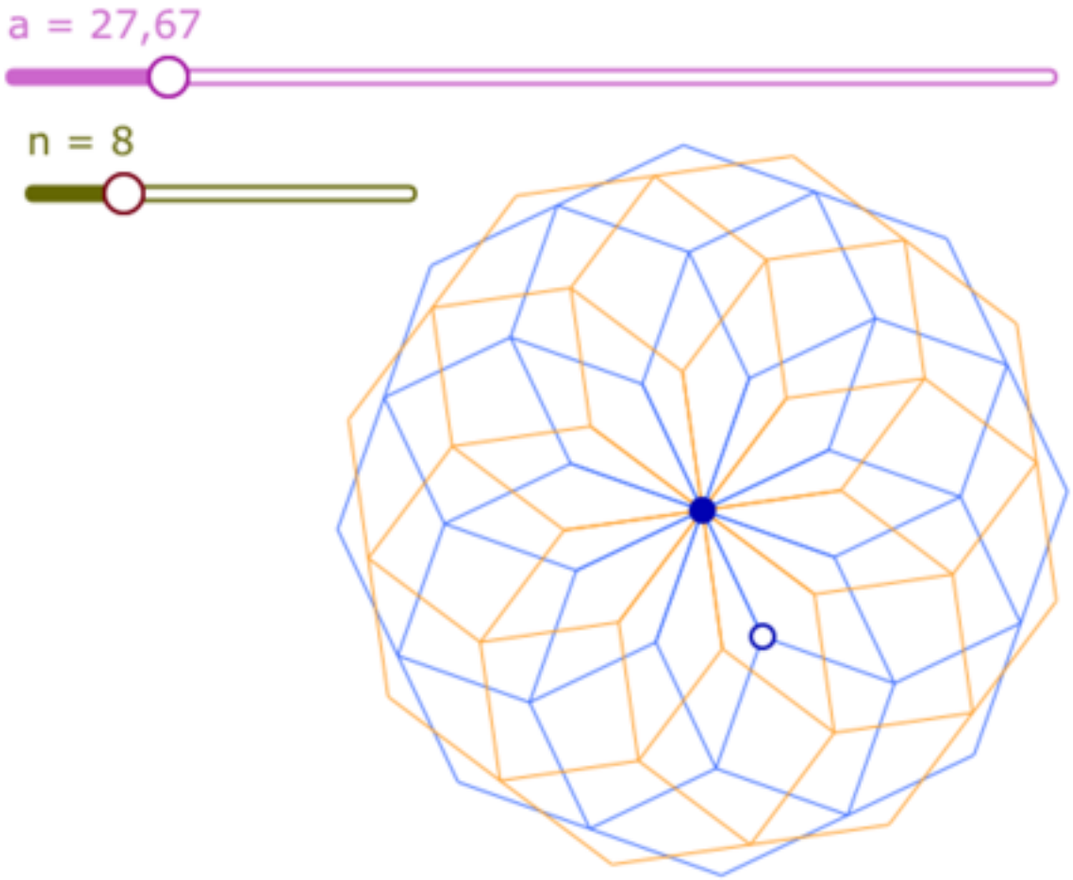
Figure dynamique 3.2 – Première activité sur la tortue



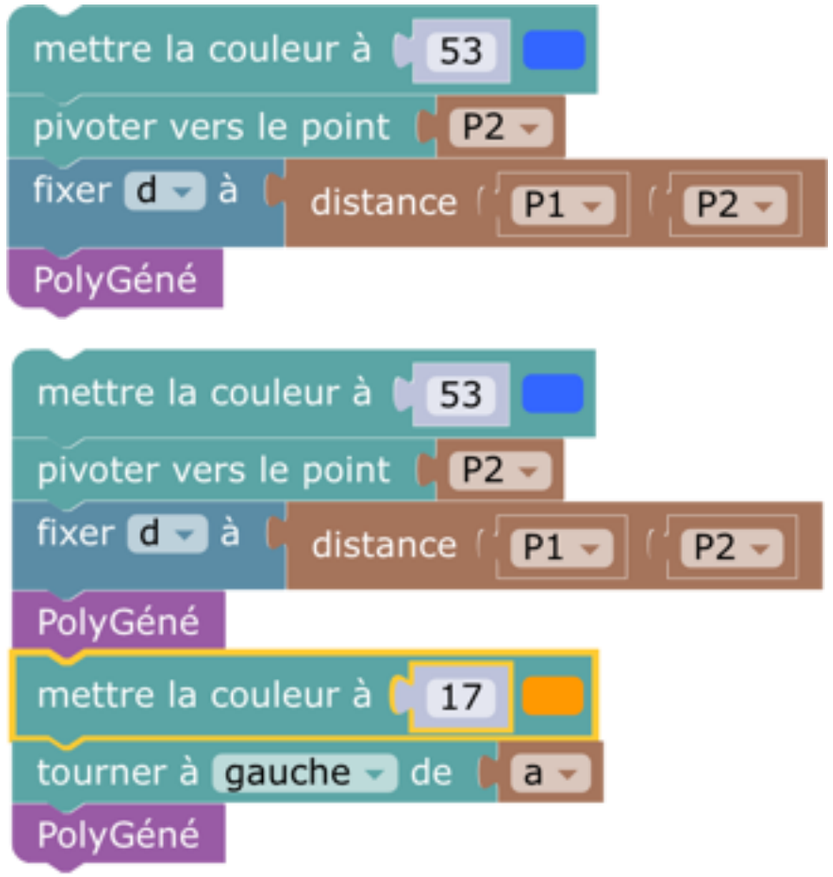
1. Il s'agit de changer 6 par n et 60° par 360/n puis
2. Mettre 2n dans la boucle externe et 180/n dans le passage d'un polygone à l'autre.
***Remarque** : la figure finale (et complétée) est disponible deux pages plus loin.*

Première utilisation des fonctions

On se propose de modifier la figure précédente de la façon suivante.
1. Tout d'abord, supprimer le remplissage des polygones.
2. Faire glisser une fonction sur l'espace de travail et l'appeler **PolyGéné** (on peut mettre des accents dans les fonctions) puis y glisser la boucle extérieure (tout le code après les trois lignes d'initialisation).
Le programme devient simplement ce que l'on a ci-contre.
3. Ajouter un curseur que l'on nommera **a**, et faire tourner la figure générale d'un angle a, après avoir changé de couleur (2° illustration).

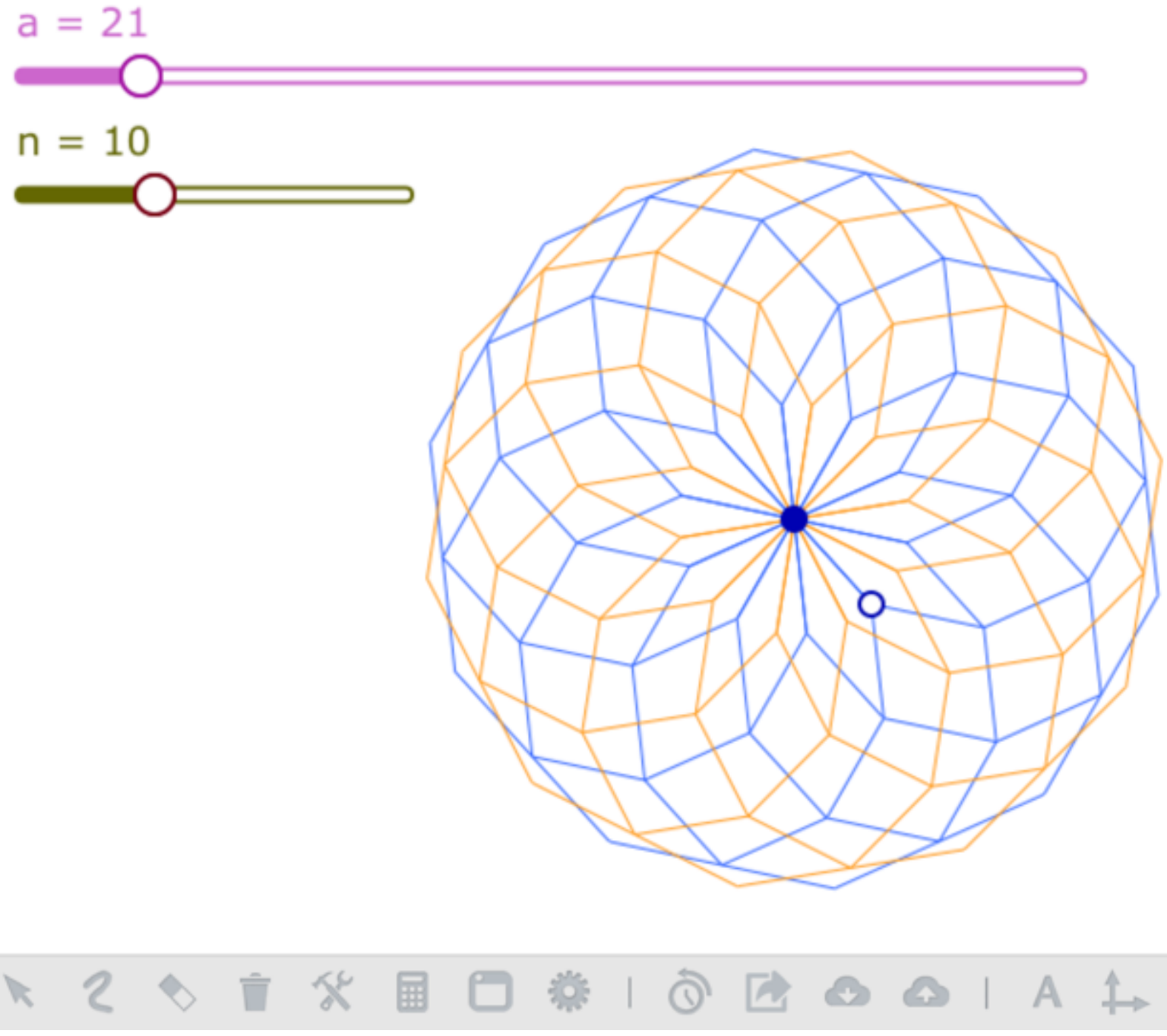


Pour les lecteurs qui ne veulent pas tout de suite s'aventurer sur le code, la page suivante contient la figure complète en manipulation.



On prendra un curseur assez long pour atteindre les nombres entiers. On peut aussi forcer le curseur à être décimal, ou demi entier. Cette construction est un outil pédagogique pour proposer des **explorations dynamiques** aux élèves, afin de mettre en évidence les propriétés de chaque figure (en fonction de n) :
Quel angle minimum pour que les deux figures soient superposées, pour n=8 ? pour n=6 ? pour n=5 ? et même n=20 ? (Réduire la figure par P2).
Pour n=12 tester aussi l'angle moitié (15°).
Pour répondre à la question avec n=10, il faut garder l'incrément libre.

Figure dynamique 3.3 – La figure précédente de rotation d'une figure... dans la même trace de tortue



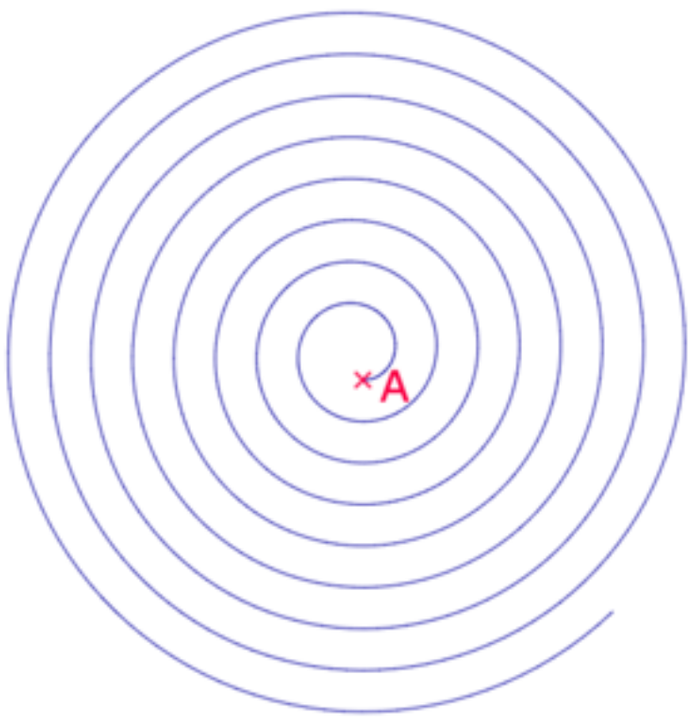
*En mode consultation à l'ouverture, mais vous pouvez accéder au code en passant en **mode standard**.*

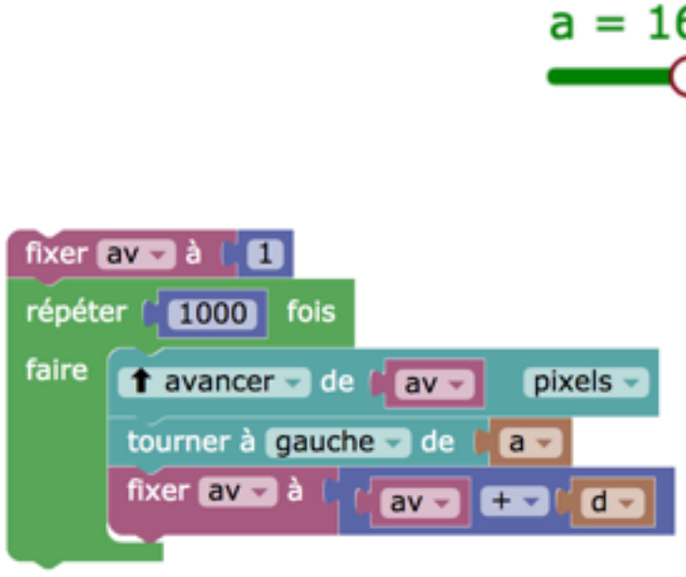
Jouer avec des spirales

Dans les pages suivantes vous pourrez manipuler différents types de spirales largement paramétrées par des curseurs.
Bien entendu, dans un contexte scolaire, on commencera sans paramètre, avec des données numériques qui sont modifiées dans le code avant d'introduire des curseurs.
Les élèves apprécient la diversité en manipulant les curseurs sur ce type de spirales.

1. **Spirales additives : la variation de l'avance est arithmétique**

1.a. Augmentation infinitésimale arithmétique. On commence par un exemple surprenant car il s'agit d'ajouter des centièmes de pixels à la longueur initiale. Avec un angle quelconque, cela produit des figures souvent esthétiques, avec un code minimaliste. L'illustration de gauche correspond à un angle petit, inférieur à 3° avec une variation de l'ordre du centième de pixel.





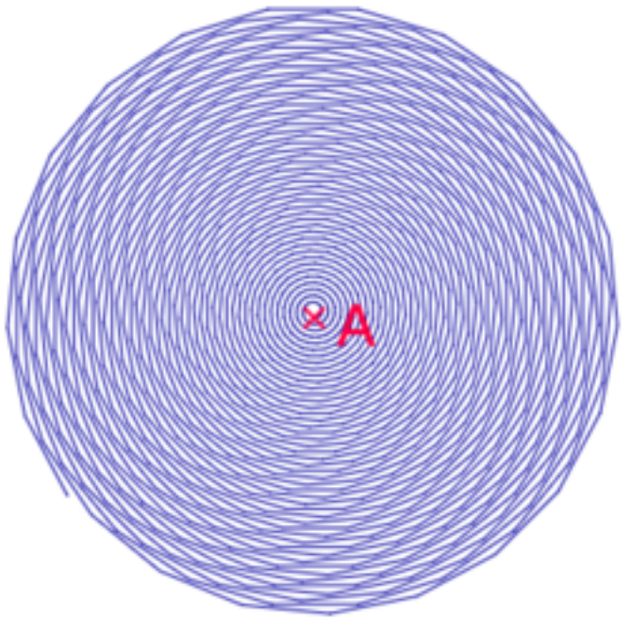
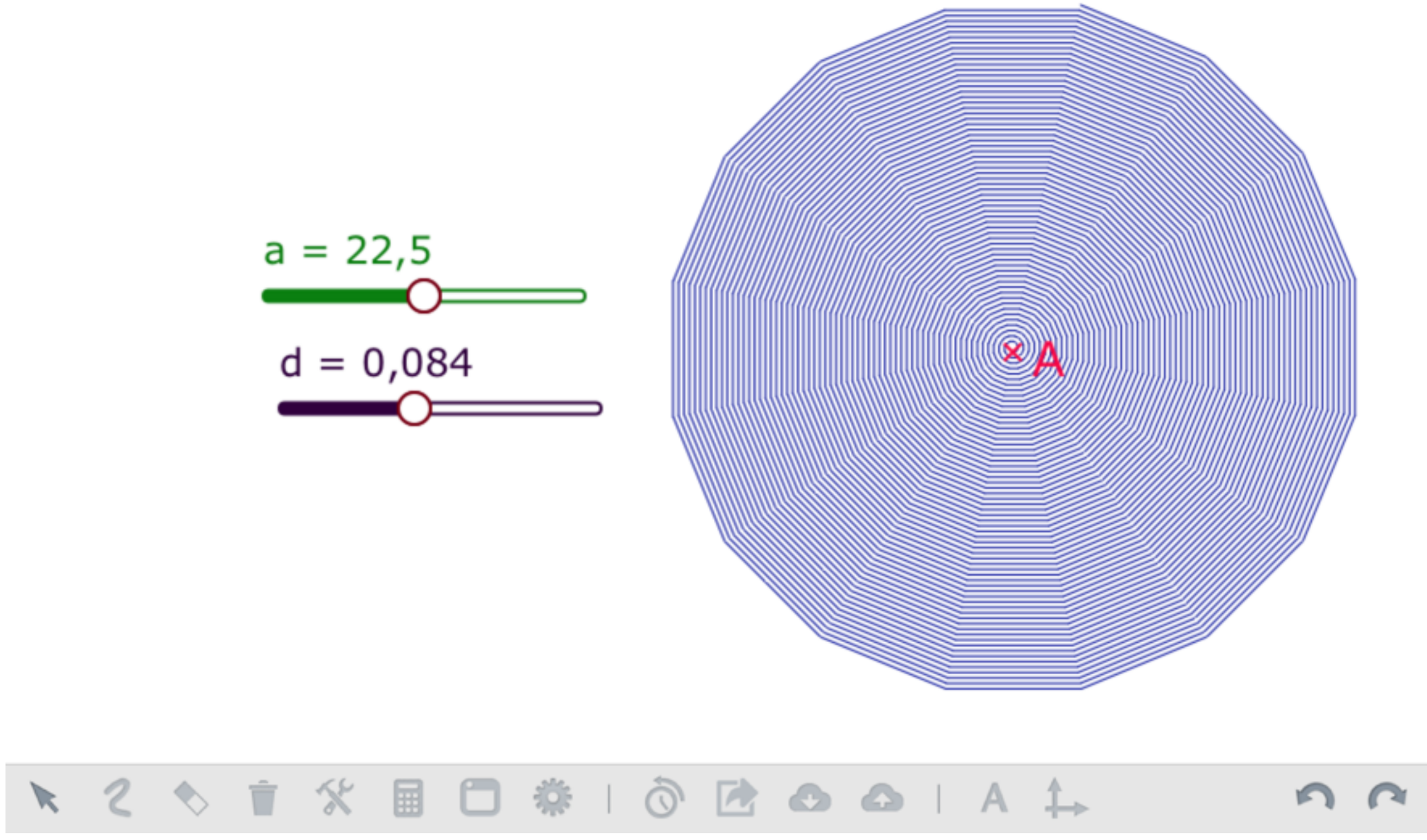


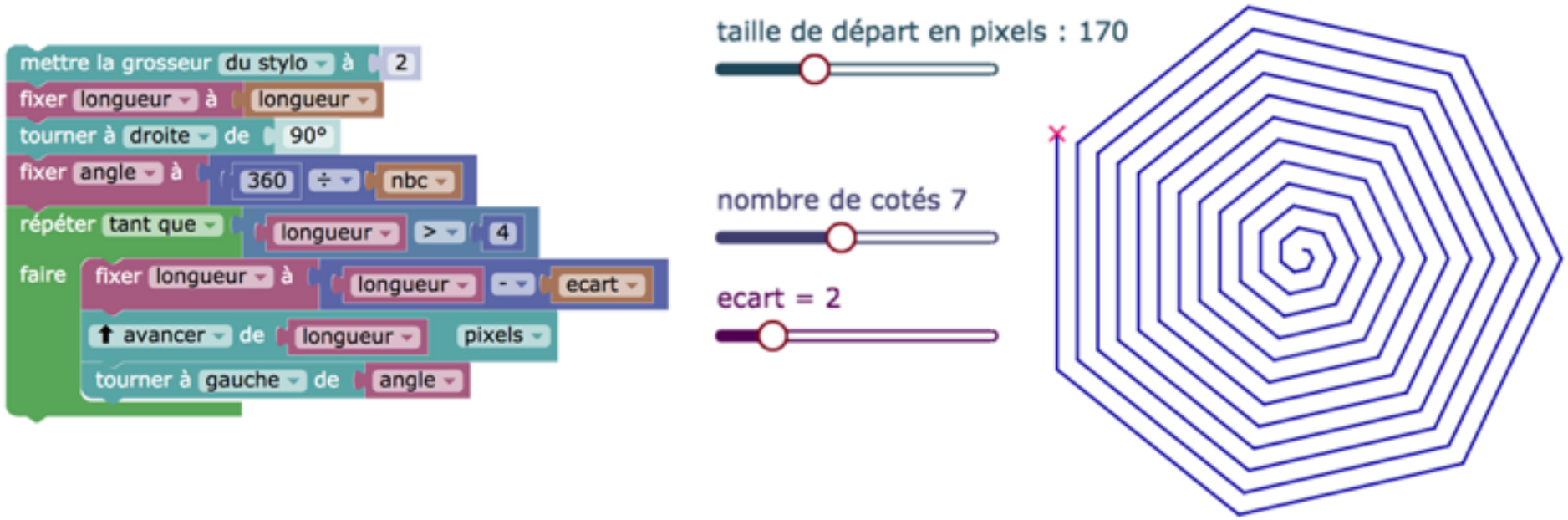
Figure dynamique 3.4 – Spirale par augmentation infinitésimale additive



Ouverture sur 22,5° – Les côtés sont-ils parallèles ? N'hésitez pas à tester les paramètres – d et a – les deux petits.

1.b. Diminution entière sur une longueur initiale.

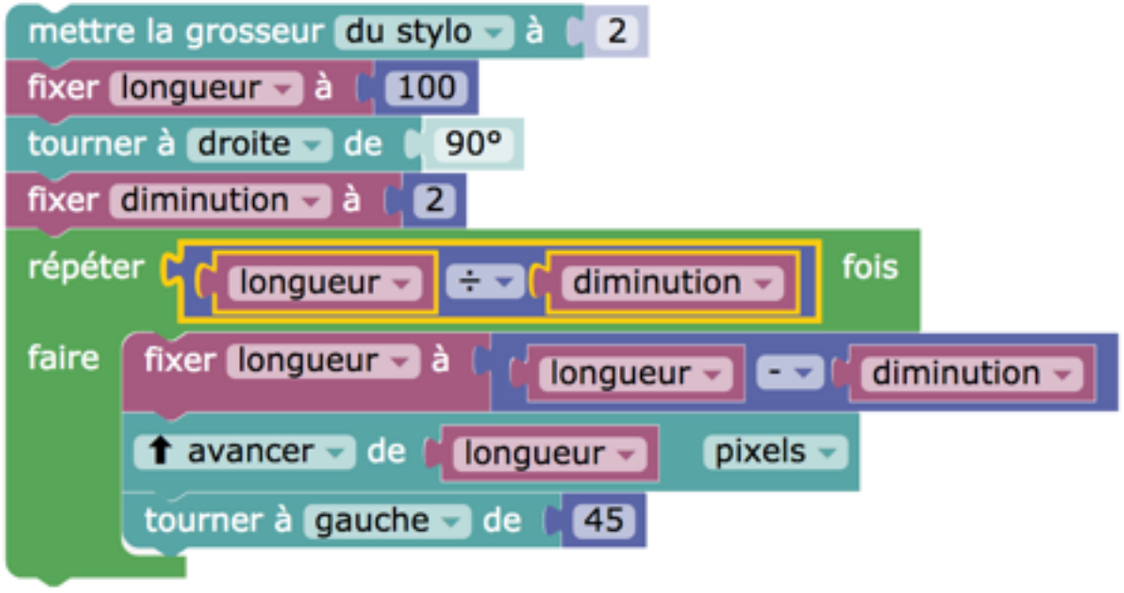
Cet exemple est présenté aussi pour aborder la seule difficulté que l'on peut rencontrer dans la programmation avec DGPad : l'utilisation d'une boucle **tant que**. En effet, comme tout est en temps réel, pour réaliser une telle boucle, il faut d'abord **la désactiver**, ce qui ne se fera pas en classe. En revanche, on peut donner ce programme avec des paramètres numériques (avec une forme initiale hexagonale par exemple). Les élèves n'ont alors que des modifications à faire, mais pas à construire la structure.

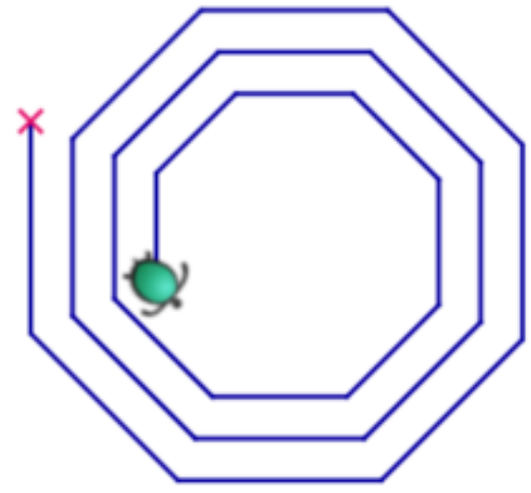


D'une manière générale, quand on commence la programmation avec le Blockly de DGPad, on évitera dans un premier temps cette boucle **tant que** qui, conceptuellement, n'est pas conçue pour son remplissage en temps réel (d'où la nécessité de la désactiver pour la construire, détaillé page 154) : le temps réel nous fait découvrir des choses parfois surprenantes.

Avant de présenter cette figure à la manipulation, voyons d'abord une erreur rencontrée lors d'une formation, dans le cadre d'une recherche d'une solution pour éviter le **tant que**. L'erreur est intéressante car faite par un programmeur averti des langages compilés, et donc qui pose question sur la façon dont le JavaScript est interprété.

Une première démarche est de partir de ce code, a priori tout à fait correct.

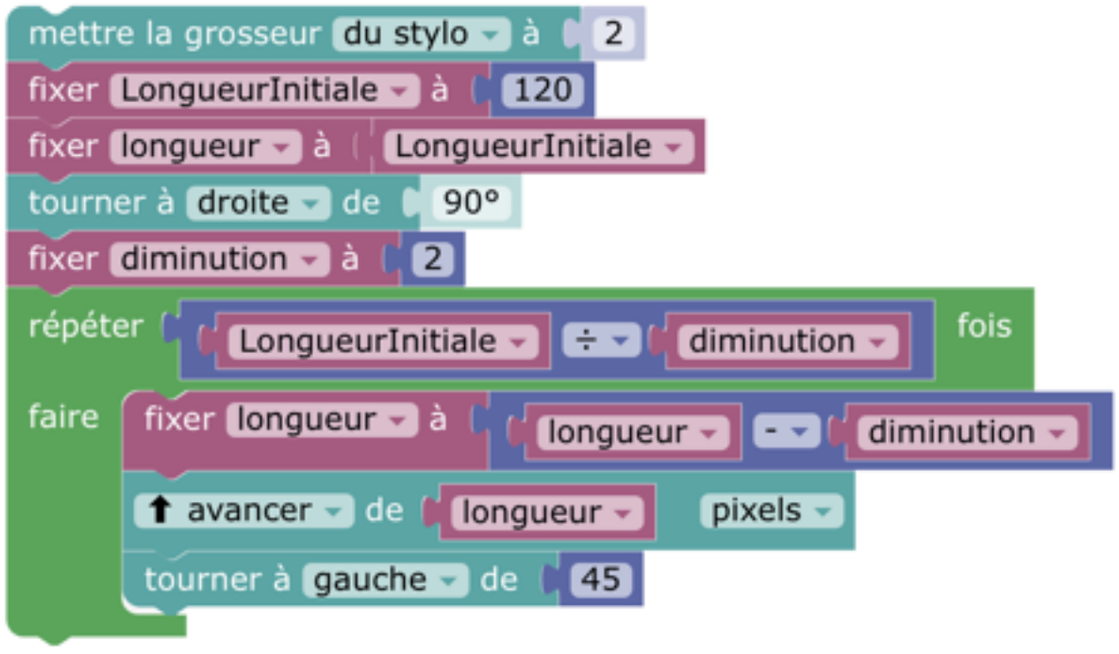


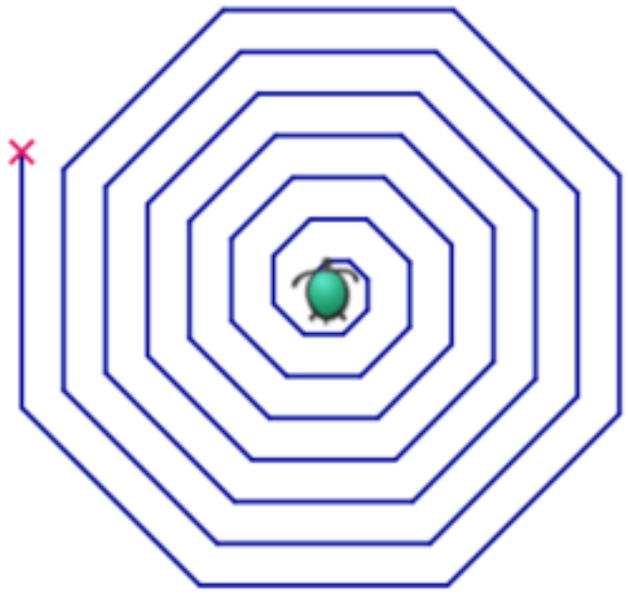


Mais alors que l'on devrait avoir 50 itérations, (100:2), on n'a que 25 segments.

La raison est très simple : comme la longueur est modifiée dans la boucle, le nombre d'itérations est recalculé au passage suivant de la boucle, du au fait que le JavaScript est interprété.

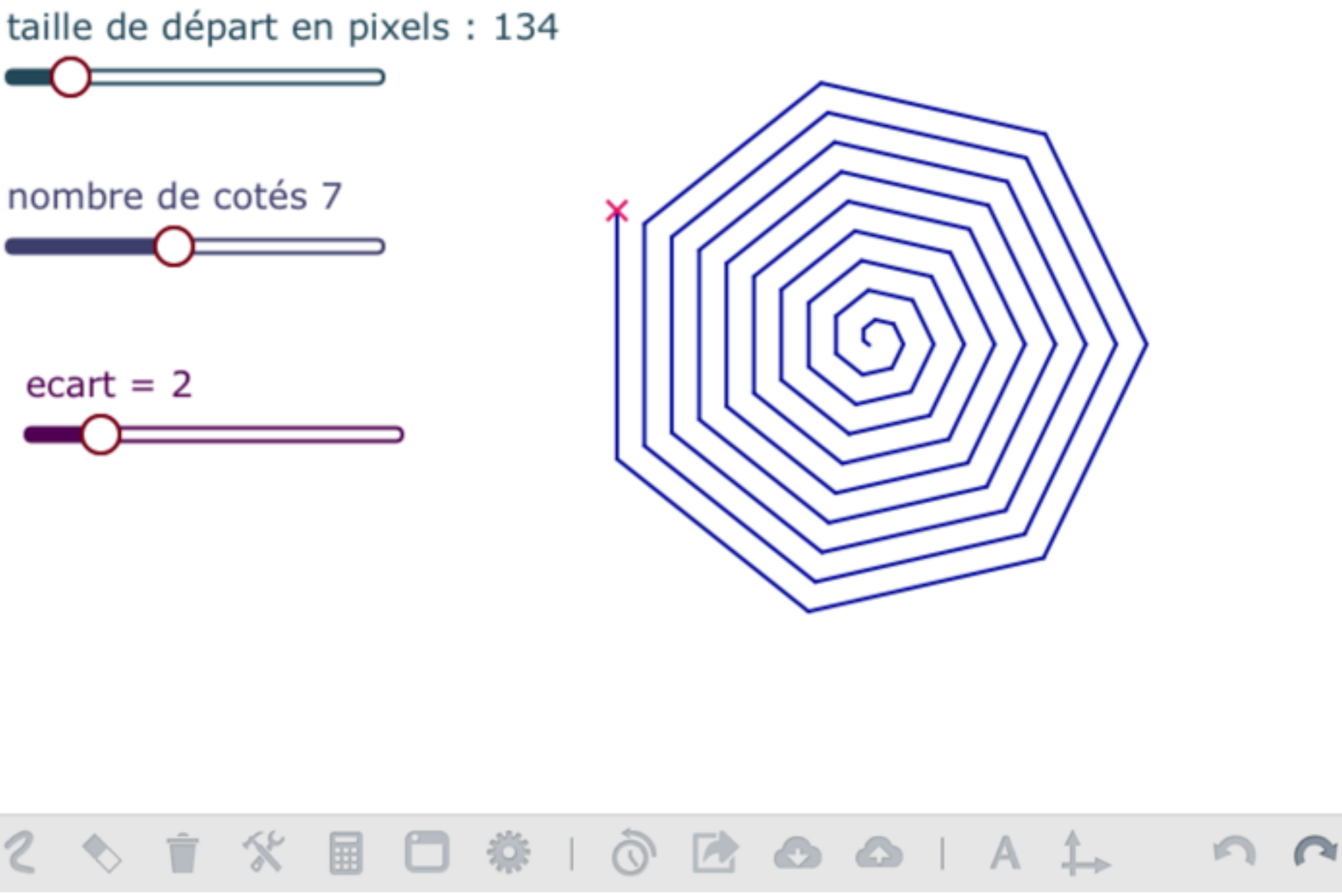
Une solution simple consiste à ajouter une variable et donc mettre dans la boucle répéter une constante.





On a choisi une version numérique, mais si on avait utilisé une variable de la figure (un curseur) **cette question ne se serait pas posée.**

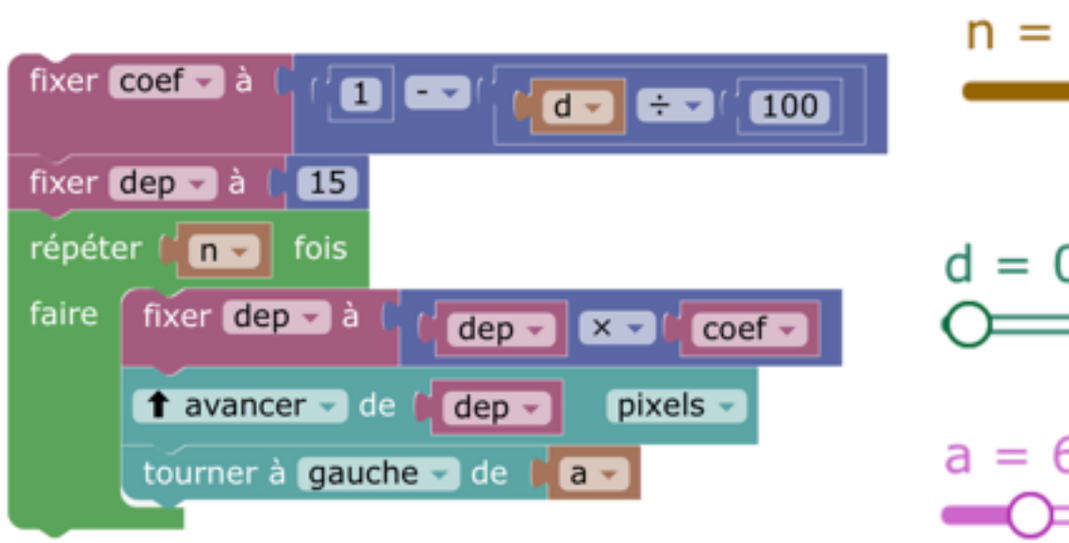
Figure dynamique 3.5 – Spirale additive à pas entier

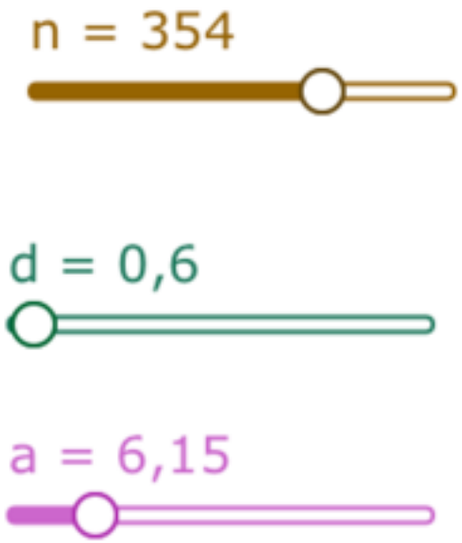


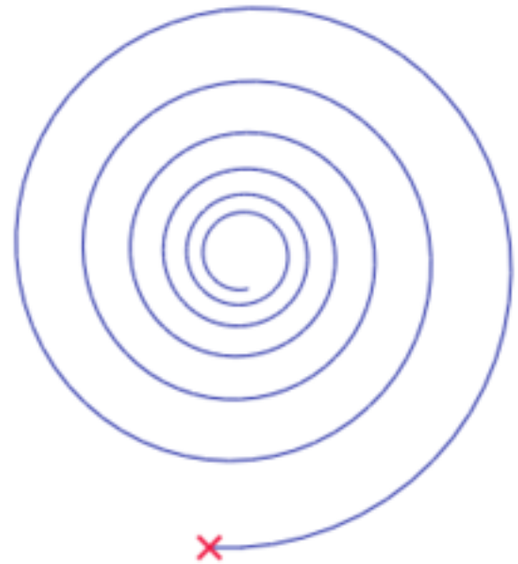
Le code est, bien entendu, dans l'unique point de la figure.

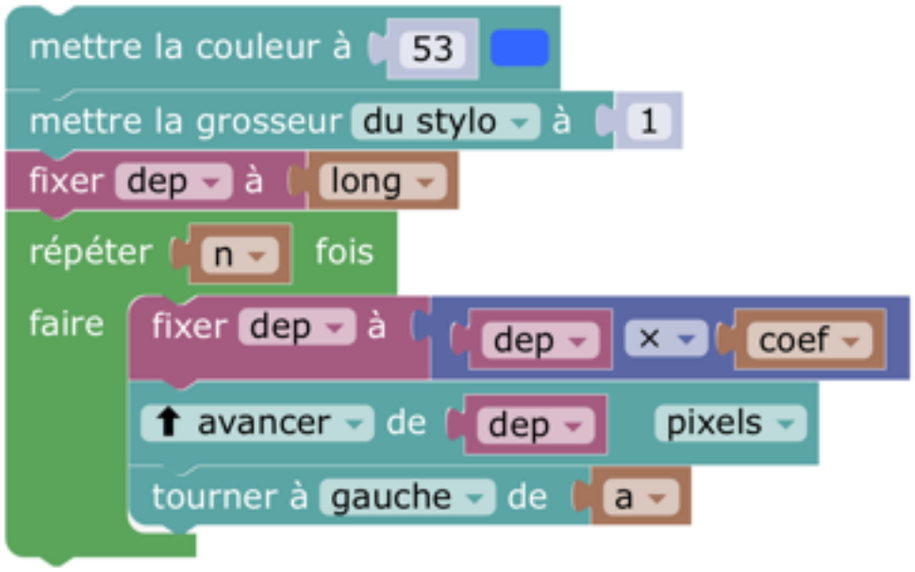
2. Spirales multiplicatives

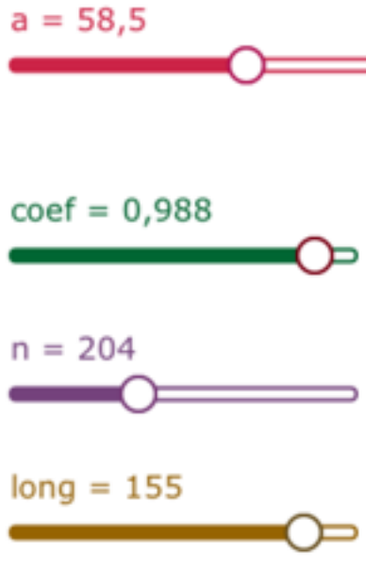
Version logarithmique et version «jolygone» : ce sont les mêmes avec des conditions initiales et des angles de rotation dans des registres différents. Dans les deux cas, il y a un point de convergence.











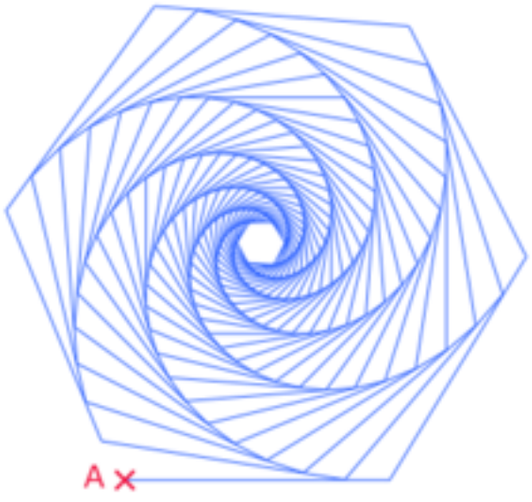


Figure dynamique 3.6 – Spirale logarithmique

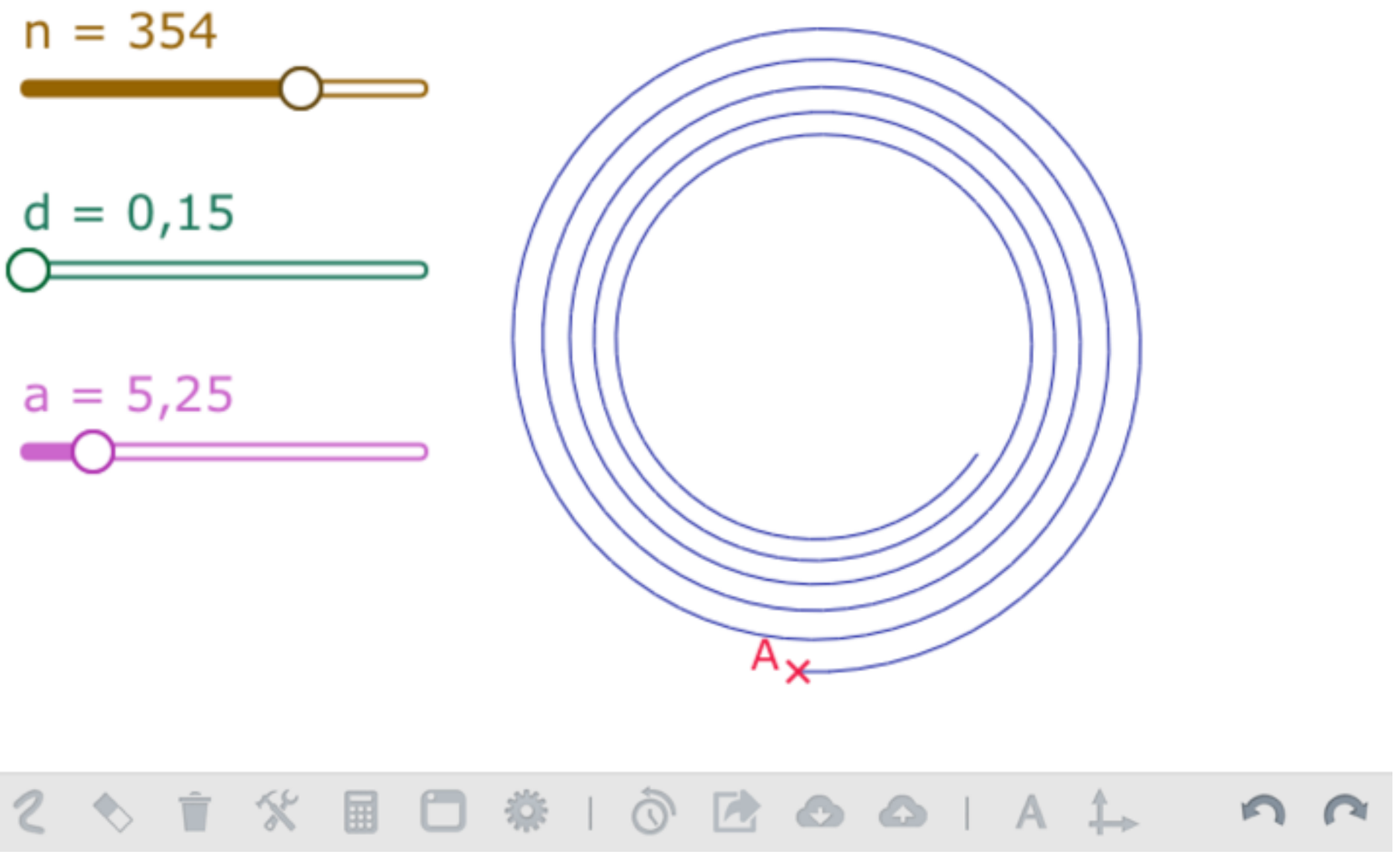
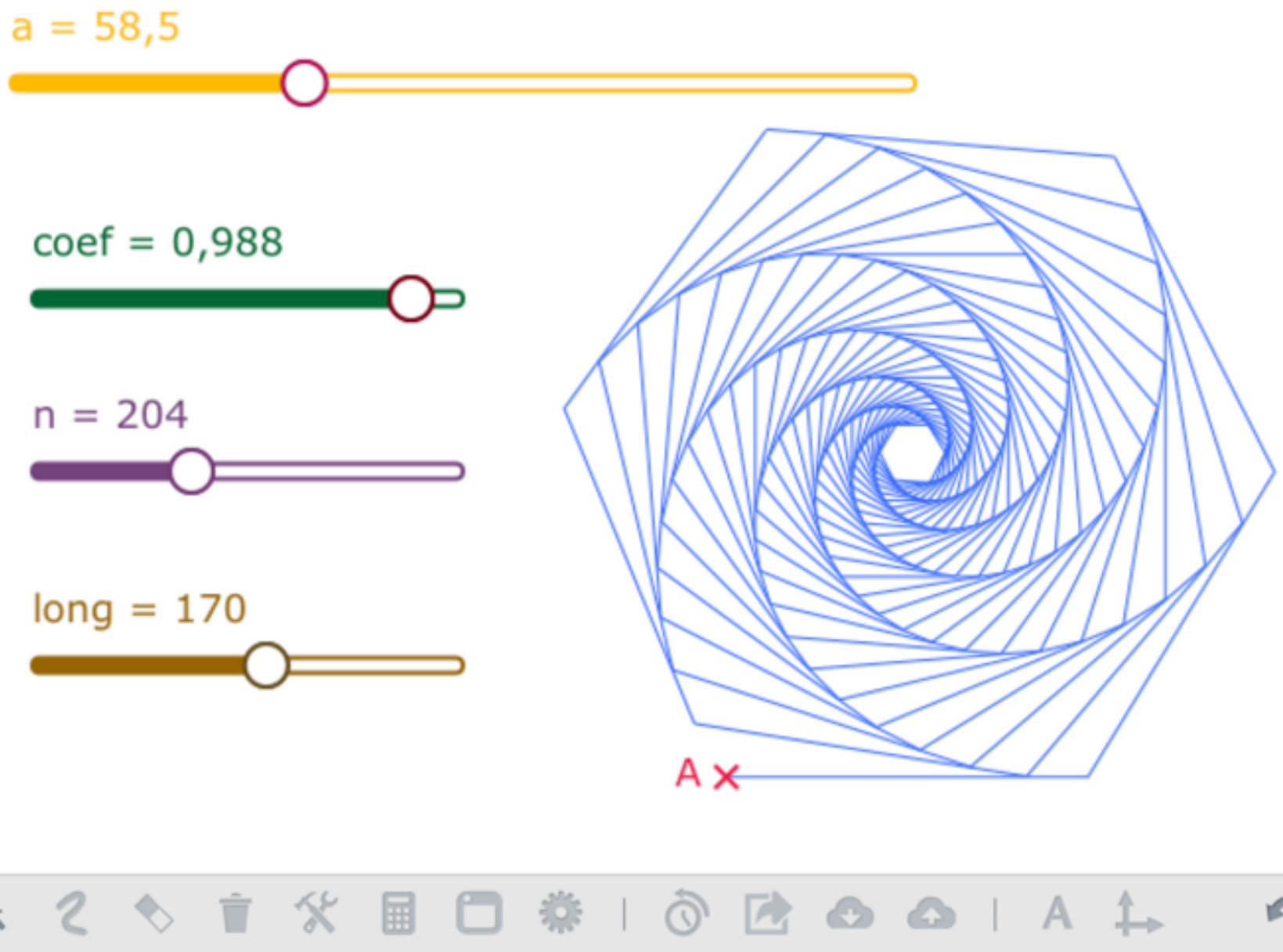
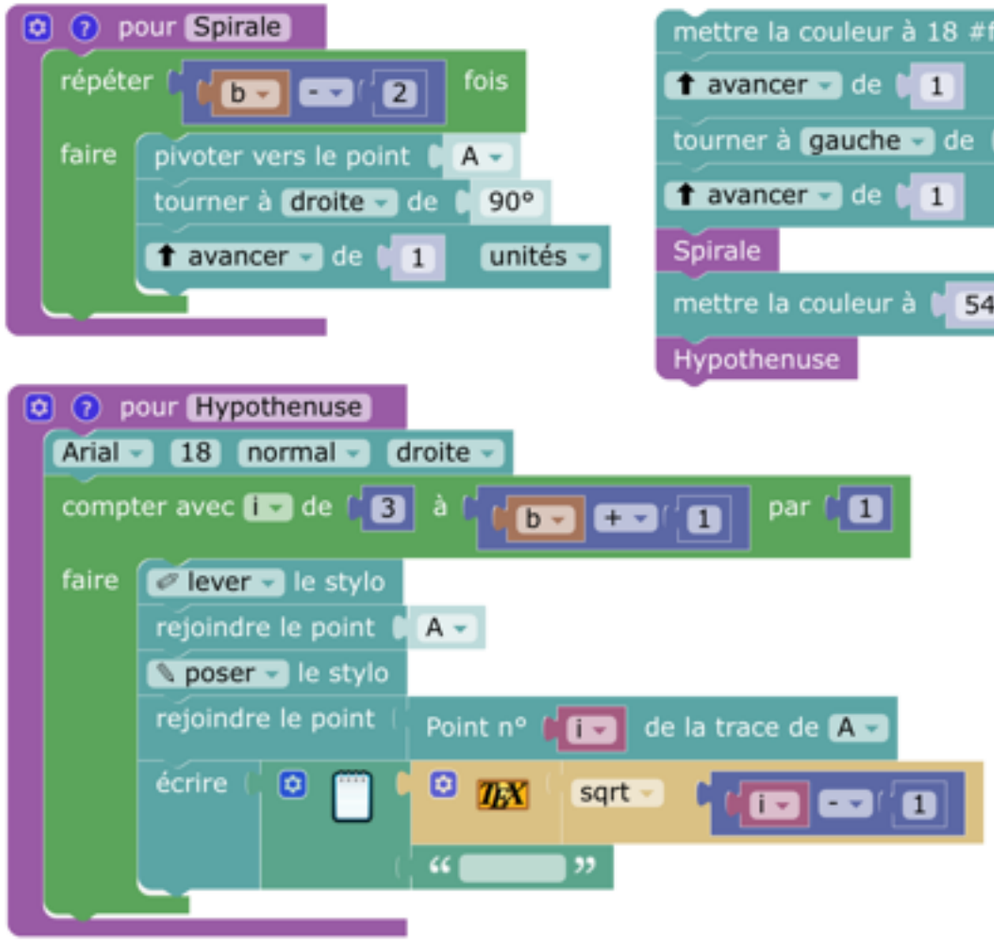


Figure dynamique 3.7 – Jolygones



3. Spirale de Pythagore - Utilisation du TeX sur les hypoténuses

Dans cette figure, on a choisi de construire les côtés d'unité 1 dans un premier temps et les hypoténuses ensuite, et ceci sans calculs, simplement par l'usage de la tortue, avec l'outil **Point de la trace**.
D'où la validation en prenant un point sur la trace, par curseur, pour comparer la distance à A et la racine carrée construite.



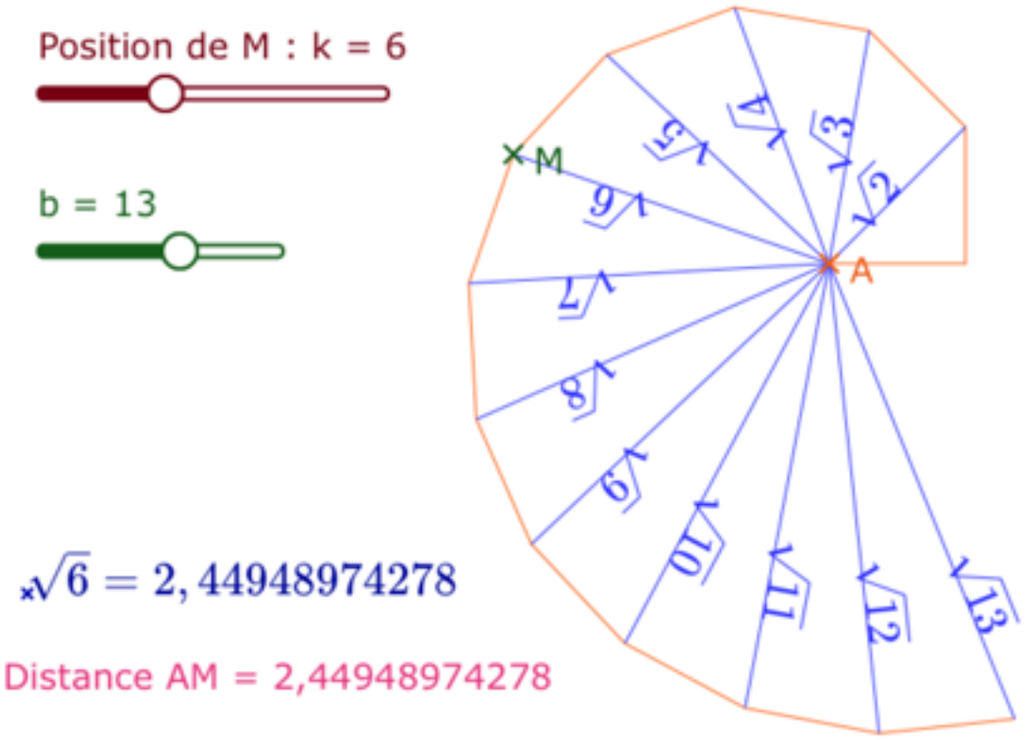
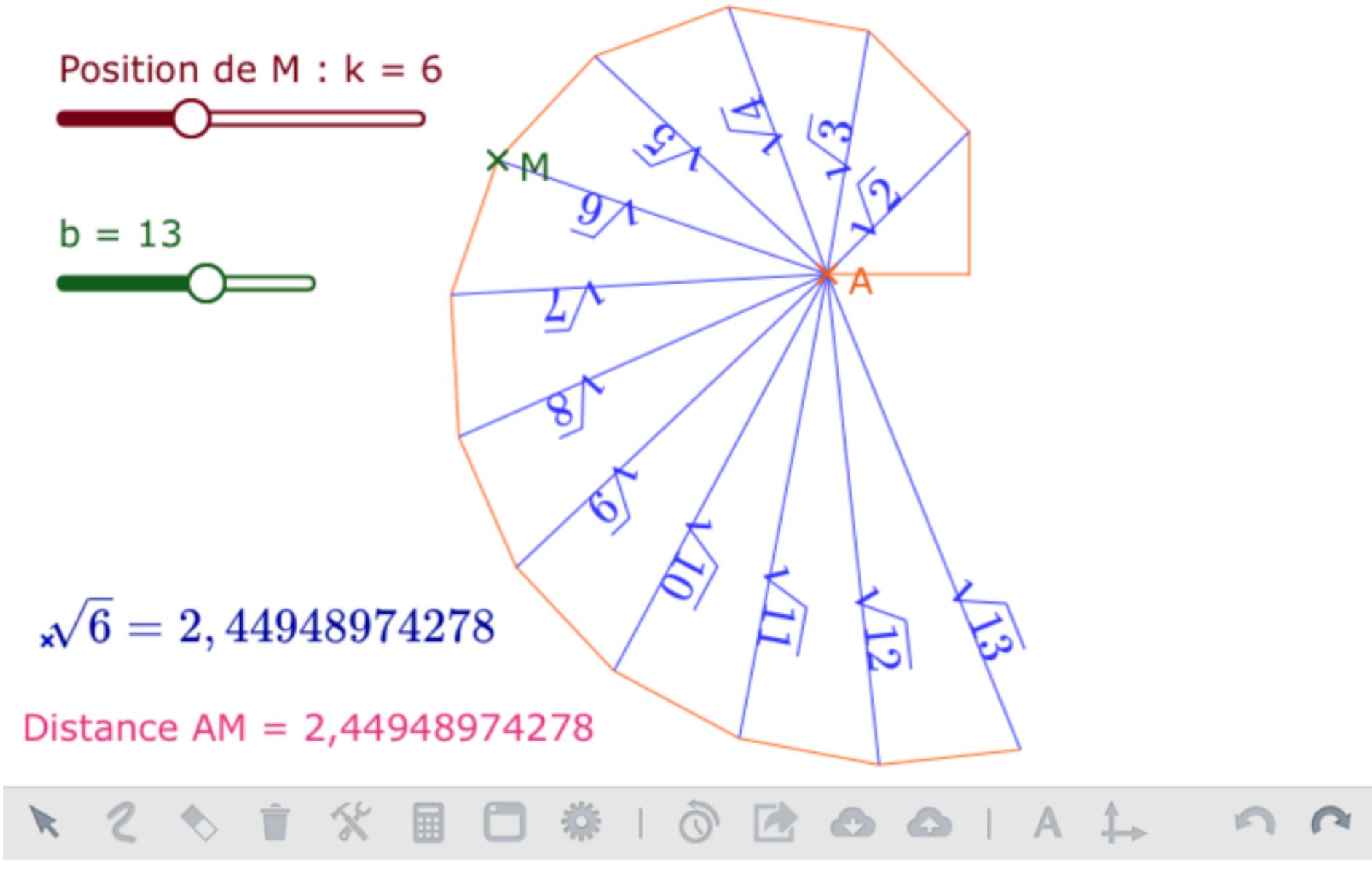


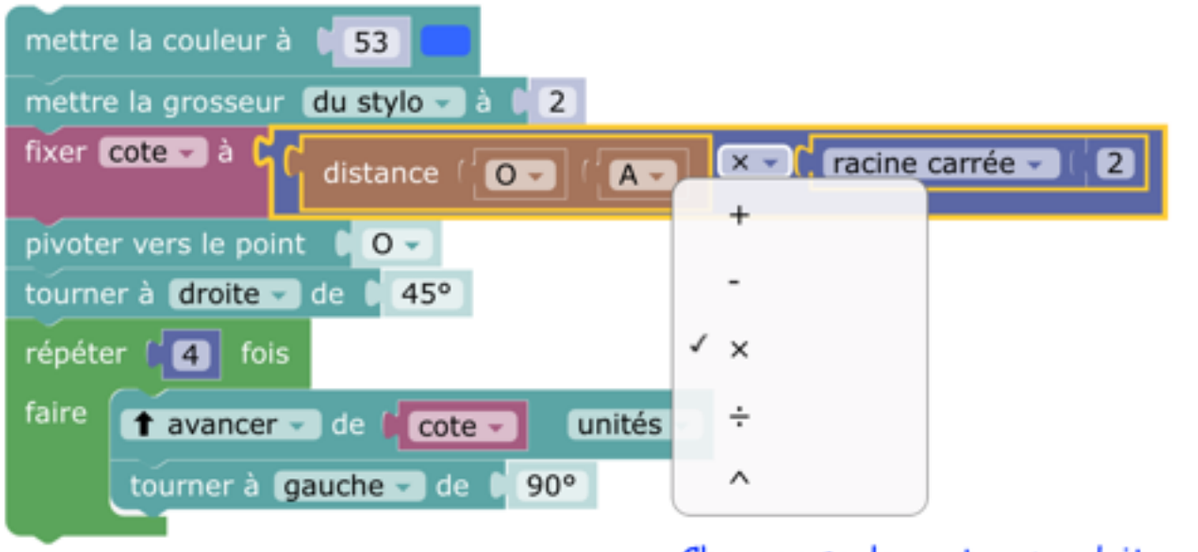
Figure dynamique 3.8 – La figure « Ballade sur la spirale de Pythagore ».

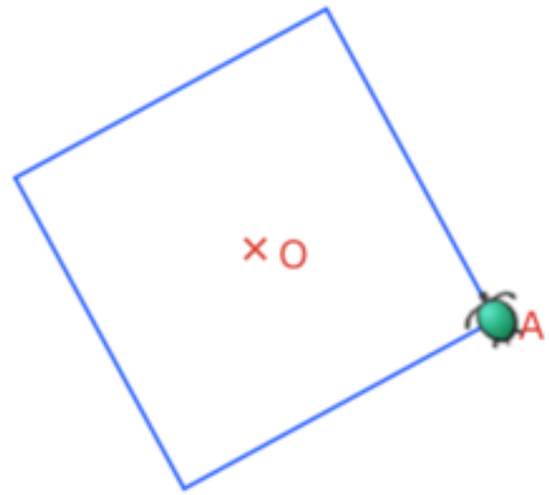


M se déplace par le curseur k pour comparer la longueur mesurée et la longueur calculée.

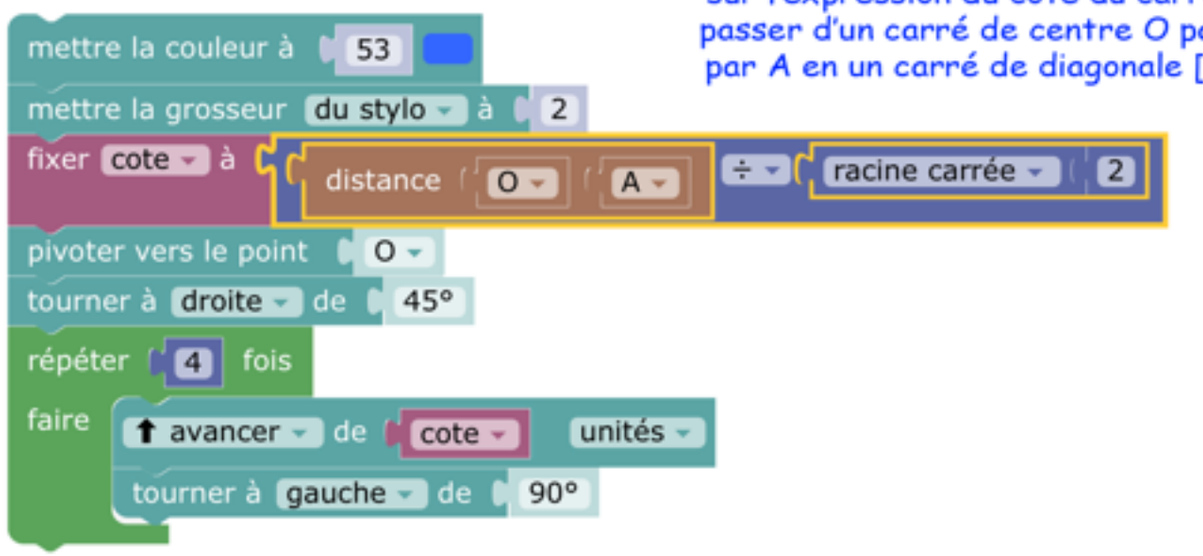
Spécificités dynamiques et géométrie

Le fait que l'on puisse travailler avec des points bien spécifiques, accessibles en manipulation directe avec une tortue qui réagit en temps réel permet des explorations et des mises en perspectives géométriques nouvelles.
Voyons quelques exemples simples.





Changer seulement un produit en division sur l'expression du côté du carré fait passer d'un carré de centre O passant par A en un carré de diagonale [OA].



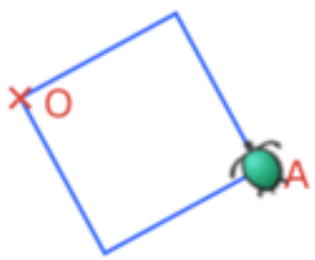
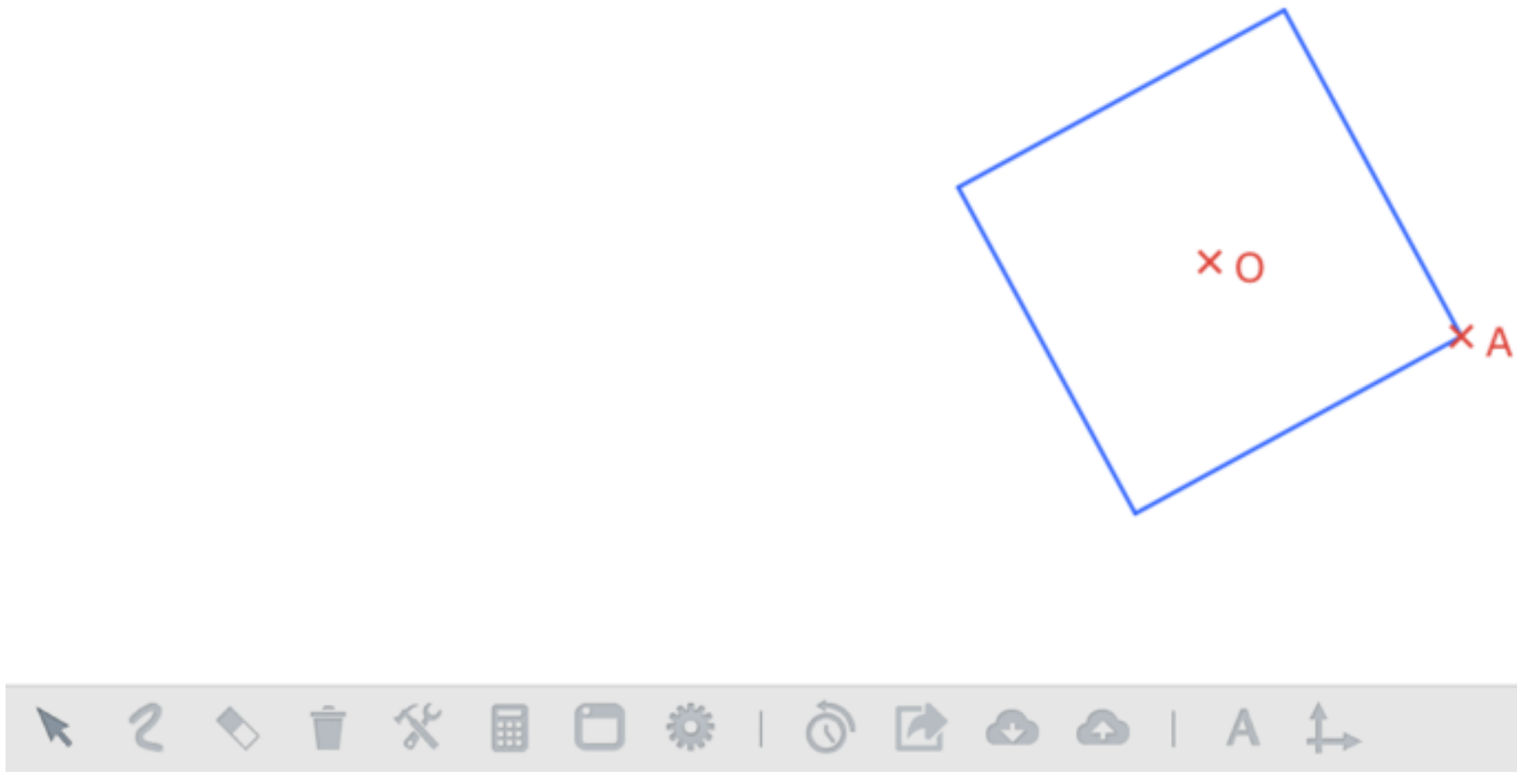


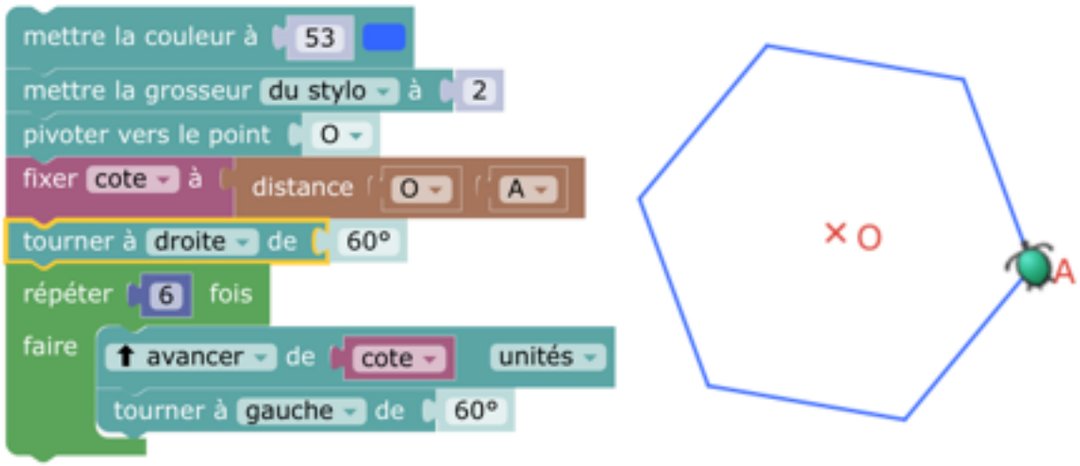
Figure dynamique 4.1 – Carré par centre et point, puis par diamètre



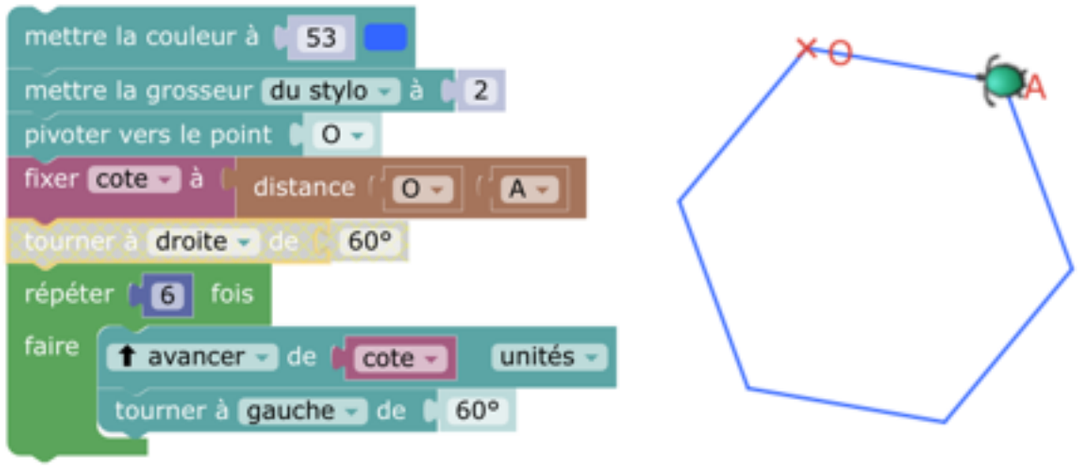
Le code est dans le point A – Pour faire la manipulation décrite à la page précédente.

Sur l’hexagone régulier

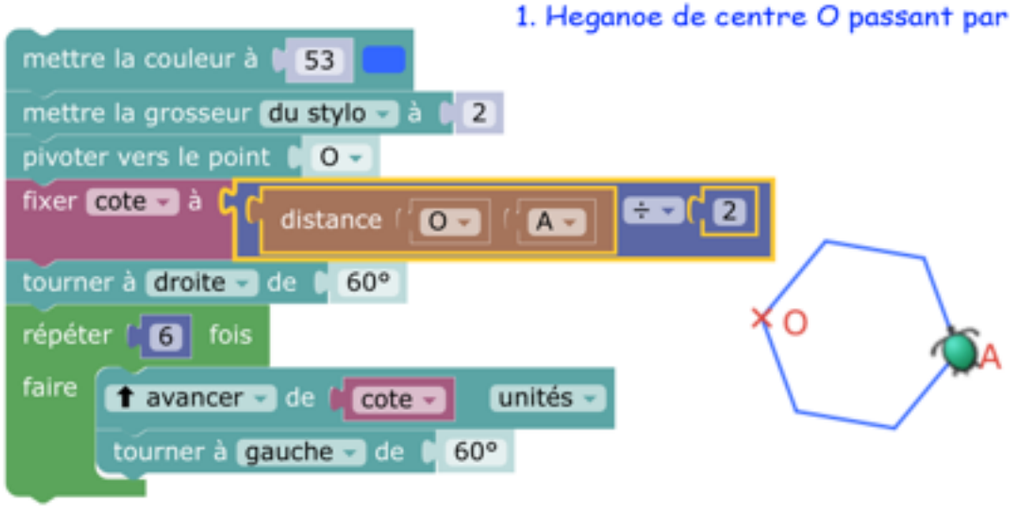
L’hexagone est la figure la plus simple à utiliser en classe pour faire se rencontrer les propriétés géométriques et le codage car, de par ses propriétés, il n’y a aucun calcul particulier, au contraire du carré.



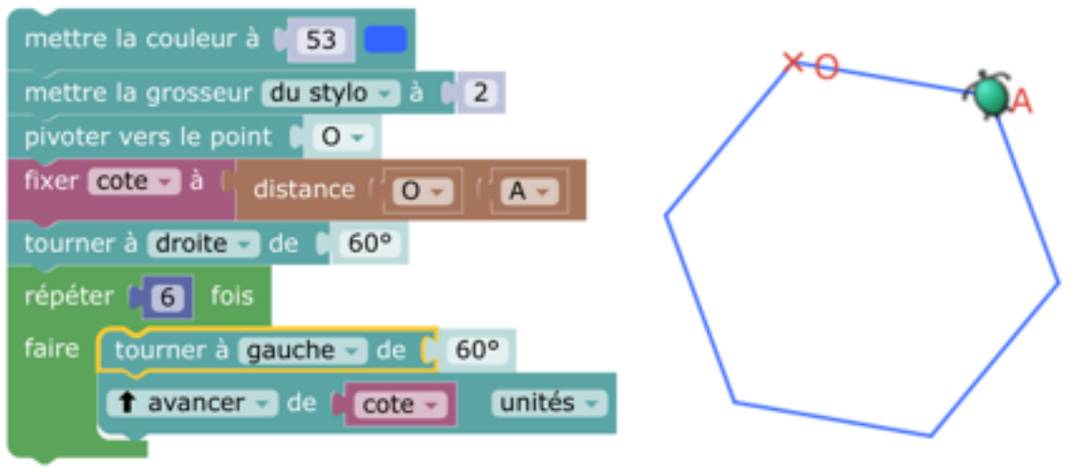
1. Heganoe de centre O passant par A.



2. Oubli de la rotation initiale avant la boucle, hexagone de côté [OA].



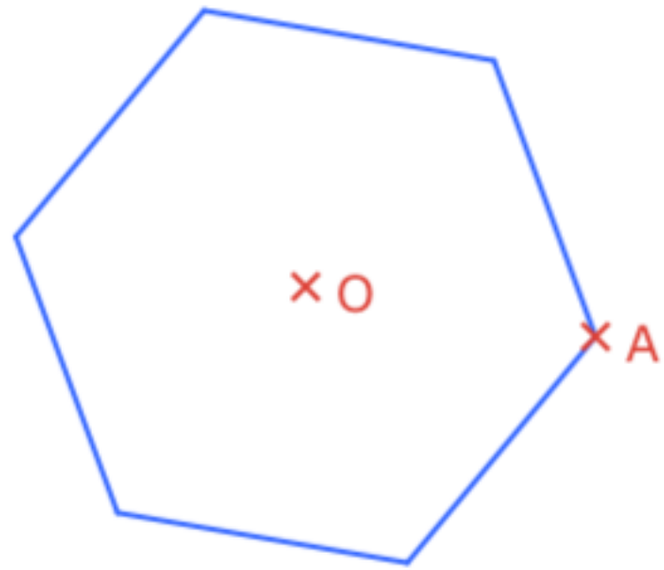
3. Le côté est divisé par 2. Pourquoi l’hexagone devient de diamètre [OA] ?



4. En inversant les deux ligne de la boucle, on retrouve l’hexagone de côté [OA].

Remarquer les positions finales de la tortue, différentes, dans les deux illustrations de droite.

Figure dynamique 4.2 – Manipulations sur l'hexagone par centre et point



Pour refaire les manipulations proposées à la page précédente, voir le code en A.

Itérations sur le carré

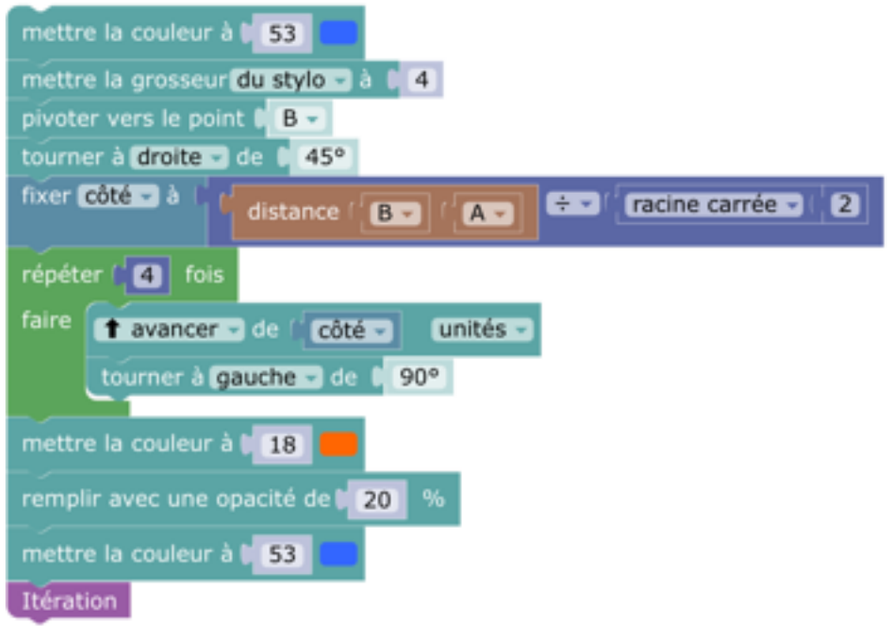
Ce paragraphe a surtout été ajouté pour montrer les possibilités de gestion de la couleur (contour et intérieur du carré).

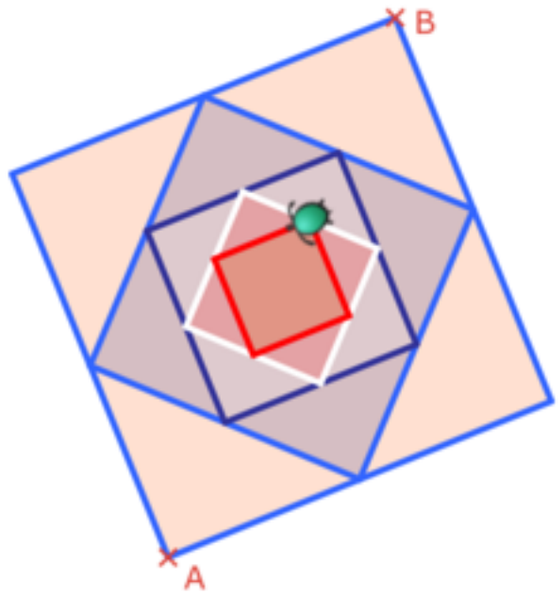
En effet, le bloc **RBGColor** de la catégorie **Aspect** ne s'applique pas à la tortue : avec la tortue, on ne dispose que de 72 couleurs que l'on peut faire itérer par ajout d'un indice de couleur.

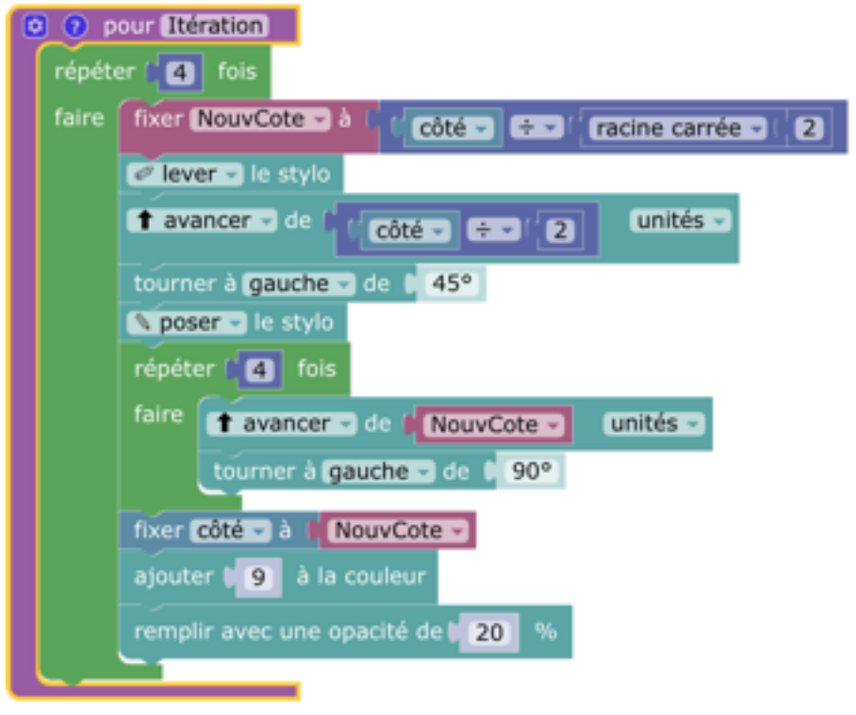
RGBColor s'applique aux listes créées avec Blockly sans la tortue (un chapitre y est consacré)

Exercice de la page suivante :

Ajouter le curseur **n** de l'illustration de droite.







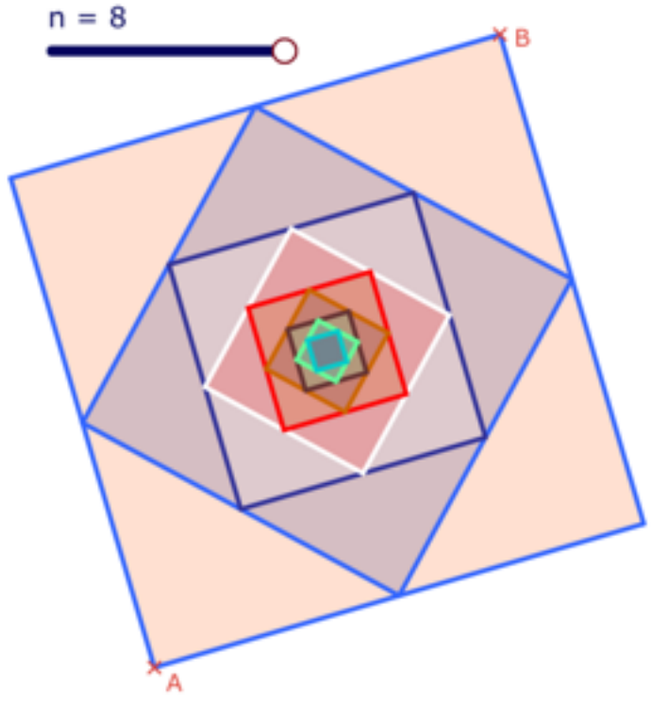
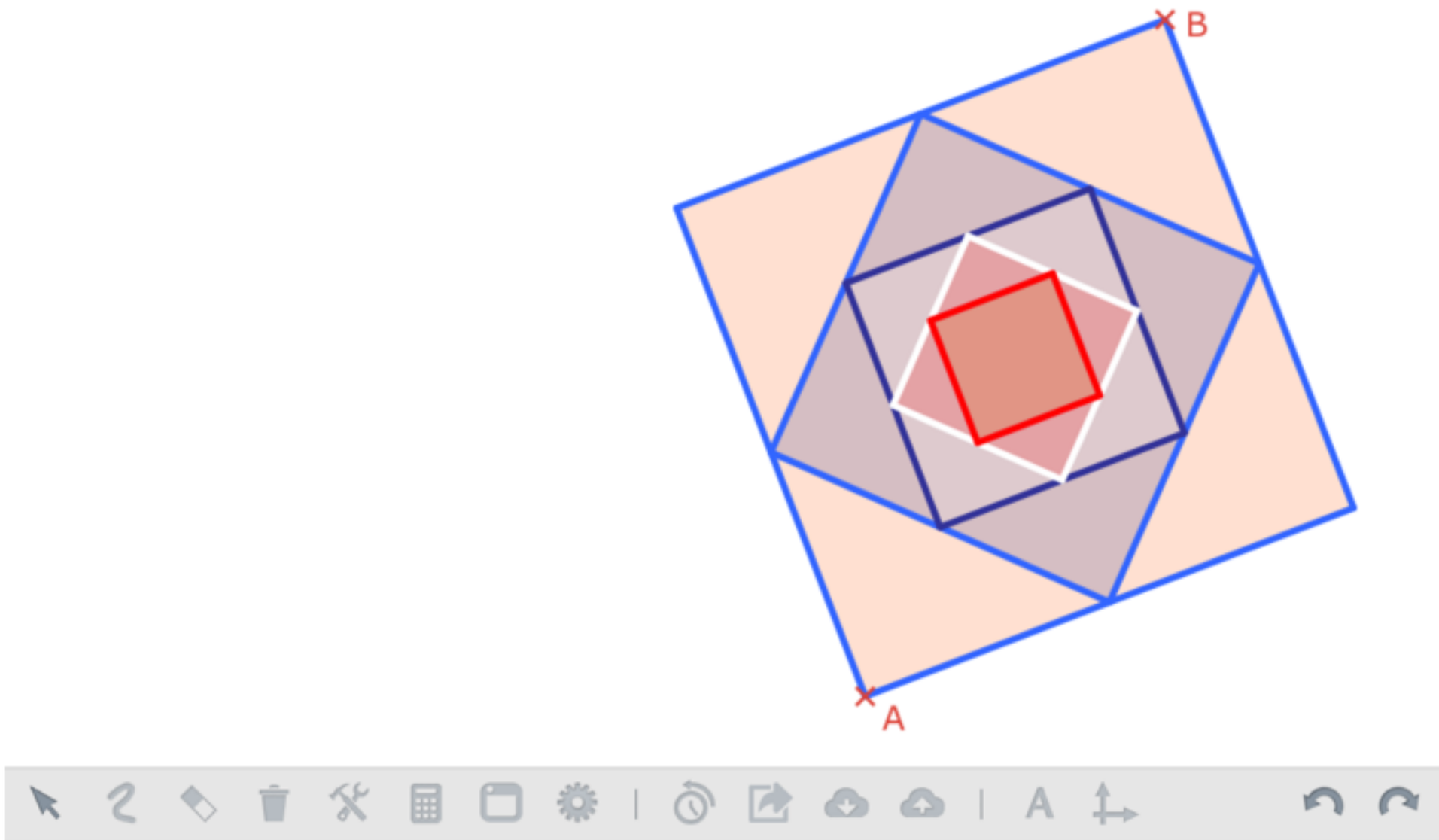


Figure dynamique 4.3 – Itération d'un carré donné par deux sommets opposés - modifiables.



*L'activité consiste à ajouter un curseur **n** et à l'insérer dans l'itération.*

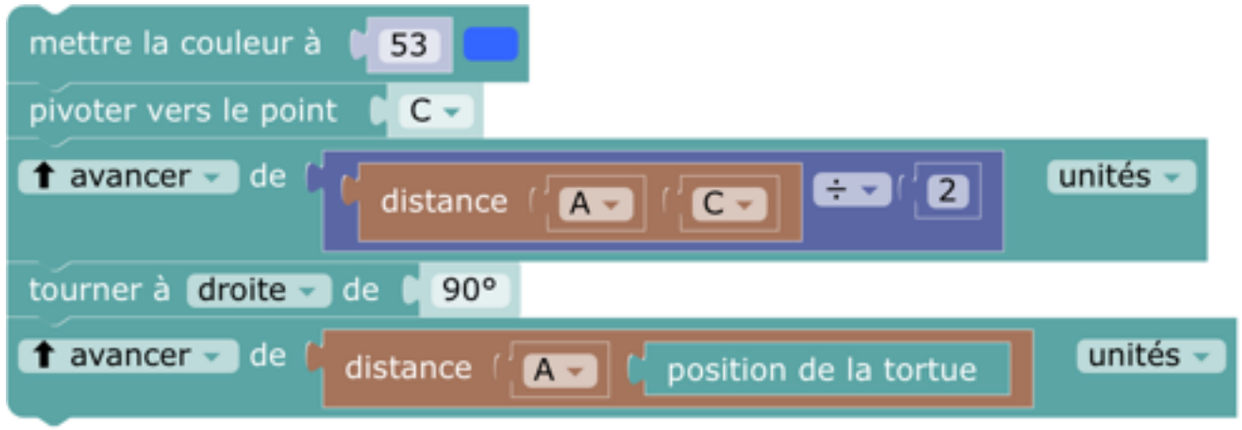
Algébrisation de la tortue

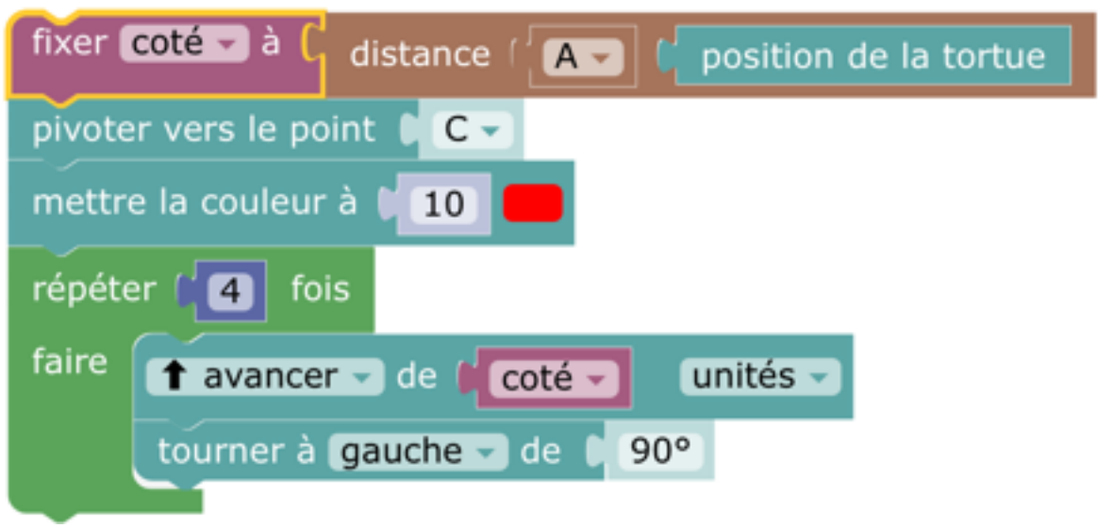
Un domaine tout à fait nouveau que permet cette tortue dynamique est la rencontre entre les **propriétés algébriques des figures** et leur codage. Plus précisément, le codage dynamique des propriétés géométriques des figures va les éclairer d'un caractère plus ou moins algébrique. Qui dit algèbre laisse à penser à l'utilisation d'une inconnue, ou d'une variable. Cette variable va être la **position de la tortue**. Et bien entendu, tout est dynamique.

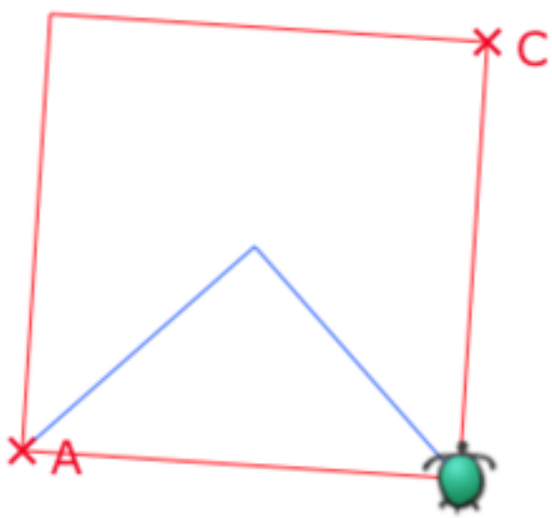
On reprend le tracé du carré de diagonale [AC] mais sans le calcul de la longueur du côté, simplement par les propriétés de ses diagonales.
La **position de la tortue** permet alors :

1. D'aller au point B.
2. De définir le côté **sans aucun calcul**.
3. De terminer le carré classiquement.

Cette démarche offre de nombreuses possibilités que nous allons détailler dans ce chapitre.





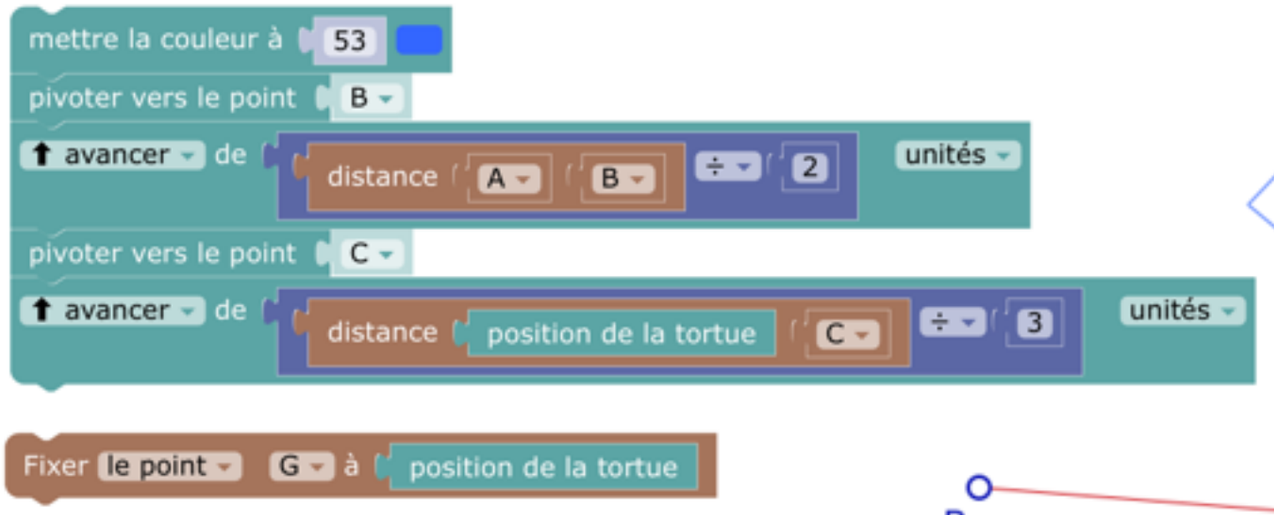


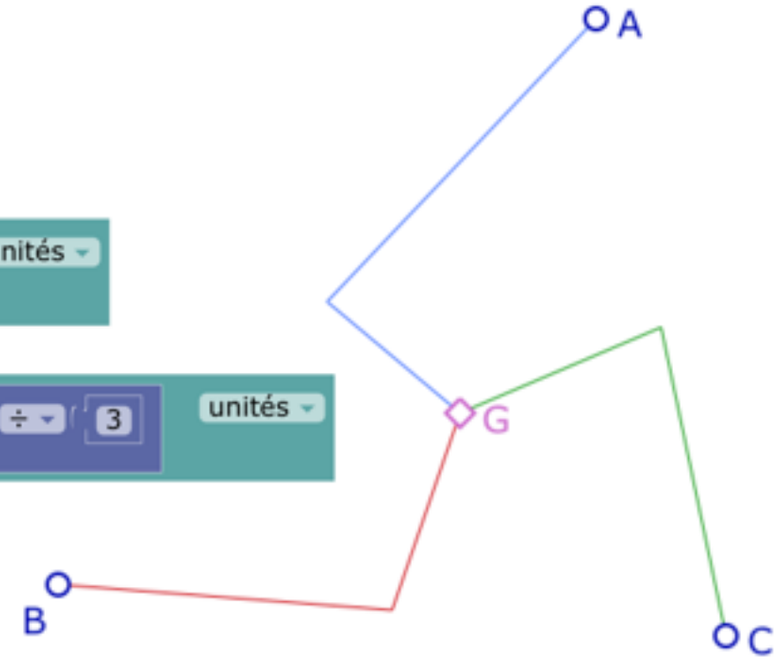
Isobarycentre dans le plan et l'espace

Avec l'intersection des médianes, très simple à réaliser à la tortue, on va aller un peu plus loin d'un point de vue conceptuel. C'est l'occasion de rappeler que Blockly est un **comportement**, il ne fait donc que modifier des points mais ne peut pas en créer. C'est pourquoi, si on veut placer un point G à la position – dynamique – de l'isobarycentre, il faut que le point existe dans la figure. Alors un point peut contenir un comportement qui place ce point G là où on le souhaite.

Le code ci-contre est celui de la trace bleue. On place la tortue au tiers de la médiane issue de C, avec une trace partant de A.
Puis un bloc de la catégorie **Expressions** permet de placer G à cet endroit.

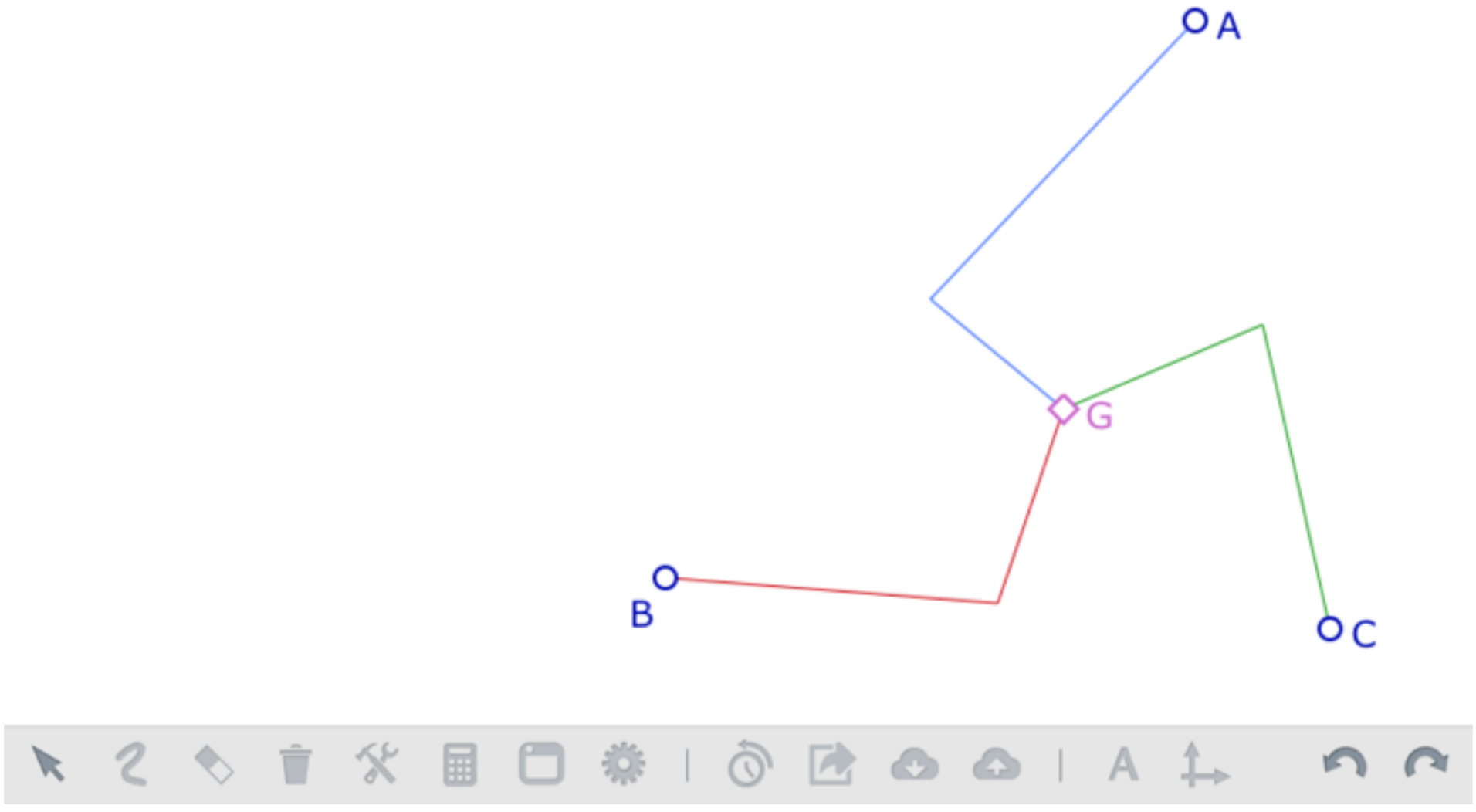
La suite du code trace les autres traces identiques issues des autres sommets.



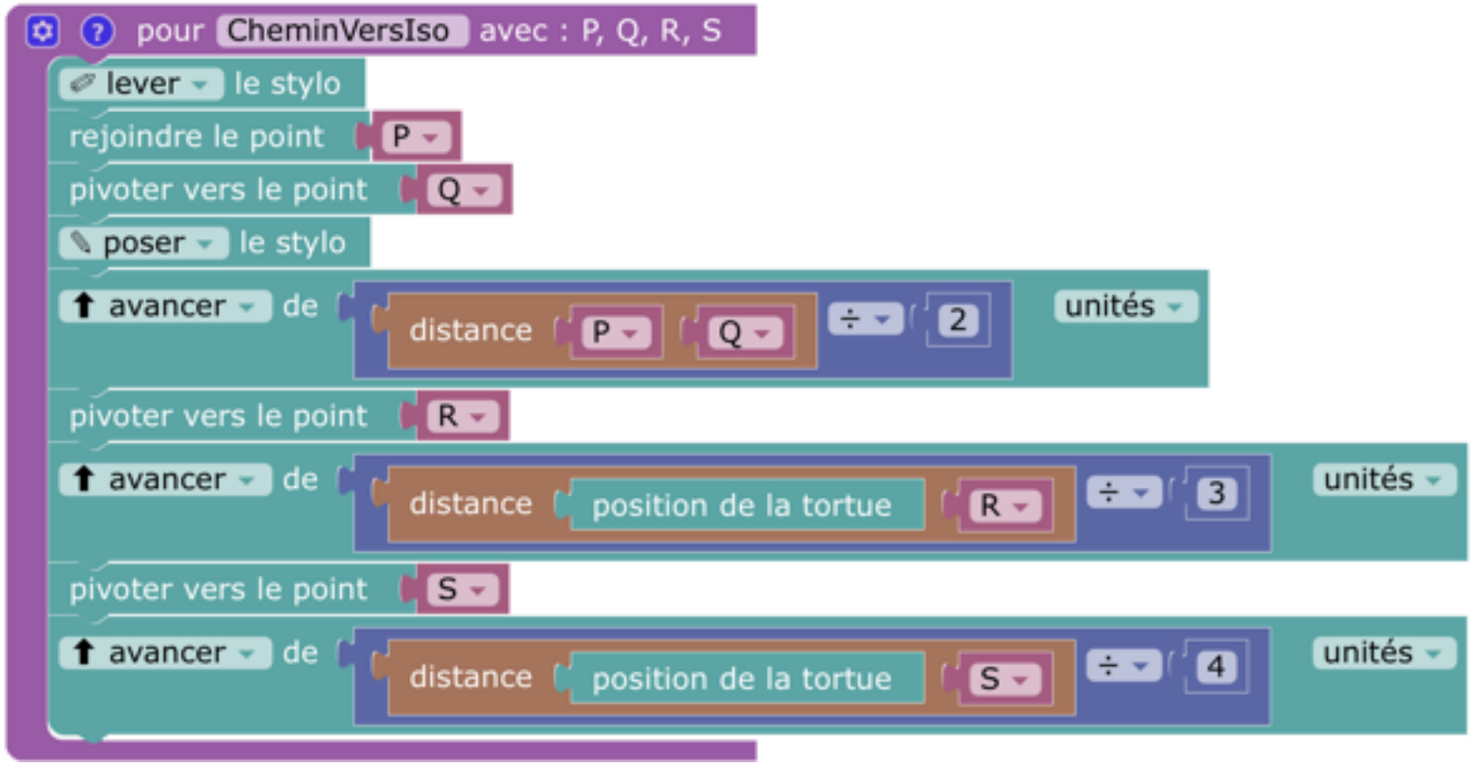


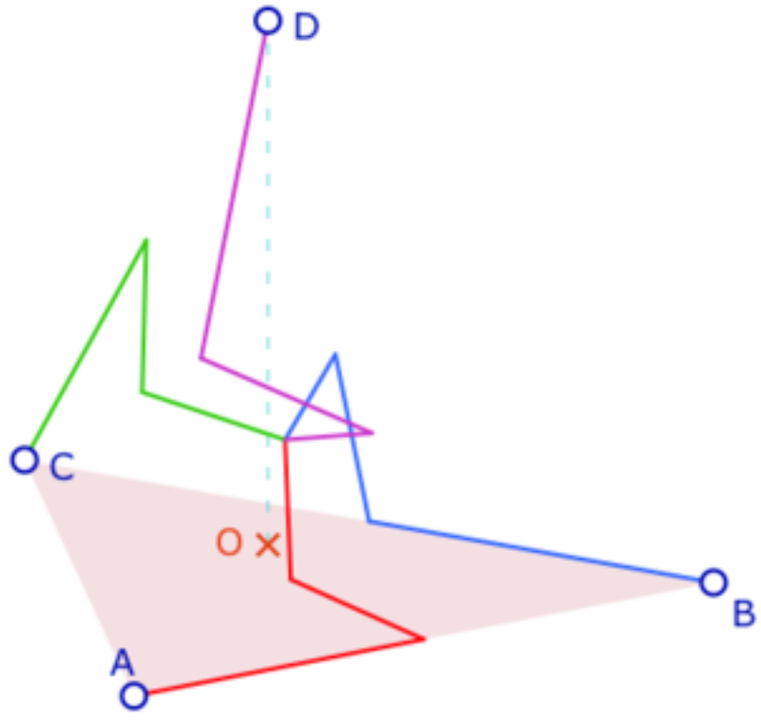
On peut faire cela en classe simplement - en cycle 4 - avec l'idée que la position de la tortue est un **autre changement de cadre** - dans le monde du code - **de la notion de variable algébrique** : on donne un nom à quelque chose dont on a besoin et dont on ne connaît pas particulièrement la valeur ou l'expression. Nous allons voir, dans les pages suivantes, que le code avec la **position de la tortue** permet différentes utilisations. On peut aussi conceptualiser un peu plus ceci avec des fonctions paramètres, ce qui est cependant moins abordable immédiatement. Dans la figure suivante, on n'a pas utilisé de fonctions : le code est plus long, mais aussi plus simple. On peut manipuler les points A, B, C.
Une conséquence importante de ce comportement : on peut aussi toujours déplacer G, car *ce n'est pas un point construit* : et en déplaçant un des sommets il reprend sa position. Cette souplesse de Blockly va avoir de **nombreux développements**.

Figure dynamique 5.1 – Traces vers le centre de gravité du triangle



Manipulation de la figure précédente. Le code Blockly est dans le point A.





Trace de la tortue de chaque sommet vers l'isobarycentre d'un tétraèdre.

On a choisi cette fois la méthode de la procédure, très efficace ici. La trace définie par P, Q, R, S va de P vers Q, jusqu'au milieu de [PQ], puis va à l'isobarycentre de la face PQR en avançant du tiers de la médiane issue de R, et termine en allant au quart de ce point au quatrième sommet S.

La procédure **CheminVersIso** utilise deux fois la position de la tortue, et donc la construction complète utilise 8 fois la position de la tortue.

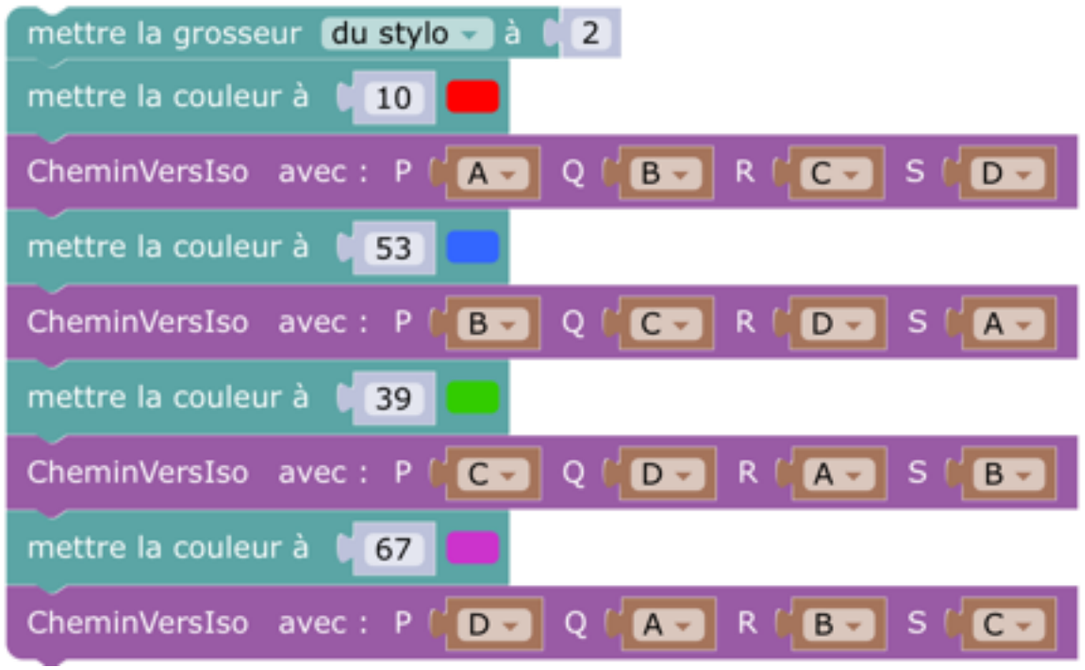
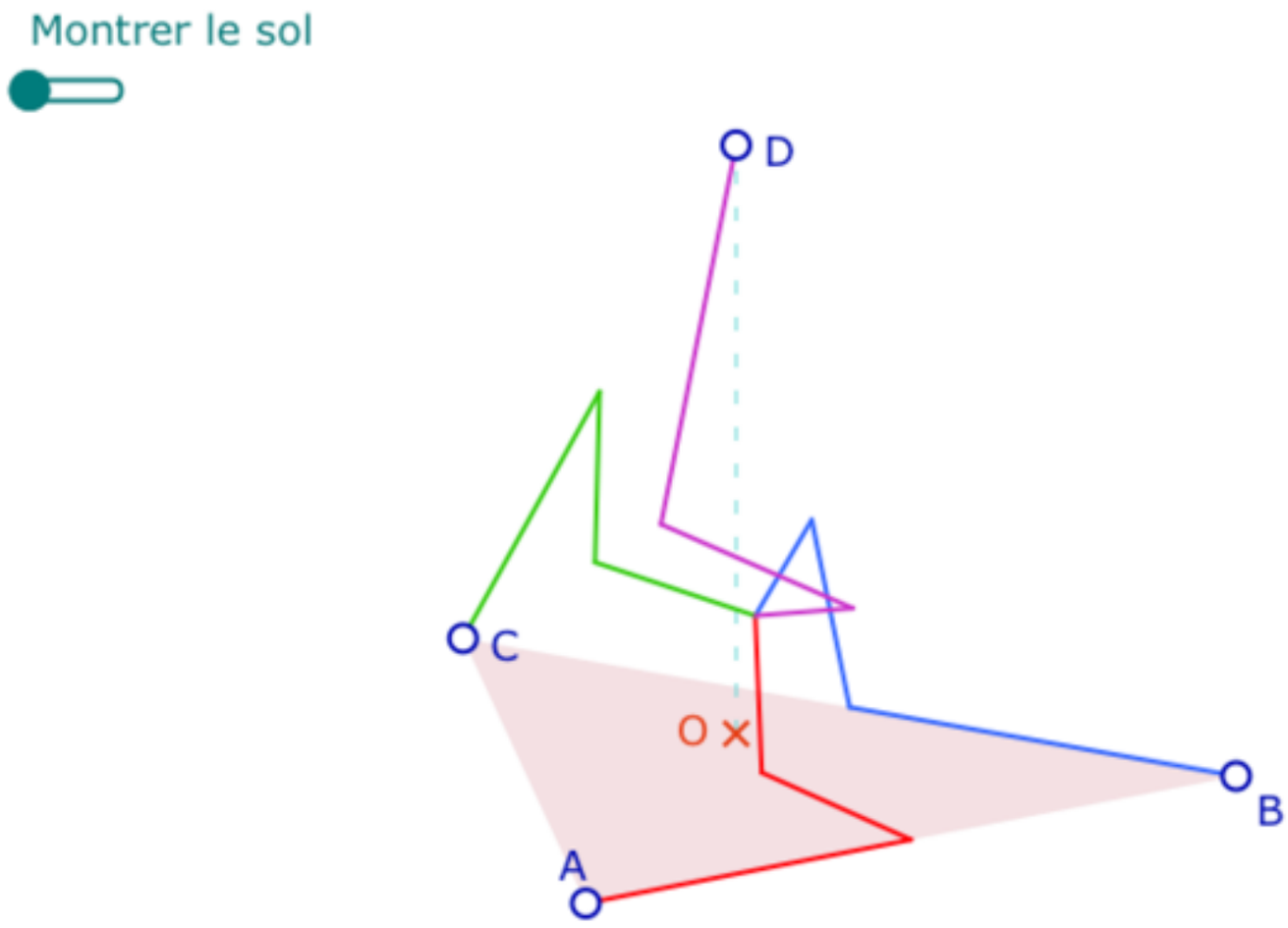


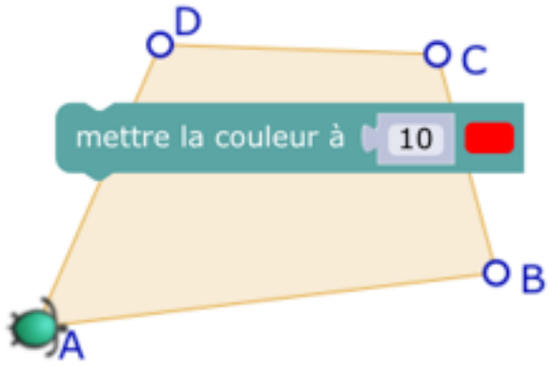
Figure dynamique 5.2 – Trace de la tortue vers l'isobarycentre d'un tétraèdre



*Mode **consultation** : déplacer les 4 sommets du tétraèdre – le trièdre se déplace au doigt sur tablette – clic gauche sur Mac.*
*Mode **standard** (pointeur de gauche activé) : le code Blockly est dans le point O. Déplacer O vers la droite.*

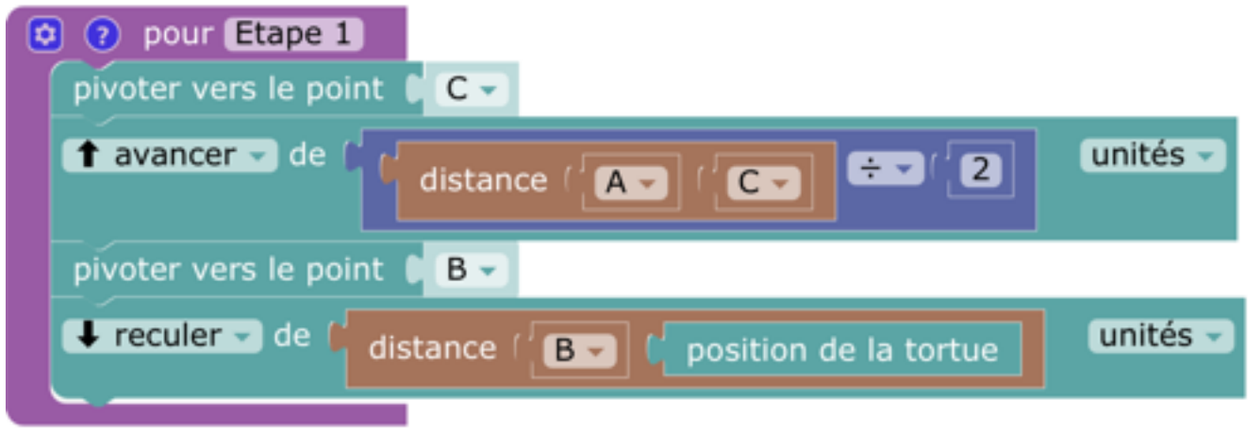
Modifier un quadrilatère à la tortue

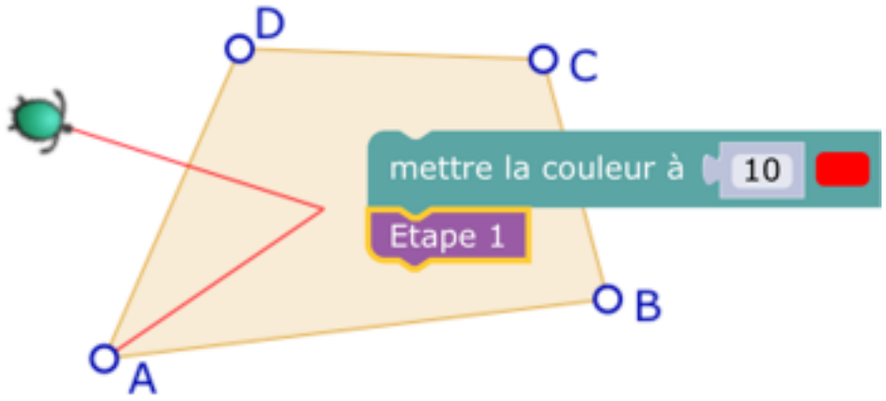
Dans cette partie, nous allons utiliser la tortue pour placer les points A, B, C, D d'un quadrilatère quelconque en un quadrilatère ayant des contraintes spécifiques : être un parallélogramme, un rectangle ou un losange ayant des contraintes données.



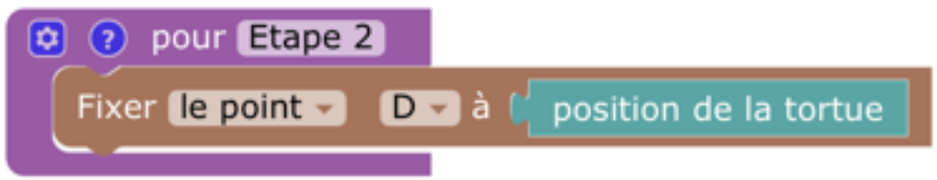
Exemple archétypique du parallélogramme

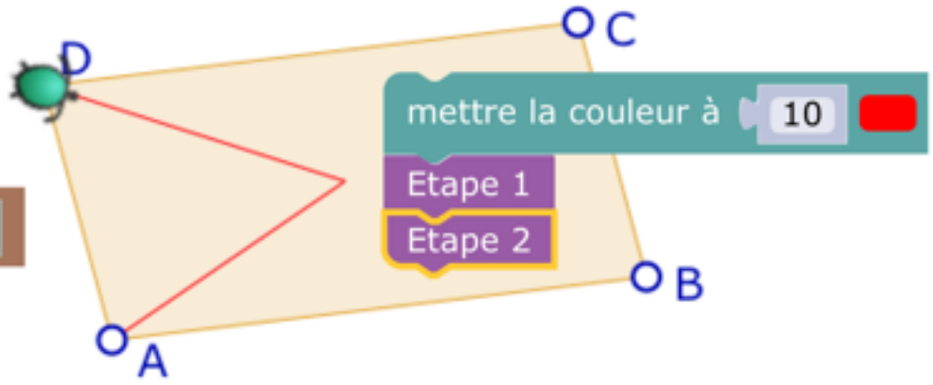
Etant donnés les trois points A, B et C. On se propose de placer D pour que ABCD soit un parallélogramme. Pour cela la tortue va au centre, puis pointe vers B et recule de la distance qui convient, par les propriétés des diagonales.



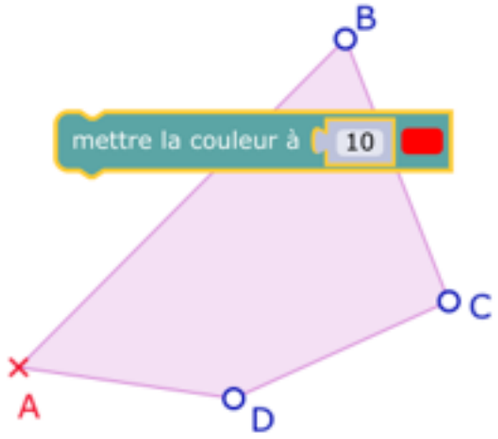


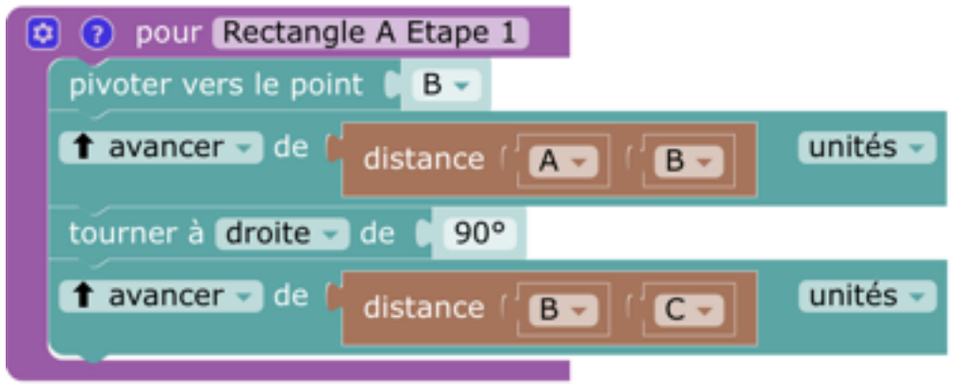
Archétypique ? C'est surtout parce que c'est un exemple **affine**.

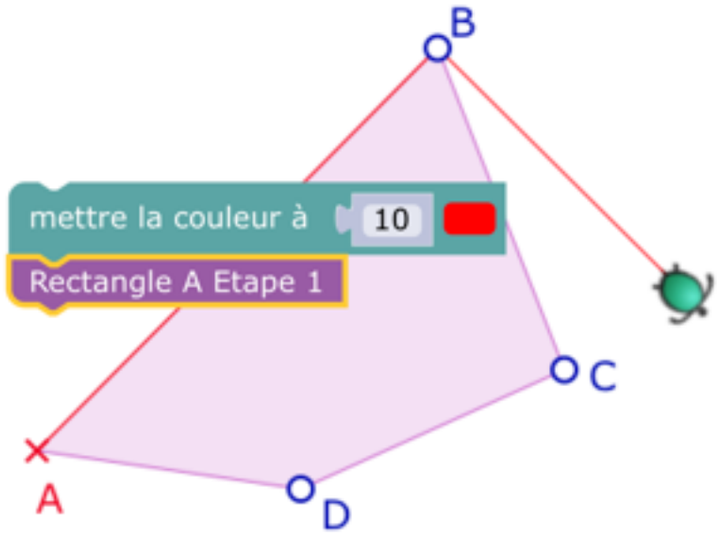




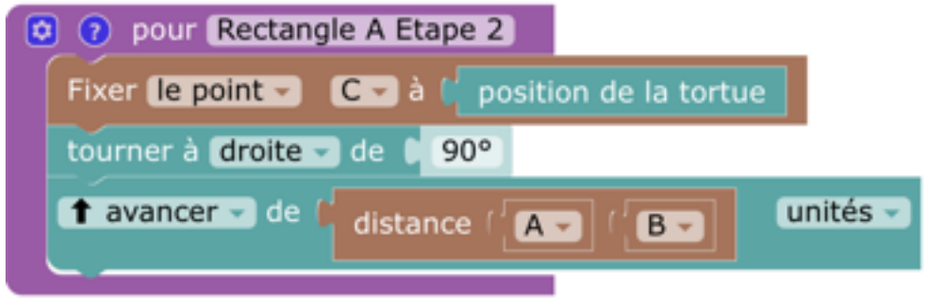
Rectangle A : dans cet exercice il s'agit de transformer ABCD en **un rectangle de côté [AB]** et dont le côté final [BC] conserve la longueur du côté [BC] initial. On voit que l'affectation de C frôle ici l'auto-référence. On verra plus loin pourquoi cela fonctionne.





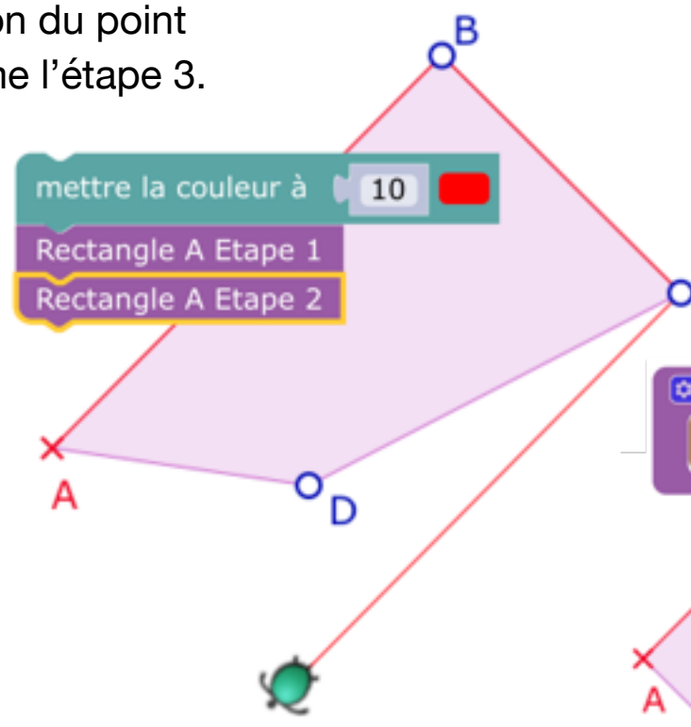


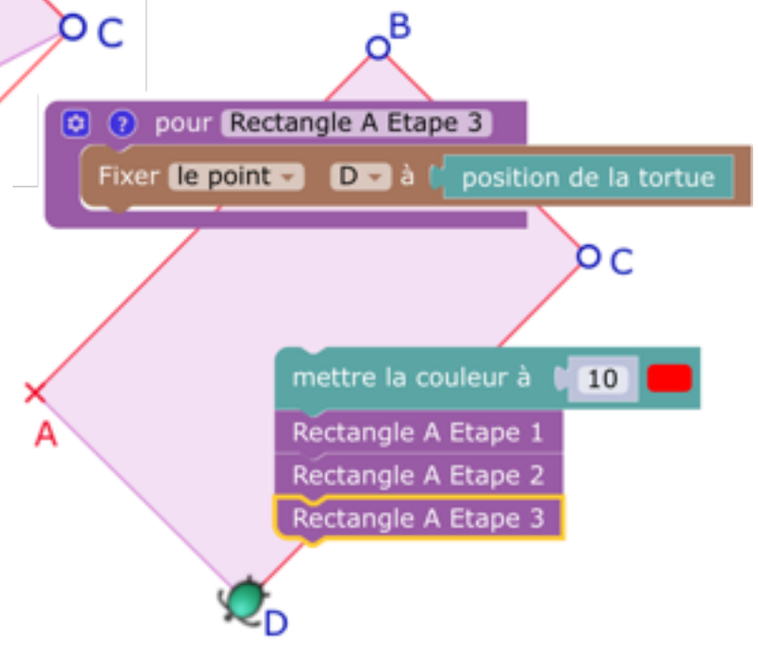
L'étape 1 est arithmétique, ne faisant appel qu'aux données initiales de la figure. Si l'étape 2 redéfinit la position du point C à partir de C, elle reste arithmétique, tout comme l'étape 3.



Quel comportement de C en manipulation directe ?

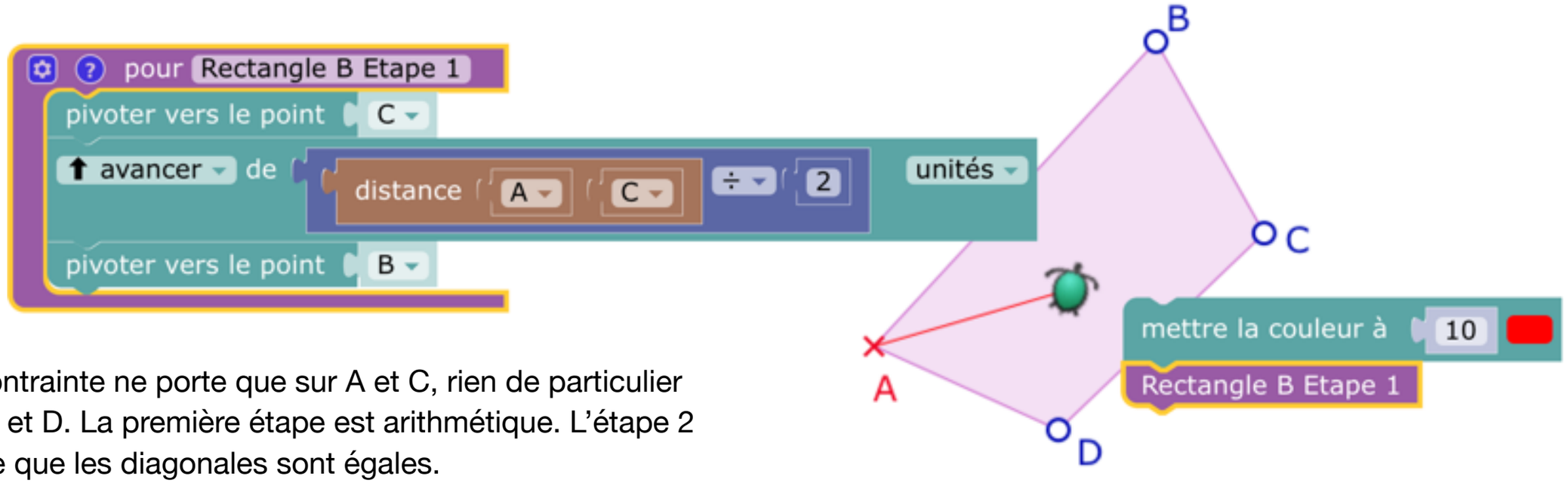
Comme l'étape 2 s'applique instantanément au déplacement de C, en pratique, C se comporte comme s'il était un point sur objet de la perpendiculaire à (AB) passant par B.



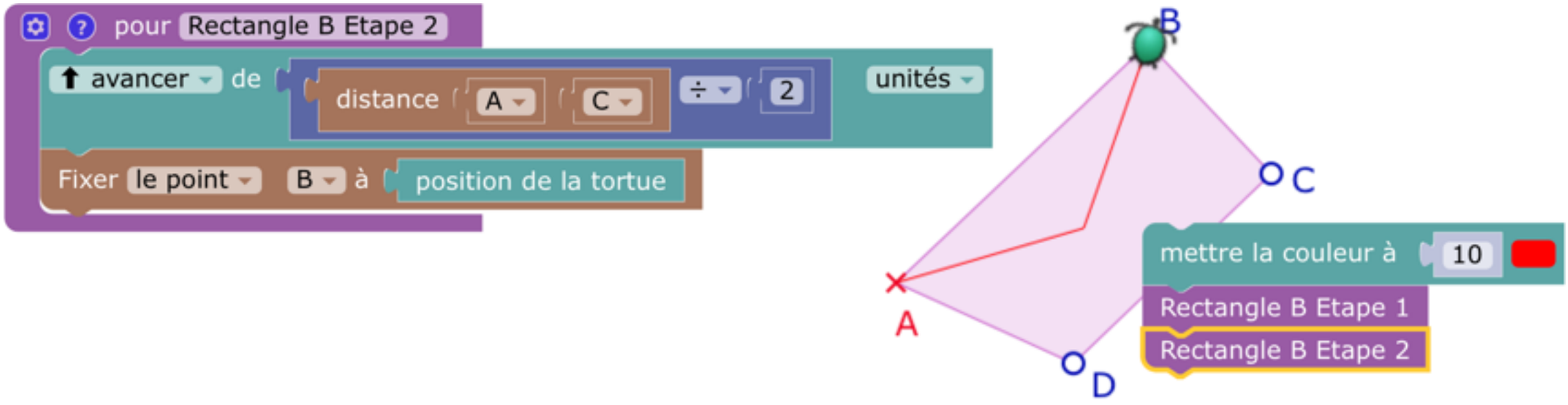


Ces figures sont manipulables dans une prochaine page.

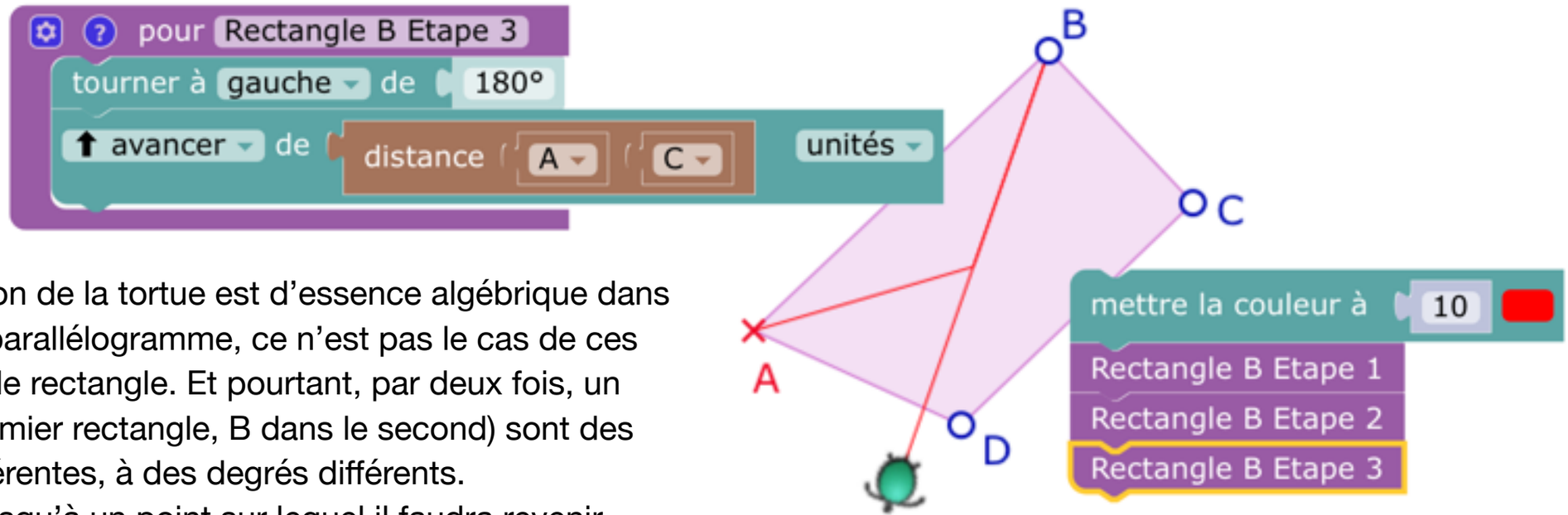
Rectangle B : Cette fois-ci, on travaille sur les diagonales du rectangle pour transformer ABCD en **un rectangle de diagonale [AC]**.



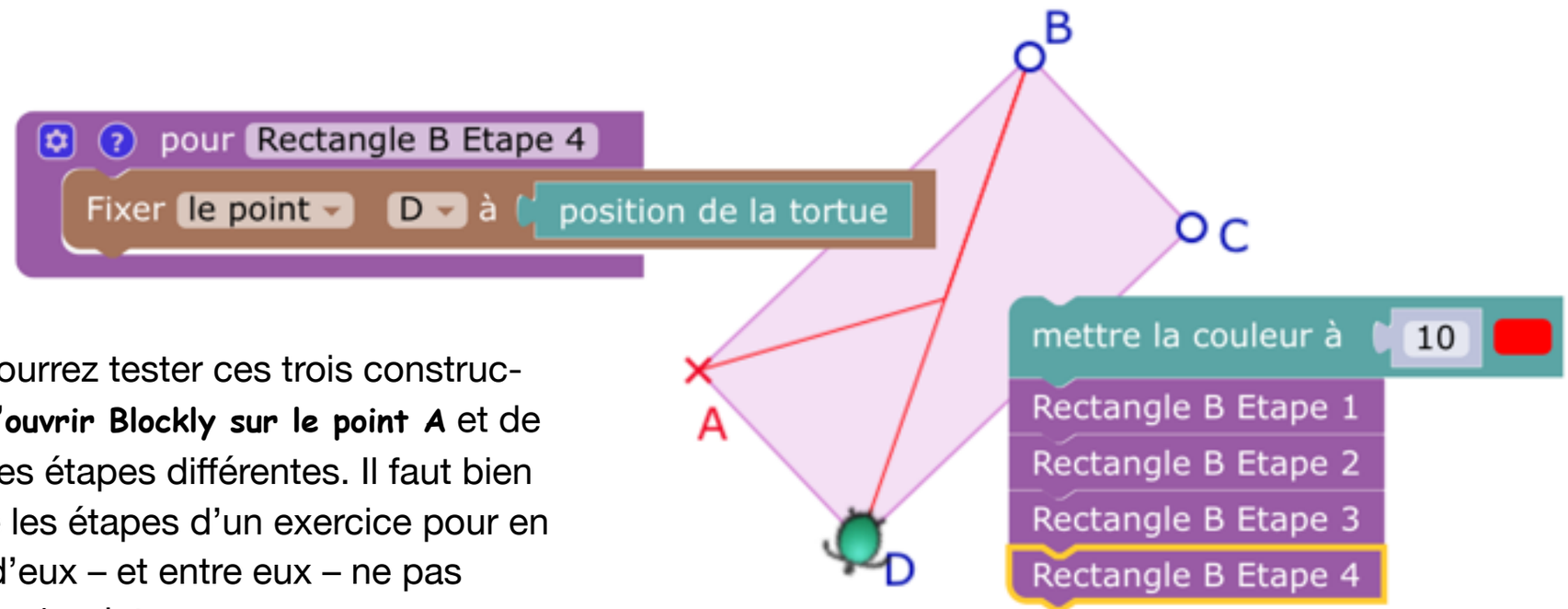
La contrainte ne porte que sur A et C, rien de particulier sur B et D. La première étape est arithmétique. L'étape 2 utilise que les diagonales sont égales.
Et donc l'auto-référence de B ne porte que sur l'orientation de la tortue. Elle est ainsi plus faible que dans l'exemple du rectangle A. C'est toujours une écriture arithmétique.



Rectangle B (suite). On veut transformer ABCD en **un rectangle de diagonale [AC]**.

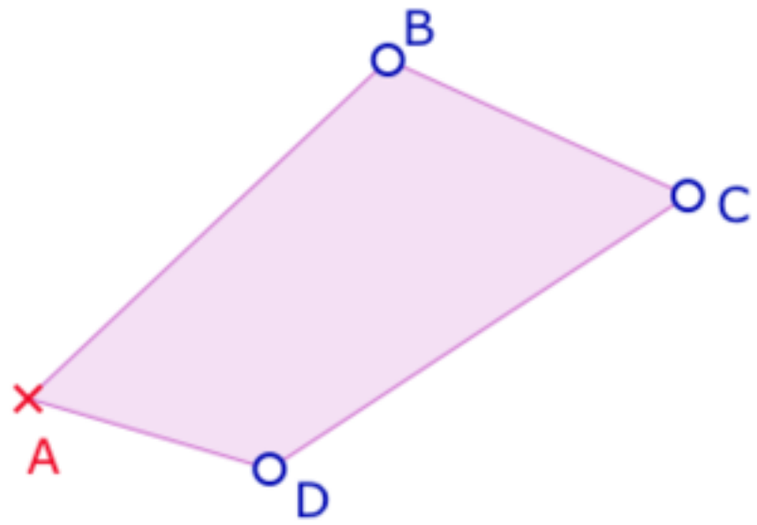


Au final, si l'utilisation de la tortue est d'essence algébrique dans la construction du parallélogramme, ce n'est pas le cas de ces deux exercices sur le rectangle. Et pourtant, par deux fois, un point (C dans le premier rectangle, B dans le second) sont des définitions auto-référentes, à des degrés différents.
Cela montre que, jusqu'à un point sur lequel il faudra revenir, **Blockly permet une certaine auto-référence des points**, et cela dans des écritures qui semblent arithmétiques.



Dans la page suivante, vous pourrez tester ces trois constructions. Pour cela il vous suffit d'**ouvrir Blockly sur le point A** et de glisser sur l'espace de travail les étapes différentes. Il faut bien entendu remettre à la corbeille les étapes d'un exercice pour en tester un autre. Dans chacun d'eux – et entre eux – ne pas oublier de déplacer chacun des 4 points.

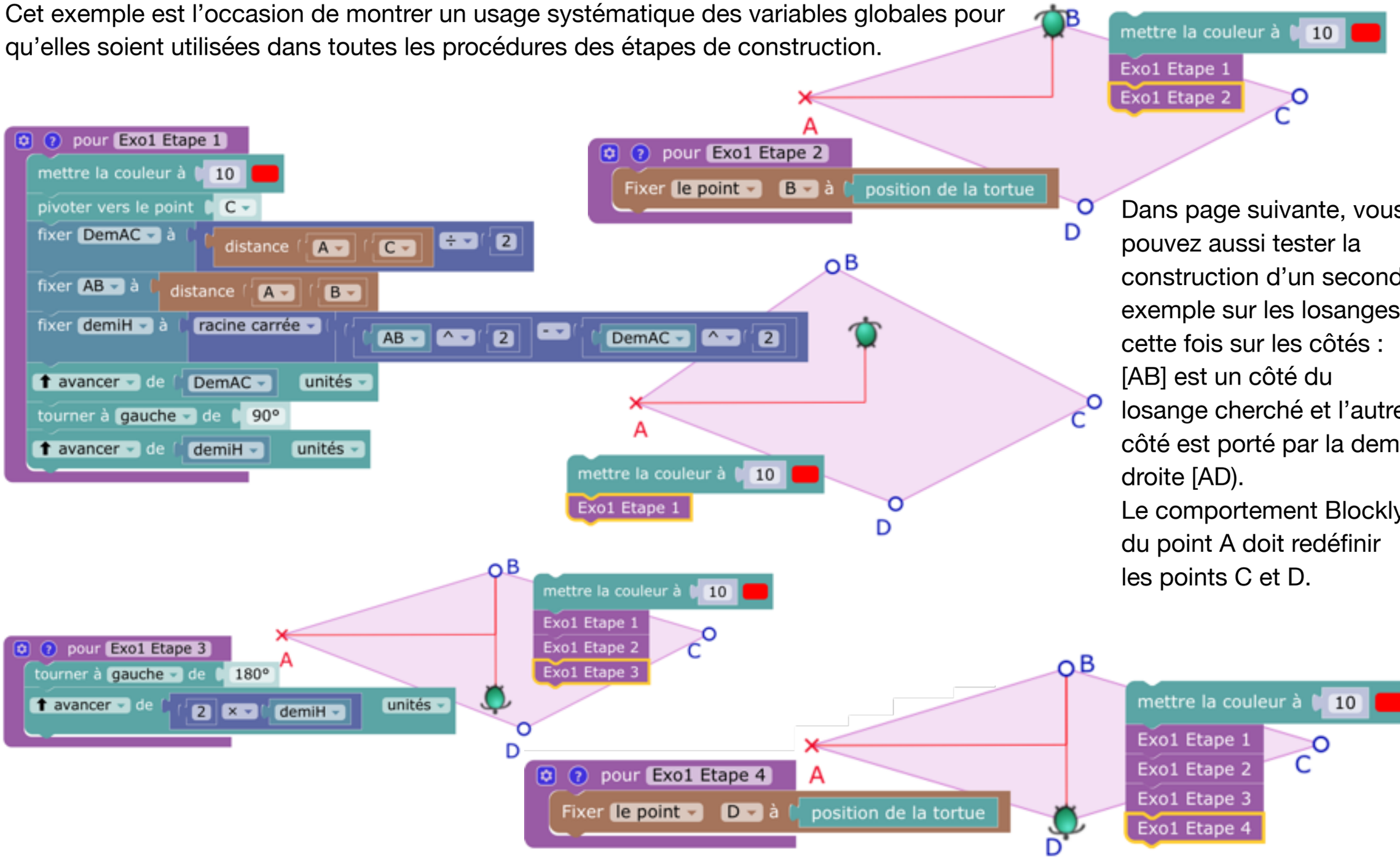
Figure dynamique 5.3 – Tester des trois exercices précédents. *Le code Blockly est dans le point A*



*Les 9 étapes des 3 exercices sont dans la rubrique **Fonctions**. On les glisse sous le bloc couleur d'initialisation.*
*Ne pas oublier qu'il faut être en mode **standard** (pointeur de gauche activé) pour activer Blockly.*

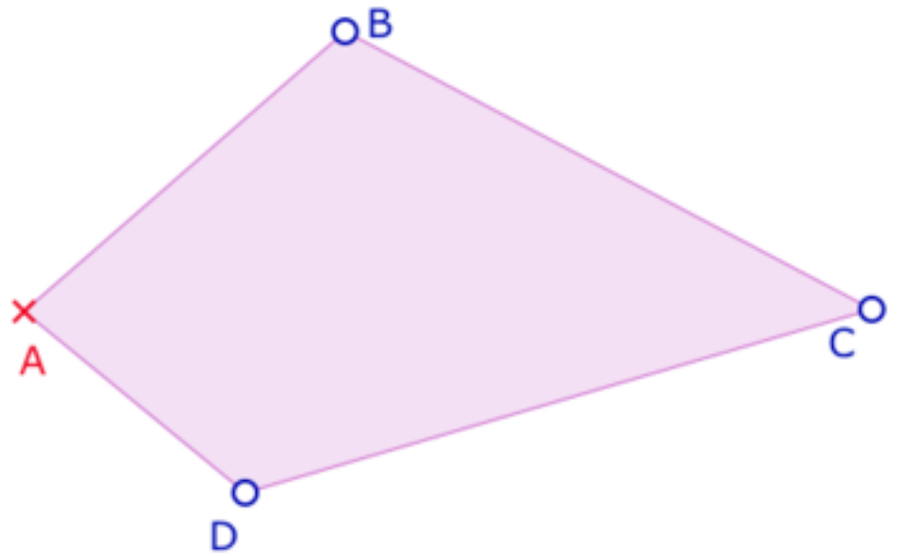
Losange A - Poursuivons cette exploration de transformation d'un quadrilatère ABCD en une figure géométrique particulière. Cette fois-ci, on s'intéresse à réaliser **un losange de diagonale [AC]** et dont **le côté est de longueur AB**, celle du côté initial [AB]. Il s'agit donc de redéfinir B, utilisé dans la définition du côté, et D.

Cet exemple est l'occasion de montrer un usage systématique des variables globales pour qu'elles soient utilisées dans toutes les procédures des étapes de construction.



Dans page suivante, vous pouvez aussi tester la construction d'un second exemple sur les losanges, cette fois sur les côtés : [AB] est un côté du losange cherché et l'autre côté est porté par la demi-droite [AD).
Le comportement Blockly du point A doit redéfinir les points C et D.

Figure dynamique 5.4 – Modifier un quadrilatère – Cas du losange – deux exemples

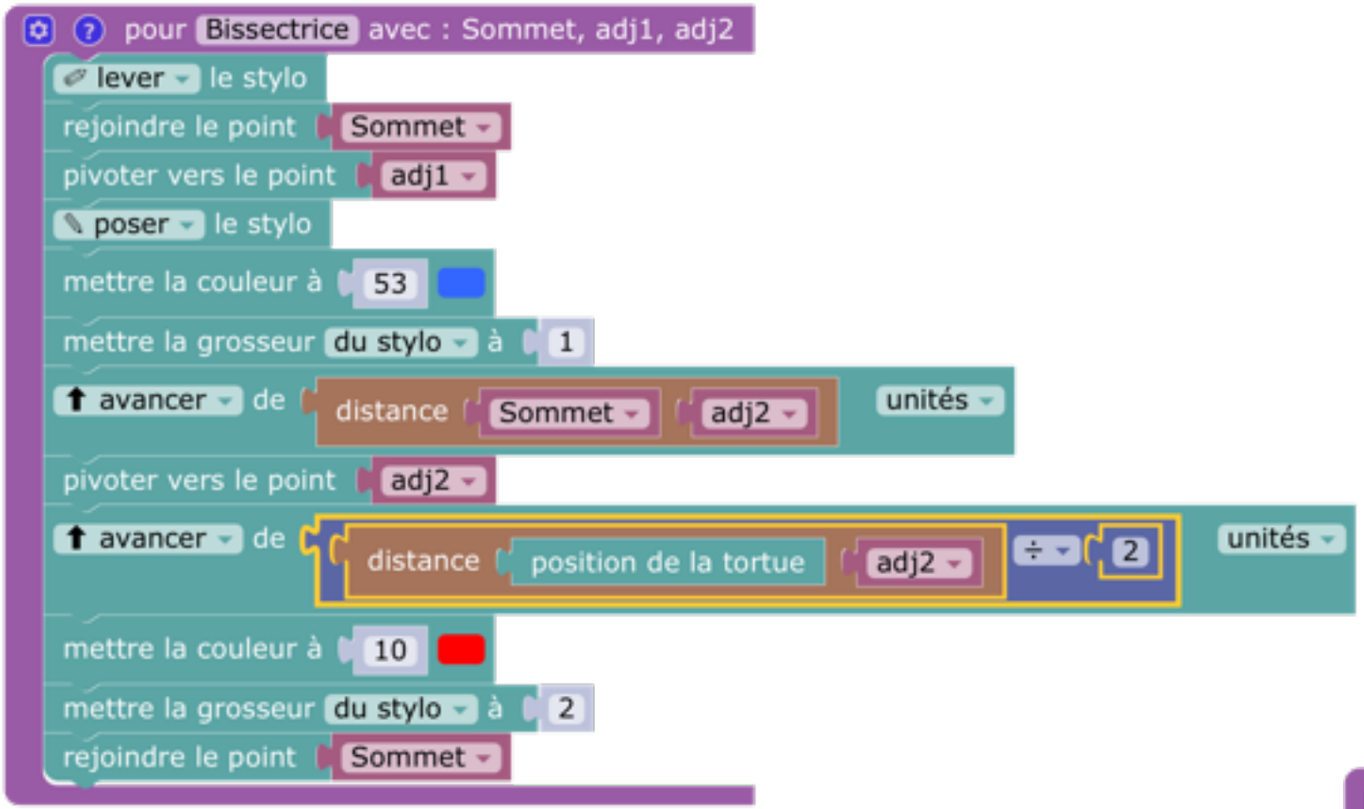


*Rappels : le code est en A. Ne pas oublier de basculer parfois du mode **standard** (Blockly) au mode **consultation** si nécessaire.*
Ne pas hésiter à modifier tous les points, en particulier ceux redéfinis par Blockly.

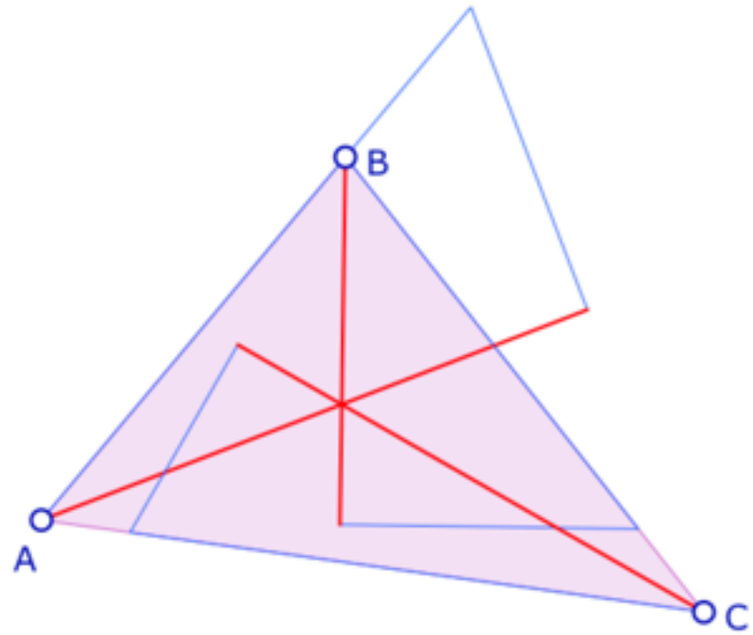
Revisiter quelques théorèmes avec la « position de la tortue »

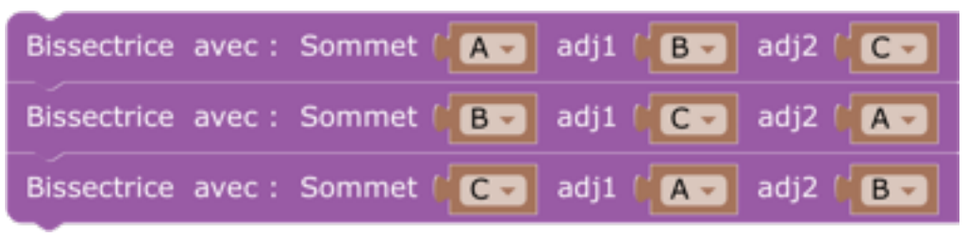
1. Les bissectrices d'un triangle comme hauteurs de triangles isocèles associés

Une façon de construire simplement la bissectrice d'un angle - sans utiliser les angles autres que droits, c'est-à-dire une équerre - consiste à utiliser un triangle isocèle annexe pour chaque angle.



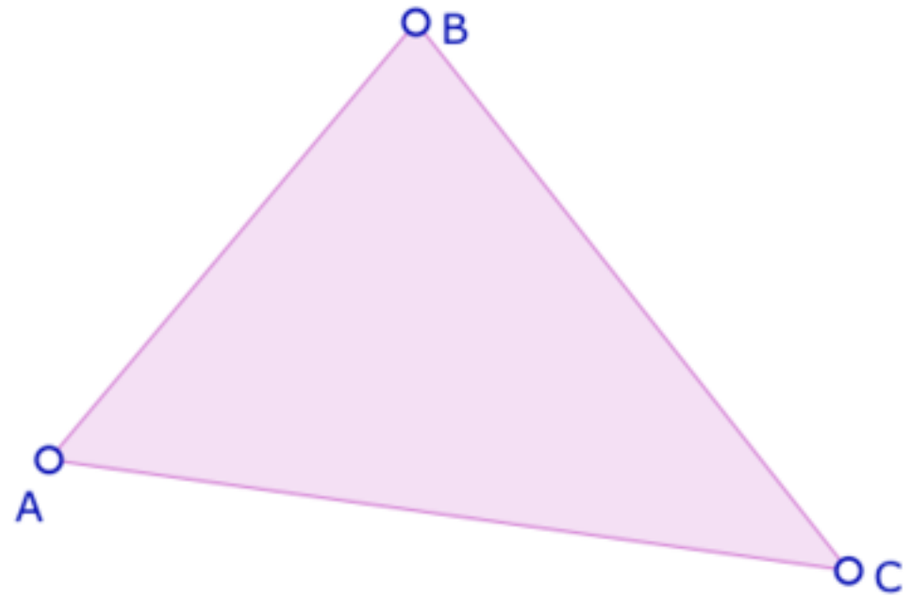
Pour construire la bissectrice issue de A, on va construire sur ABC un **triangle isocèle** de côté AC et prendre sa hauteur : partant de A, dans la direction de B, la tortue avance de AC. On pointe alors vers C, et en avançant de la moitié (partie surlignée dans le code) on est au pied de la hauteur d'un triangle qui est isocèle : il suffit de tourner à angle droit et de rejoindre A pour tracer la bissectrice sans utiliser les angles.





Dans la page suivante, la procédure est déjà écrite. Vous êtes invité à l'appliquer 3 fois avec des paramètres différents pour construire les 3 bissectrices.

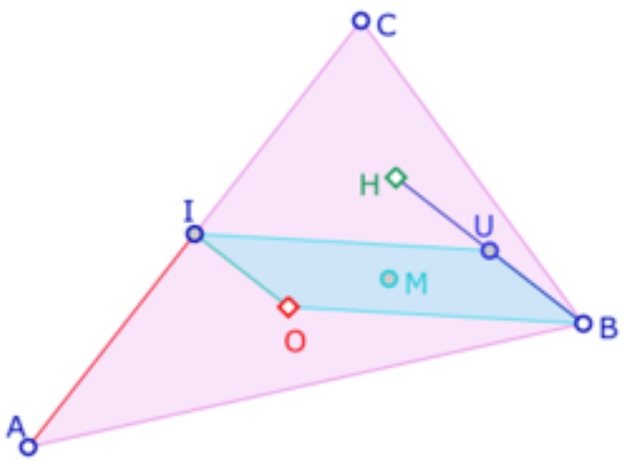
Figure dynamique 5.5 – Les bissectrices d'un triangle sans usage des angles autres que droits



Le code est en A. Penser à un touché long pour mettre les paramètres des fonctions en ligne, cela prend moins de place.

2. Trace vers l'orthocentre (hors contexte scolaire) - Droite d'Euler

Les relations d'Euler dans le triangle sont l'occasion de revisiter un chemin vers l'orthocentre pour la tortue et la mise en place de la droite d'Euler. Pour cela il faut néanmoins placer le centre du cercle circonscrit, ce que nous allons aussi faire avec la tortue.



I est le milieu de A et C, O est le centre du cercle circonscrit. Si U est le milieu de [BH], outre qu'il est sur le cercle d'Euler, on sait que $\overrightarrow{BH} = 2\overrightarrow{OI}$, et donc que BUIO est un parallélogramme de centre M milieu de {IB]. Si on sait construire le centre O, alors on a une trace de la tortue qui peut aller d'un sommet à l'orthocentre - avec l'usage du parallélogramme par « position de la tortue » - peut être le trajet AIMUH.

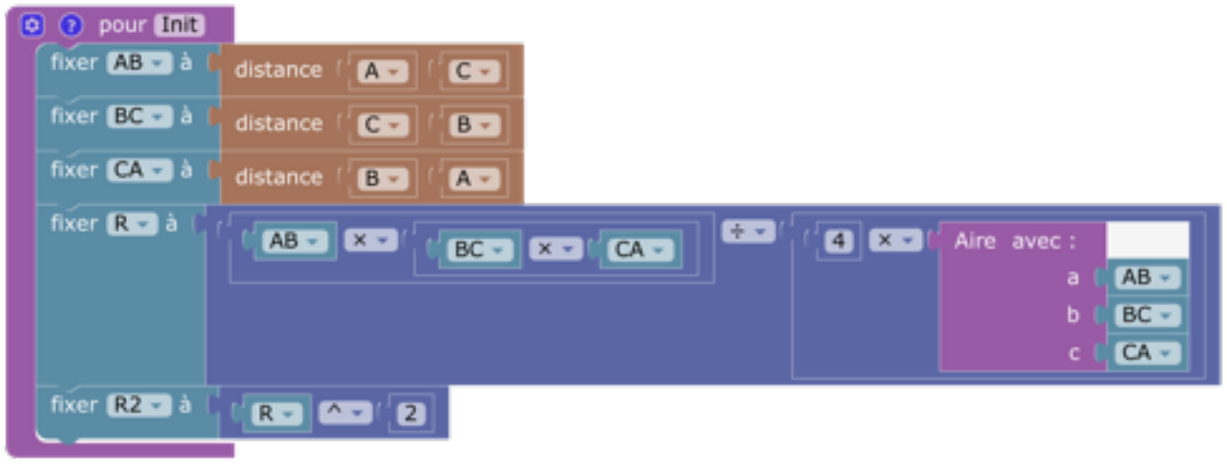
On va utiliser le point O pour pointer vers lui, il faut qu'il existe comme point de la figure. Il est construit avant, comme le point G dans un exemple précédent,

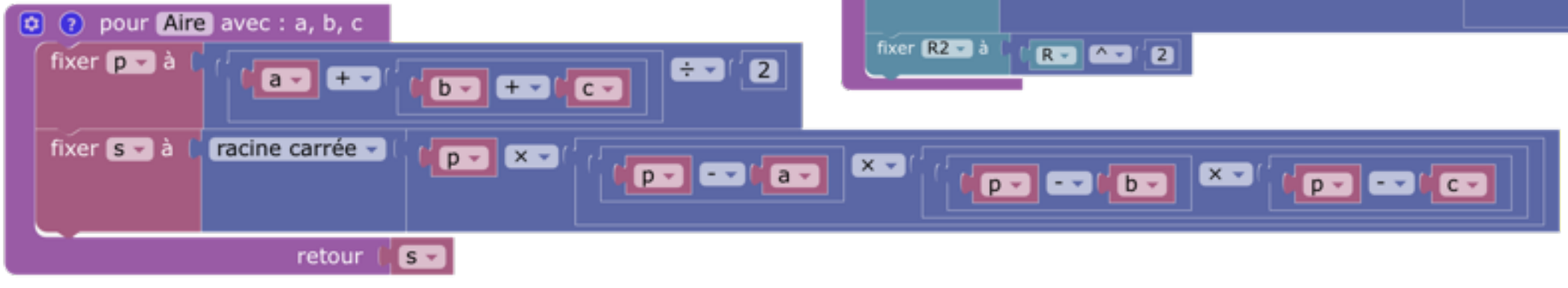
Centre du cercle circonscrit *(de rayon R, pour un triangle de côtés a, b, et c)*

On suppose connaître la formule de Héron $S = \sqrt{p(p-a)(p-b)(p-c)}$ où $p = \dfrac{a+b+c}{2}$, et la relation $R = \dfrac{abc}{4S}$

On commence donc par une procédure **Aire** et une autre, **Init** qui fixe les variables (globales) et le rayon **R**.

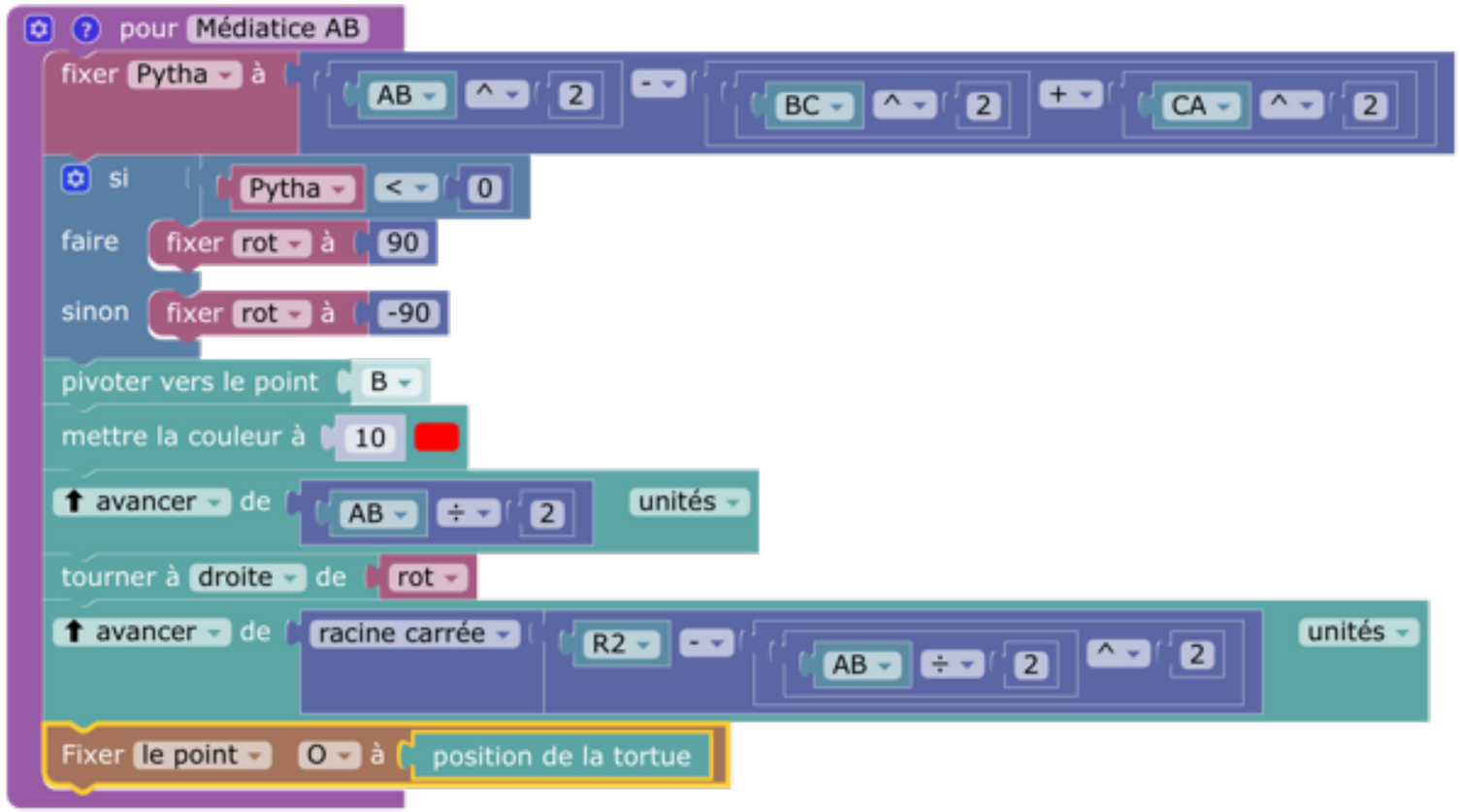
Une procédure va ensuite **placer O** en testant s'il est à l'intérieur où à l'extérieur du triangle.

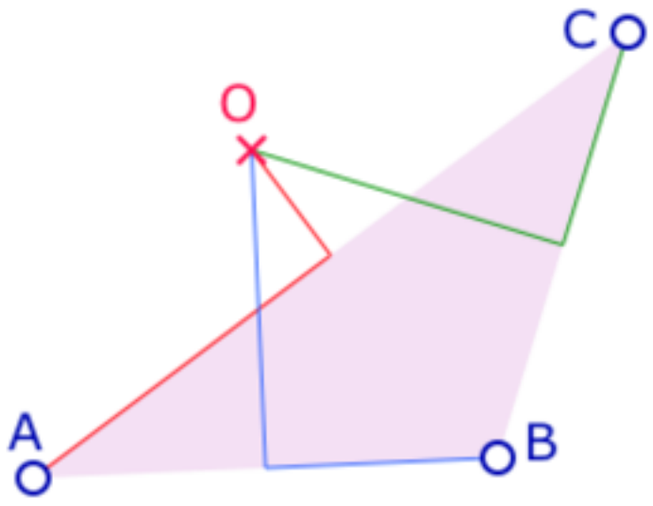




Placer le centre du cercle circonscrit *O*

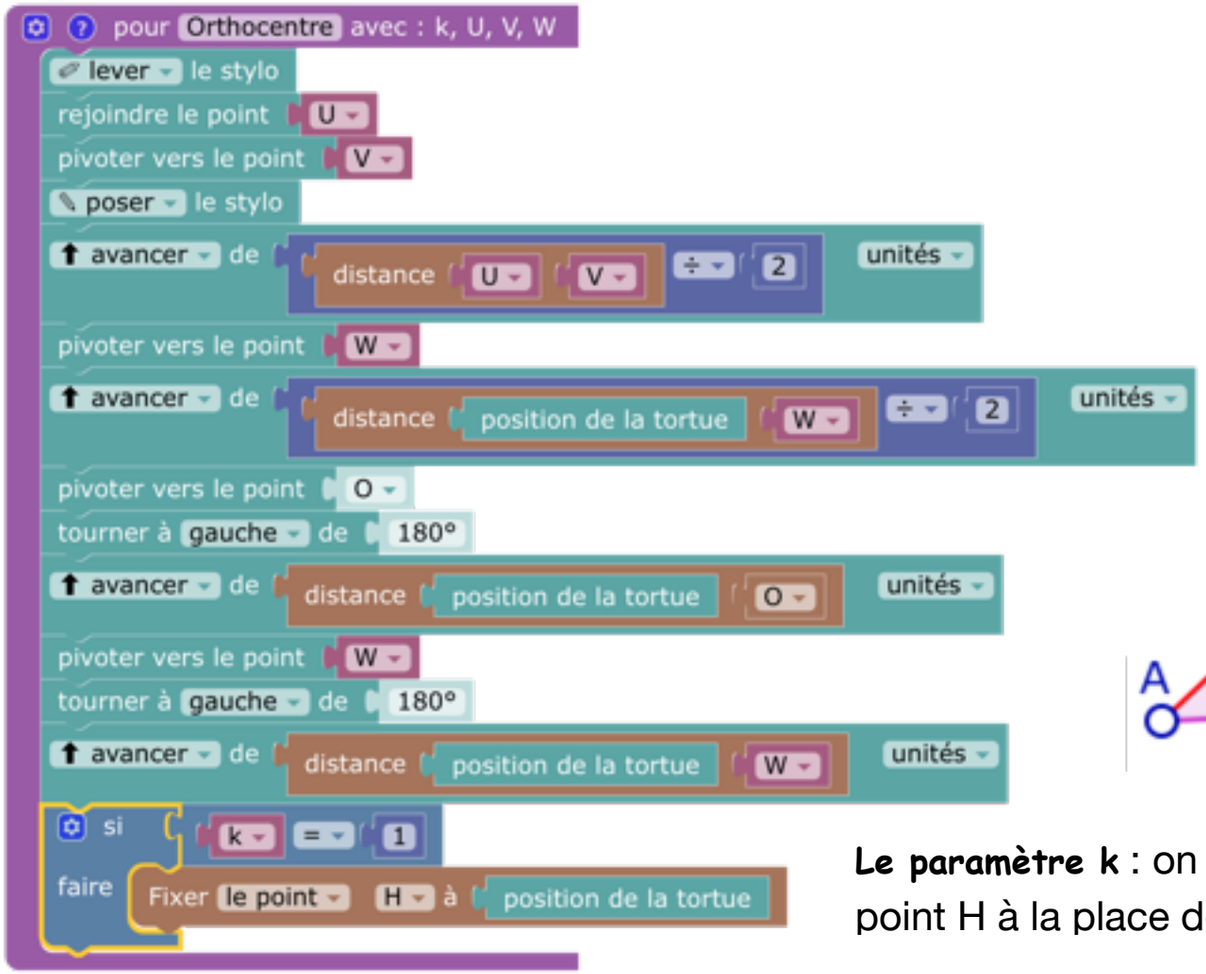
Là encore, on voit que le changement de cadre de la tortue et du code, invite à chercher des critères métriques (*) pour savoir si la tortue, arrivée au milieu d'un segment, doit tourner vers l'intérieur ou l'extérieur du triangle pour placer O.
On sait que le point O est au milieu de [AB] ssi $AB^2 - BC^2 - CA^2$ est nul (car AB est hypothènuse). Comme O est sur la médiatrice de [AB], c'est le signe de cette expression (**Pytha** ci-contre) qui précise la position de O. Si elle est négative, le centre du cercle est à l'intérieur du triangle, sinon il est à l'extérieur. C'est l'objet de la procédure ci-contre pour placer O.

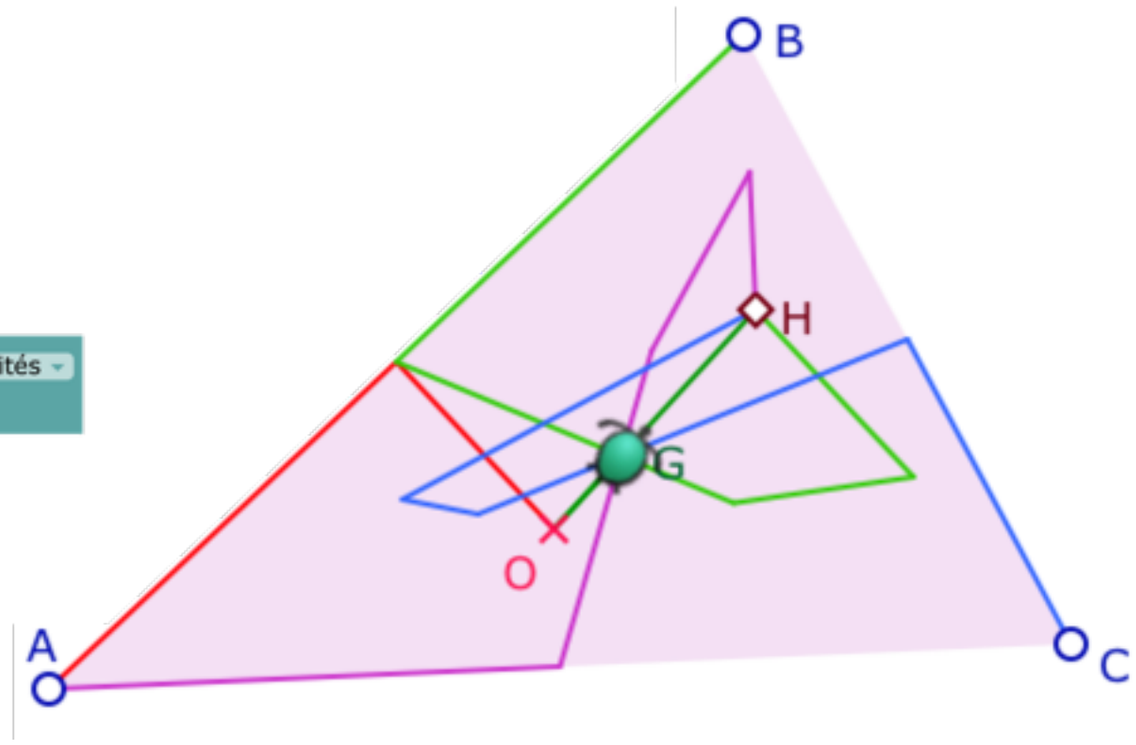




(*) En fait cette figure a été faite avant que le bloc **Angle** *existe dans DGPad, qui est d'un ajout récent. Bien entendu on pourrait préférer le critère angulaire, mais cela ne simplifierait pas le code de manière significative. Et donc sans cette absence, on n'aurait pas cherché ce critère métrique, intéressant.*

Nous pouvons donc, enfin, faire une trace de tortue pour aller d'un sommet à l'orthocentre d'un triangle, et poursuivre sur la droite d'Euler.

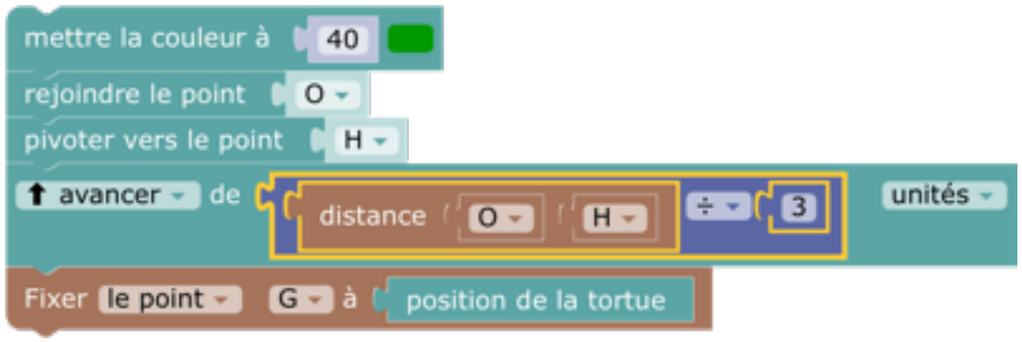




Le paramètre k : on va appliquer la procédure 3 fois, mais il ne faut placer le point H à la place de l'orthocentre qu'une fois. On a choisit de le faire si k = 1.

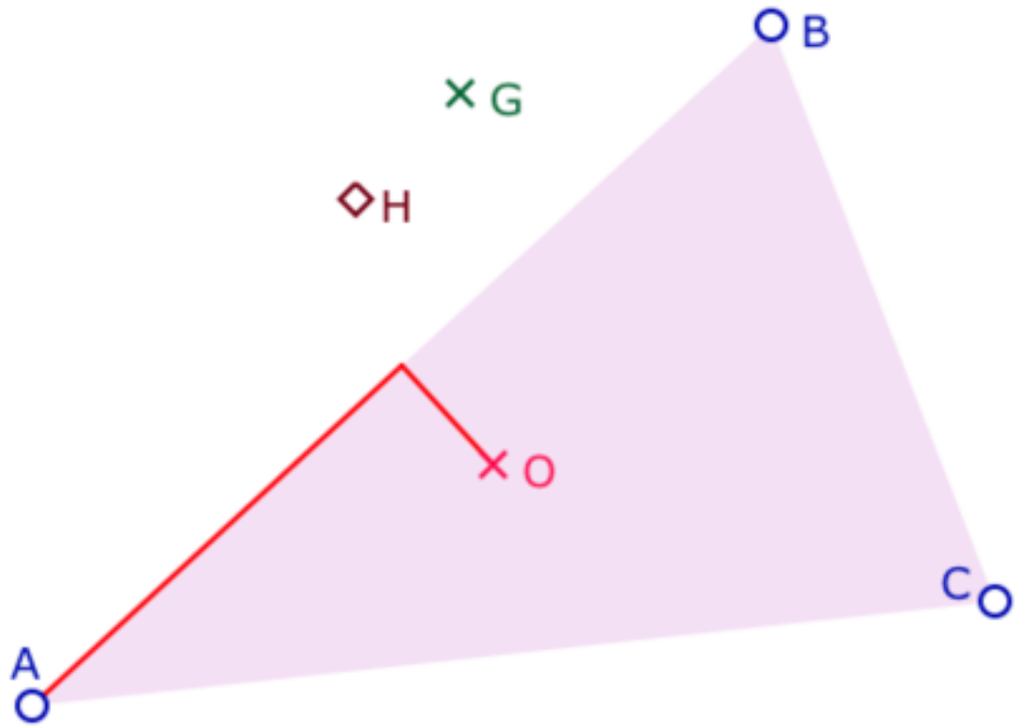
La trace orthocentrique : elle reprend ce qui a été dit sur la relation d'Euler. Détaillons le trajet issu de A avec les paramètres A, B, et C. On commence par avancer de la moitié de [AC], puis de la moitié de la position de la tortue au 3° sommet B (et donc on passe par G). Ensuite, on pointe vers le centre du cercle circonscrit et on recule de la distance séparant la tortue à ce point. On réitère ensuite avec le sommet pour arriver à l'orthocentre. Ainsi **cette trace utilise 3 fois la position de la tortue de manière algébrique**.

La droite d'Euler : il s'agit de placer ensuite G et de construire [OH] par exemple comme ceci :



La page suivante propose ces constructions, les procédures vues dans ces pages étant disponibles.

Figure dynamique 5.6 – Aspect algébrique de la tortue – Orthocentre et droite d'Euler à la tortue



*Sous les deux procédures **Init** et **MédiatriceAB**, appliquer 3 fois **Orthocentre**, dont une fois avec k=1 pour placer H.*
Puis ajouter du code (vu à la page précédente) pour placer G et tracer la «droite d'Euler» (trace de la tortue).

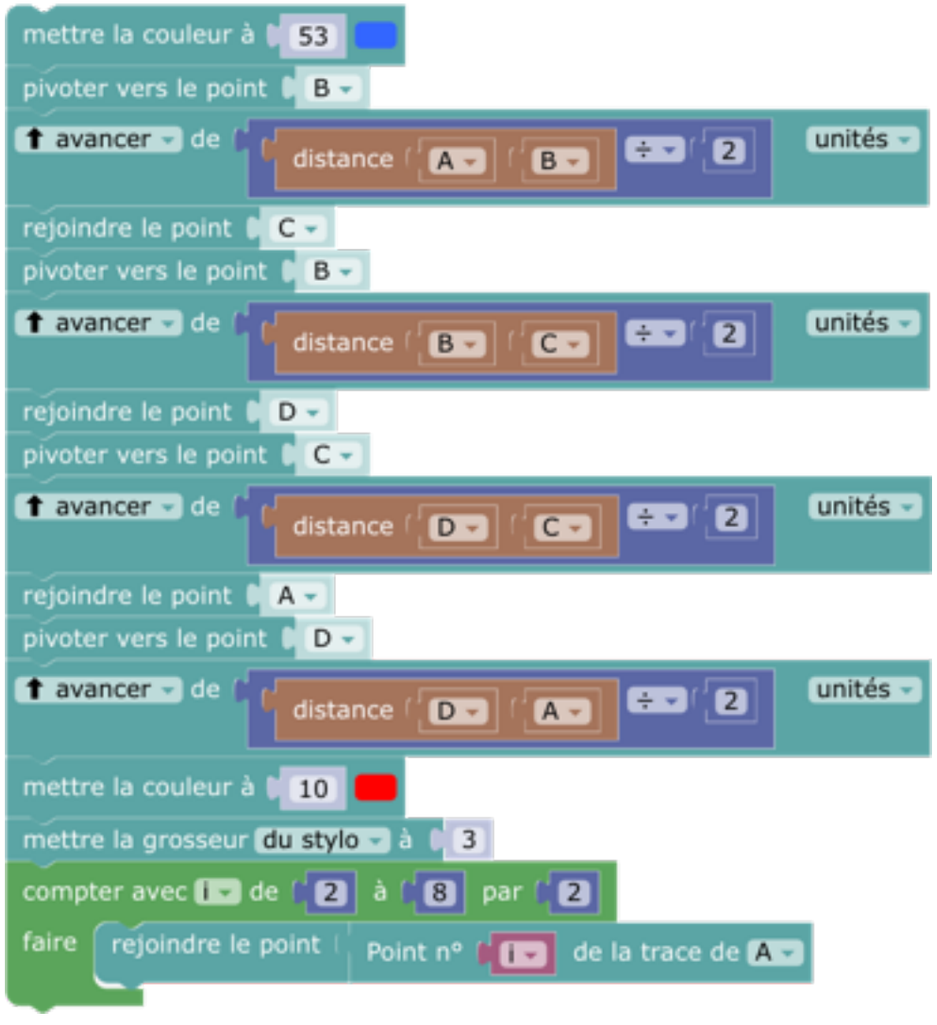
Démarche affine et démarche euclidienne

On pourrait penser que la tortue n'est qu'euclidienne, puisque l'on avance toujours d'une longueur donnée : il n'y a finalement rien d'affine avec une tortue. Pourtant on a vu dans les exemples précédents – sur le parallélogramme ou certaines parties de la trace vers l'orthocentre - que **certaines avancées de la tortue peuvent être considérées comme affines** quand elles traduisent des relations barycentriques. Par exemple, la prise d'une longueur initiale entre deux points peut être considérée comme une définition d'un repère affine sur une droite. De même, le bloc qui permet de pivoter la tortue vers un point n'a pas à être considéré comme un bloc euclidien, il s' agit de l'utilisation d'une règle, pour joindre deux points : la droite est un outil affine.

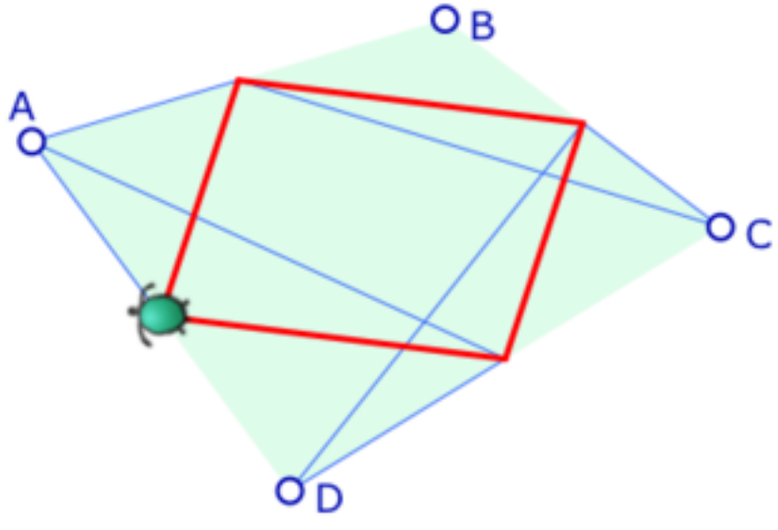
Utilisation du théorème de Varignon.
Version affine

On se propose de tracer le quadrilatère des milieux d'un quadrilatère, sachant que c'est toujours un parallélogramme.

Dans cette trace, on utilise la propriété des milieux.



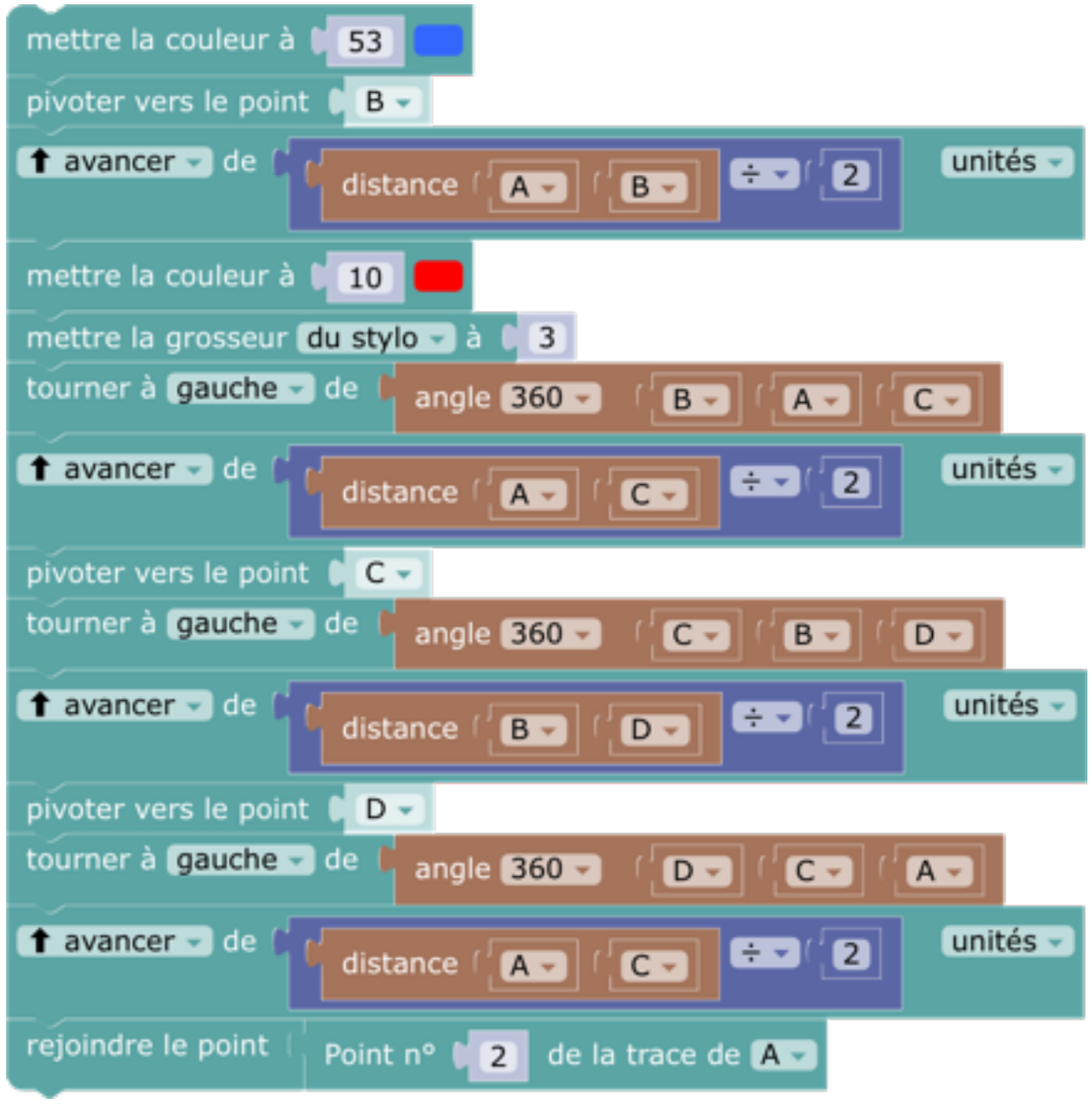
Dans cette construction, les sommets du parallélogramme de Varignon sont un point sur deux de la trace de la tortue. D'où la construction finale avec un compteur qui parcourt les points d'indices pairs



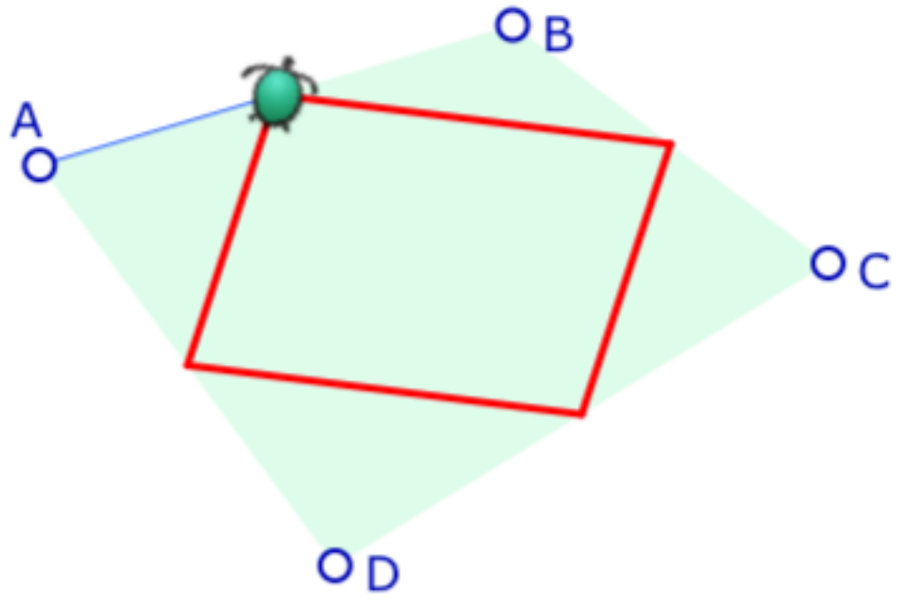
Utilisation du théorème de Varignon – Version euclidienne

Dans la construction précédente, on ne pouvait pas donner une direction particulière à la tortue pour parcourir les droites des milieux des différents triangles. Pour le faire, il faut utiliser un angle : c'est là la frontière entre une démarche affine et une démarche euclidienne. La trace est beaucoup plus courte, même si le code n'est pas significativement plus court.

Dans cette construction, les angles nous permettent de construire les parallèles aux diagonales [AC] et [BD], ce qui nécessite donc de pointer vers les sommets du quadrilatère.



Même si la question de la distinction entre affine et euclidien n'est pas à l'ordre du jour des programmes scolaires, cette question peut être intéressante à être soulevée lors de formations d'enseignants.



Exemples d’utilisations spécifiques

On regroupe dans ce chapitre quelques exemples d’utilisation en 2D qui utilisent une technique particulière, qui n’a pas été et ne sera pas détaillée dans un autre chapitre.

Trois parties dans ce chapitre :
a - la référence à la construction antérieure de la trace de la tortue en cours de sa propre réalisation
b - un exemple d’utilisation géométrique de l’aléatoire
c - deux figures de pavage du plan
d - quelques figures de Eric Hakenholz sur les décimaux et les fractions.

La dernière section de ce chapitre est l’occasion de manipuler quelques figures de l’auteur de DGPad lui-même, figures construites pour ses classes de 5° ou 4°. Elles sont d’un tout autre style que ce que nous présentons ici, souvent techniquement soit complexes, soit subtiles à réaliser. Ces figures sont fournies en mode consultable (avec le tableau de bord). Le lecteur courageux fouillera dans le code des DGScripts qui contiennent souvent l’essentiel de la programmation. Ces figures constituent une très bonne école pour aller au-delà de ce que nous présentons dans le reste de cet ouvrage.

Liens vers les cours de Eric Hakenholz, pour visualiser ou télécharger ses figures :

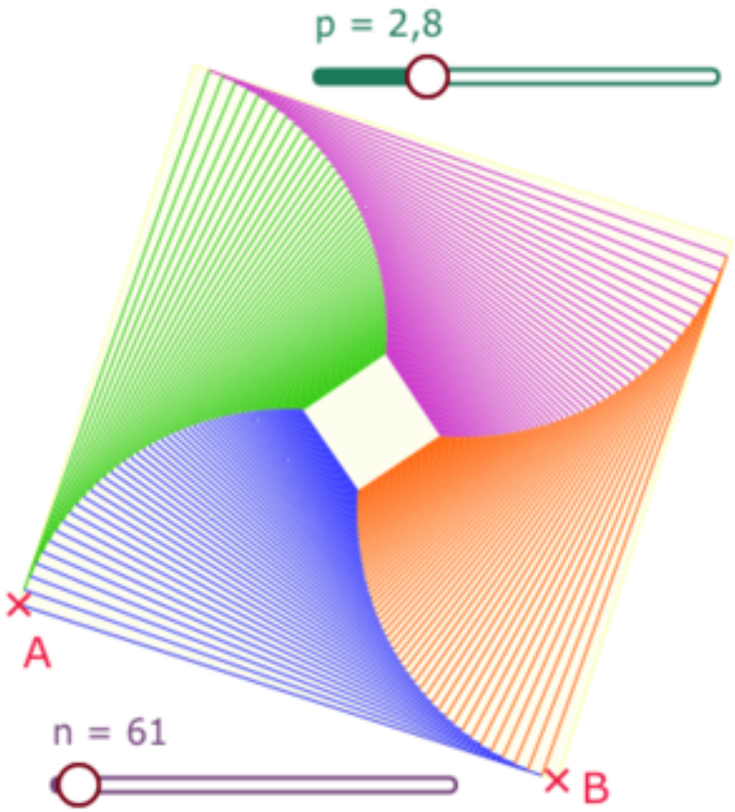
1. Angles et triangles (5°)
2. Nombres relatifs (5°)
3. Triangles rectangles et Pythagore
4. Fractions (5°)

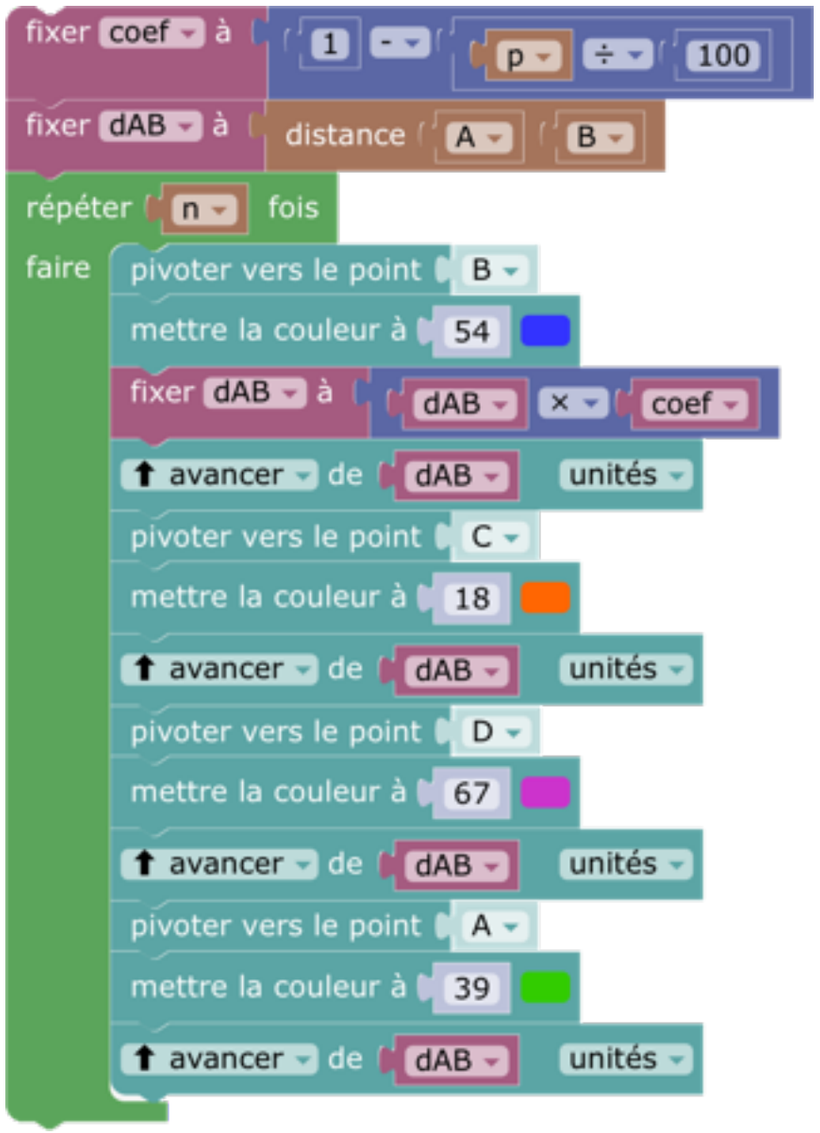
Construction auto-référente de la trace

Sur l'exemple standard de la courbe de poursuite dans un carré, nous allons voir comment on peut **construire la trace de la tortue en auto-référence (itérative) à sa construction**. On notera qu'il y a une différence avec un script ordinaire de construction car la tortue réagit en temps réel. On commence par une fausse piste, pour mieux exposer le code final.

Etape 1 : une fausse courbe de poursuite

Une erreur élémentaire – mais un code simple – est celle qui oublie de diriger la tortue vers l'itération précédente, en continuant de la pointer vers les sommets initiaux du carré :





Le carré initial est un carré ABCD (C et D cachés ici).

On voit sur l'illustration que les traces bleues sont orientées vers B, les oranges vers C, les mauves vers D et les vertes vers A.

Ce n'est donc clairement pas une courbe de poursuite car, dans celle-ci, la tortue devrait être orientée vers la position antérieure de chaque sommet et non vers sa position initiale.

Cette figure peut être manipulée dans la page suivante.

Figure dynamique 6.1 – Fausse courbe de poursuite (centration sur les sommets)

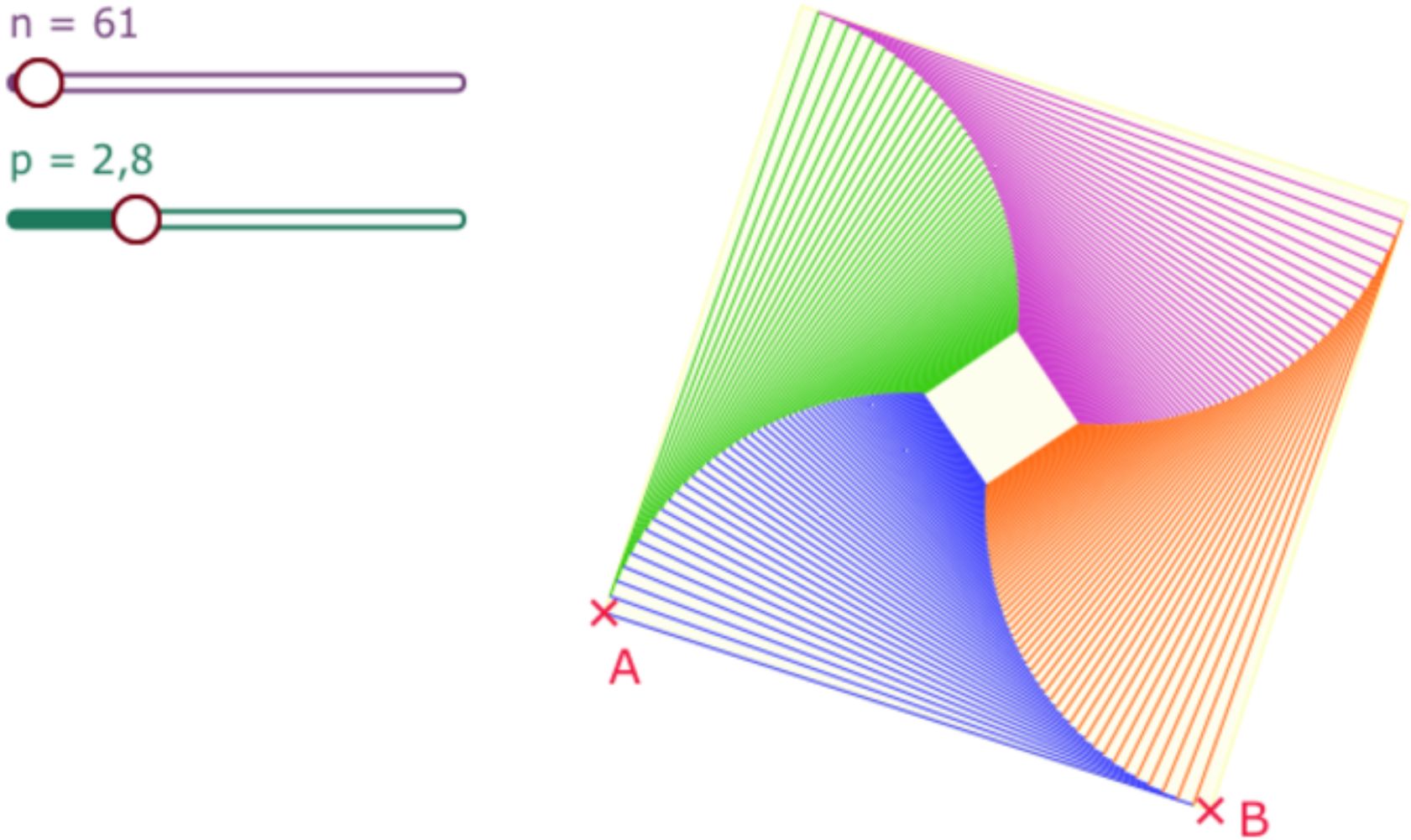
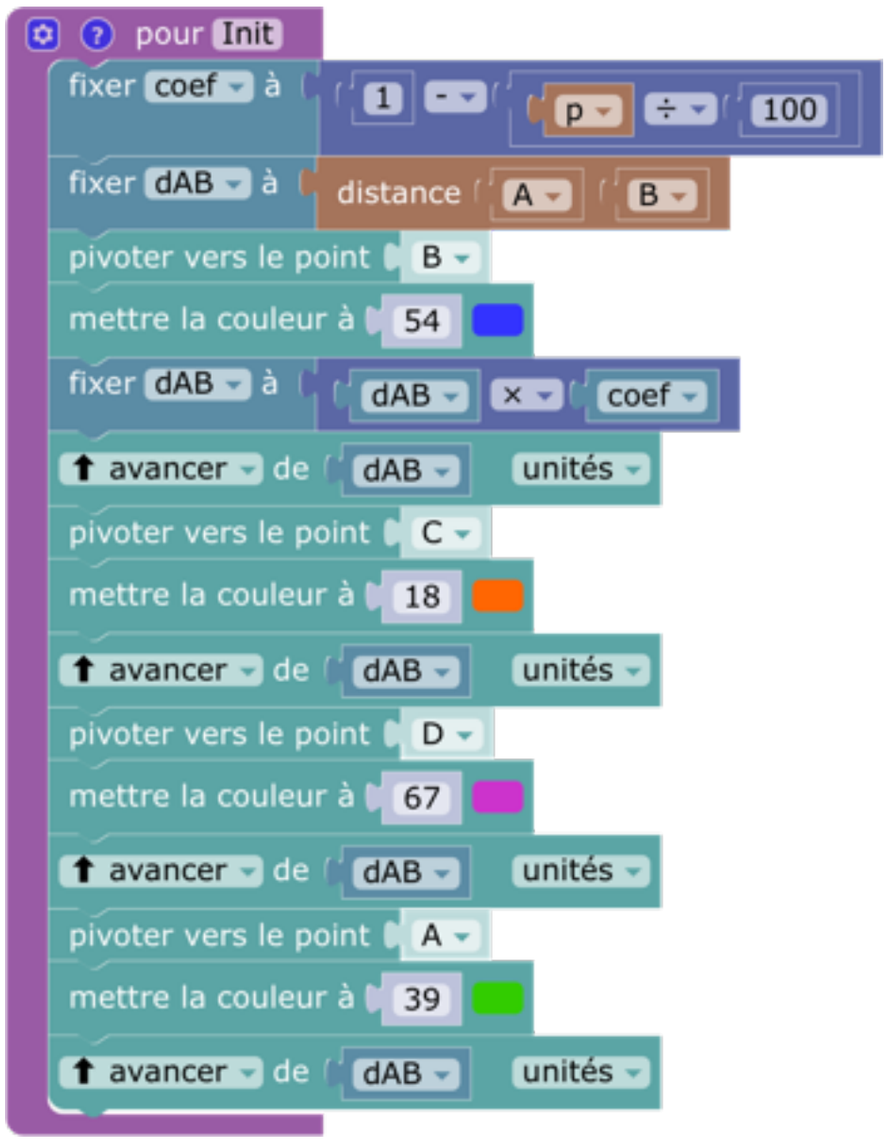


*Figure en mode **consultation** à l'ouverture - le code est dans le point A.*

Etape 2 : itération auto-référente

Pour itérer la référence, il faut commencer par faire un tour, c'est-à-dire placer le carré initial dans la trace de la tortue. C'est ce que fait la procédure d'initialisation.



La procédure **ItereCarre** construit un carré (B'C'D'A') à partir du carré précédent en se référant directement à la trace en cours de construction.

```
12 // 3. Itération directe par indiciation des points
13 for (i=0; i<nmax; i=i+1){
14     k=i+1;
15     m=Point("A"+k,"x(A"+i+")+coef*(x(B"+i+")-x(A"+i+"))","y(A"+i+")+coef*(y(B"+i+")-y(A"+i+"))");
16     n=Point("B"+k,"x(B"+i+")+coef*(x(C"+i+")-x(B"+i+"))","y(B"+i+")+coef*(y(C"+i+")-y(B"+i+"))");
17     p=Symmetry("C"+k,"O",m);
18     q=Symmetry("D"+k,"O",n);
19     Segment(m,n);Segment(n,p);Segment(p,q);Segment(q,m);
20     SetHide(m,true);SetHide(n,true);SetHide(p,true);SetHide(q,true);
21 }
```

Version CaRMetal en JavaScript (CaRScript) de la même procédure

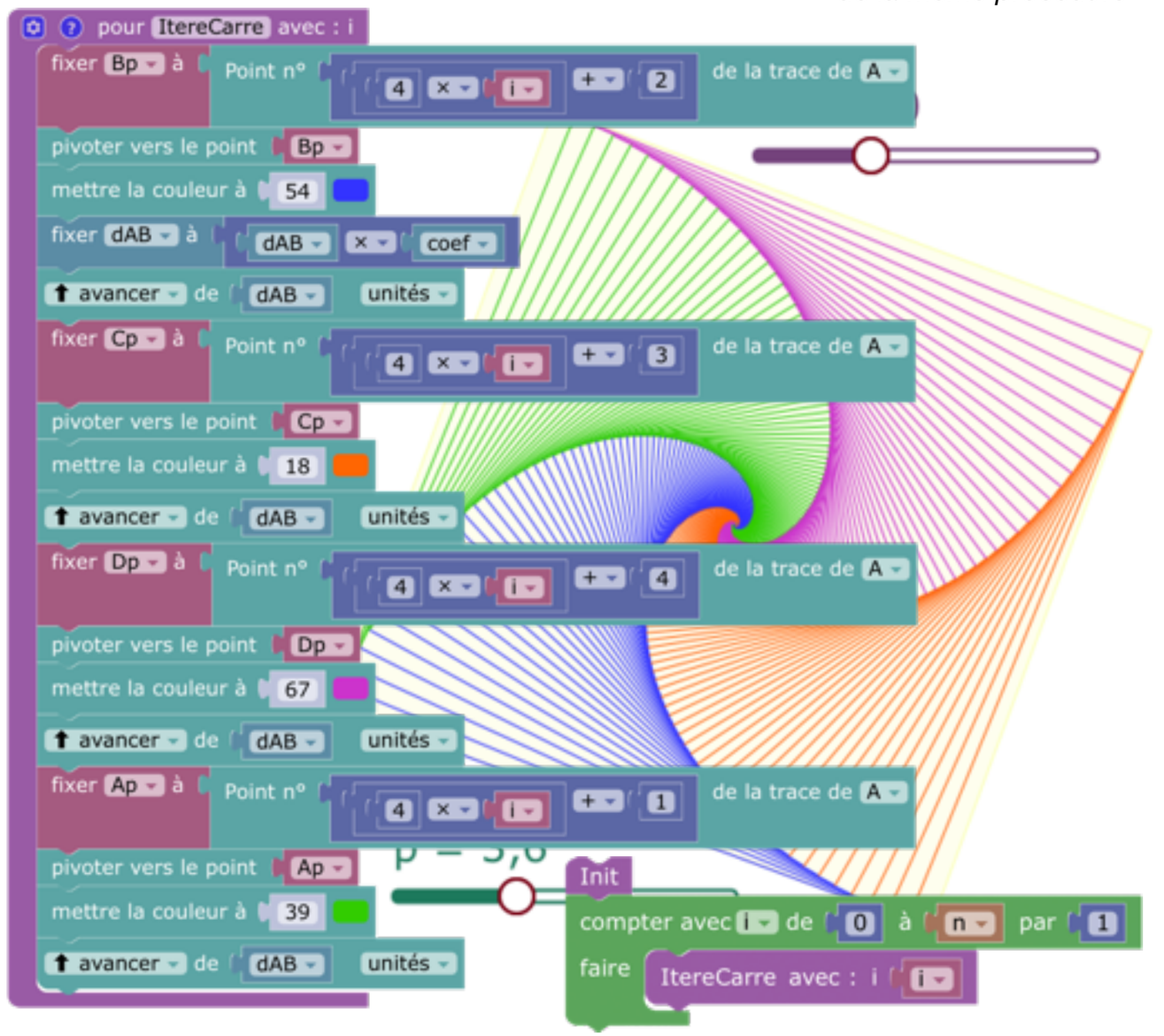


Figure dynamique 6.2 – Courbe de poursuite

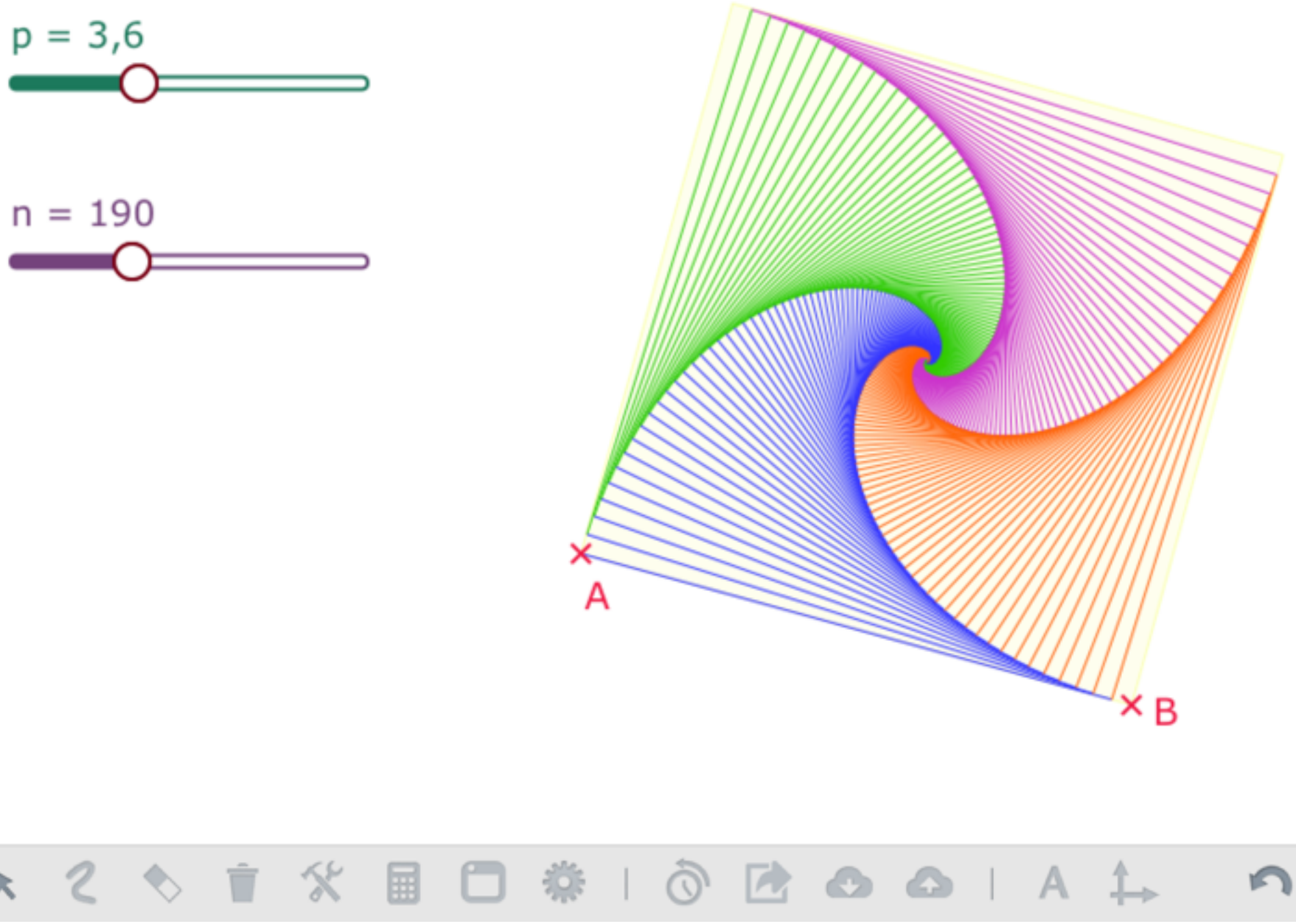
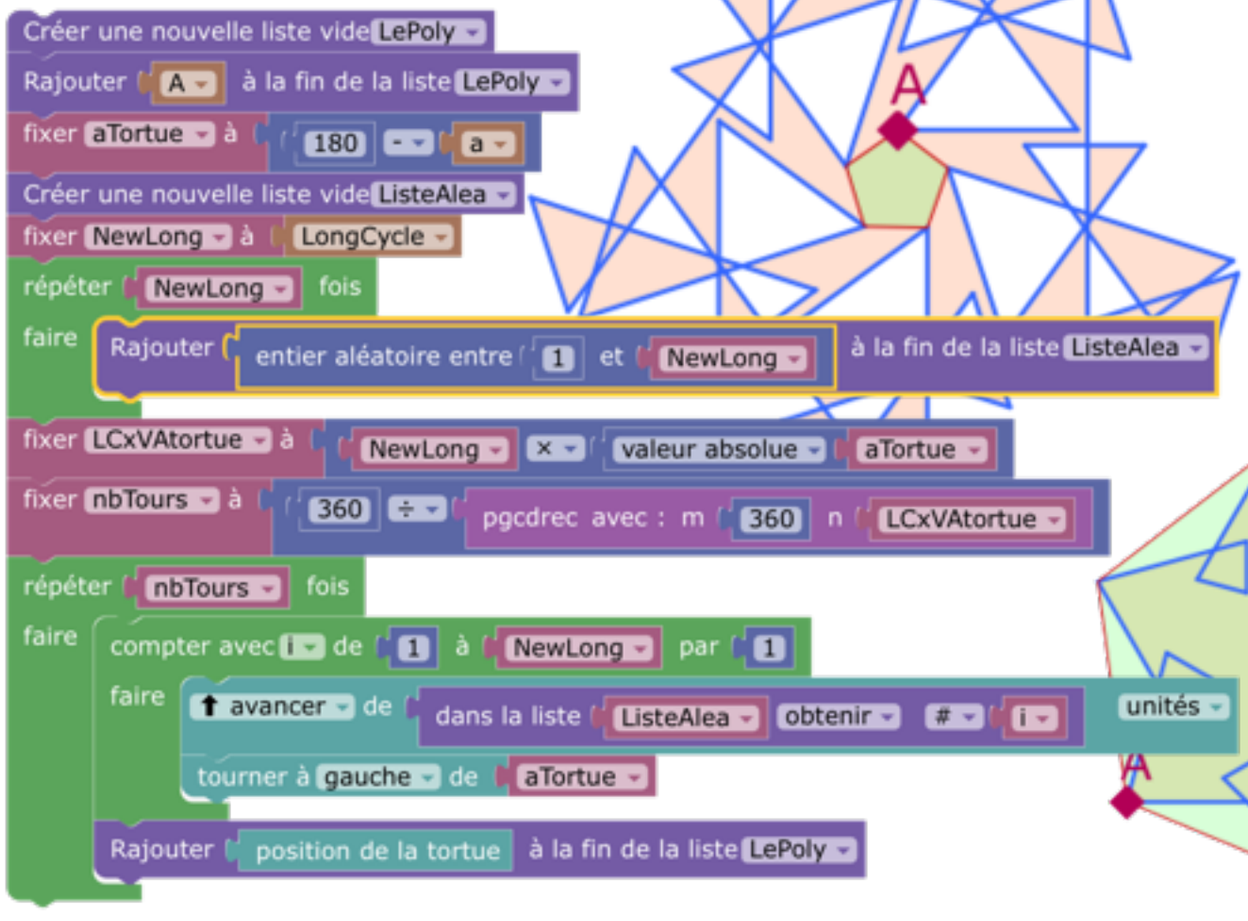


*Figure en mode **standard** à l'ouverture. Le code est en A.*

Utilisation géométrique de l'aléa

On reprend le thème déjà abordé des **spirolatères**, avec, pour un angle donné (curseur) et une longueur de cycle donnée aussi (ce qui fixe le nombre de côtés du **polygone des extrémités** de cycle). Cette fois c'est **un cycle aléatoire** qui va être remis à jour – et la figure – à chaque micro modification, d'un curseur ou du point A.

Voici un extrait du code où l'on voit **la construction du cycle aléatoire** (premier **répéter**) et sa construction (le second). On notera la construction du polygone des extrémités des cycles qui se fait, encore une fois **algébriquement**, en engrangeant la position de la tortue dans une liste.



La construction du polygone se fait simplement par **parcours de la liste**, sans qu'aucun calcul n'ait à être fait. Un exemple assez spectaculaire d'algébrisation.

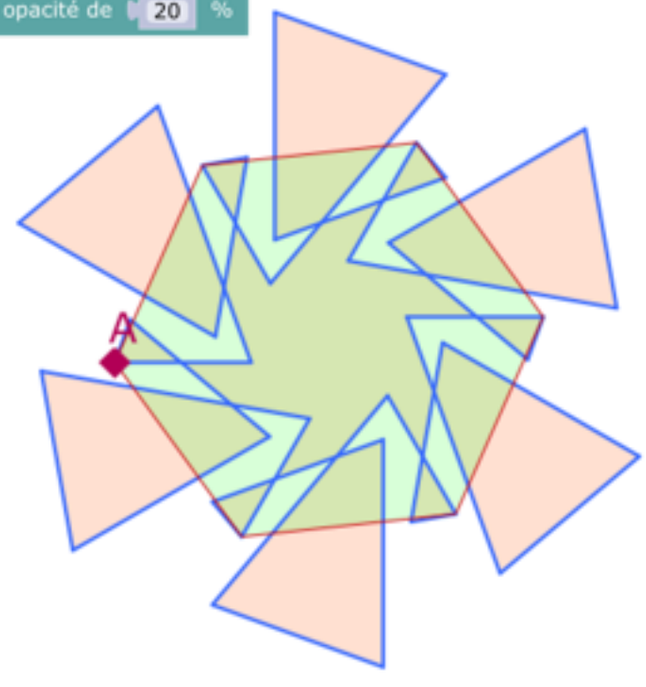
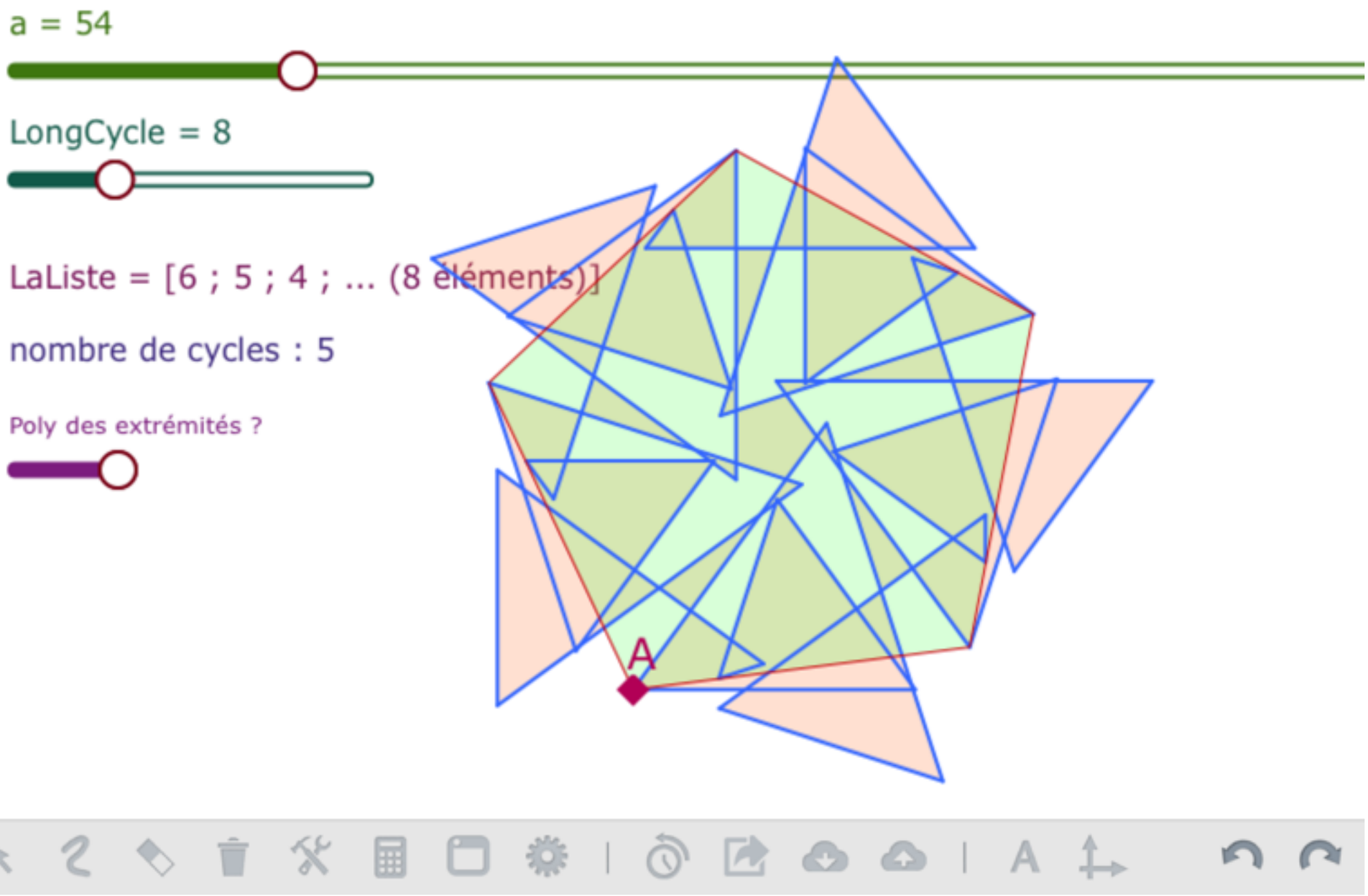


Figure dynamique 6.3 – Spirolatères à cycles aléatoires



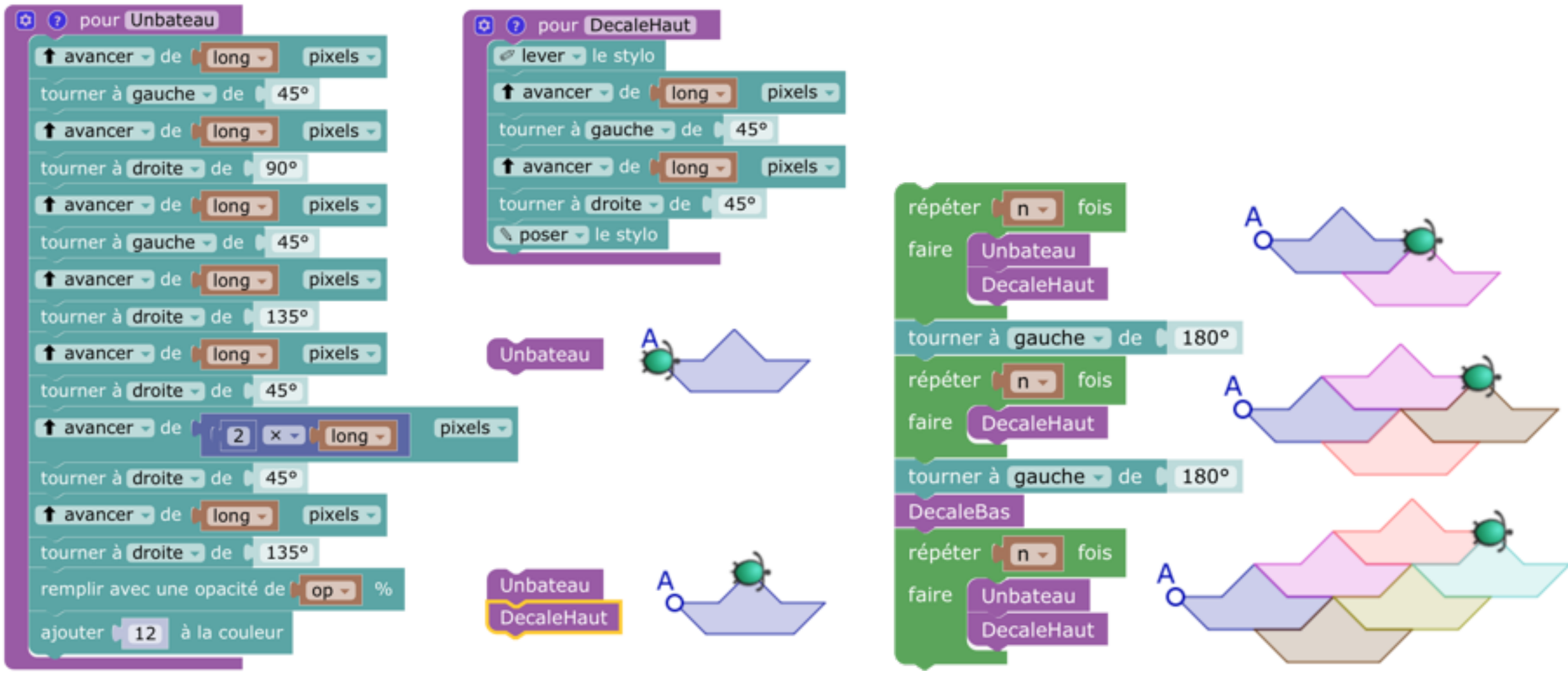
Déplacer simplement A. On choisira des angles ayant un diviseur commun avec 360, de même pour la longueur de cycle. Même lors d'une micro modification d'un curseur (sans changer sa valeur), le spiromètre est mis à jour.

Deux exemples de pavage - et explorations ludiques

Il y a de nombreuses façons de réaliser des pavages du plan : les 17 standards et bien d'autres dont cette très belle figure dynamique de Julien Pavageau sur les quadrilatères, sans parler des non-périodiques. On se propose de faire ici des choses plus simples, mais différentes.

1. Pavage par un motif de bateau (heptagone)

La définition du bateau n'utilise que des entiers (pas de calculs). On peut déplacer la tortue par défaut à plusieurs endroits (d'où la procédure **DecaleHaut**). La deuxième illustration propose une exploration en fonction de n (pour n=1, 2 et 3).



Pavage par étoile : avec les deux simples procédures de la définition du motif et d'un décalage, on peut réaliser un pavage étoilé, mais aussi proposer plusieurs explorations sur le motif, comme ci-contre.
Dans la figure suivante, enlever le coefficient n dans les boucles et le remplacer par des valeurs numériques différentes.
Notez que le point A n'est pas au centre.

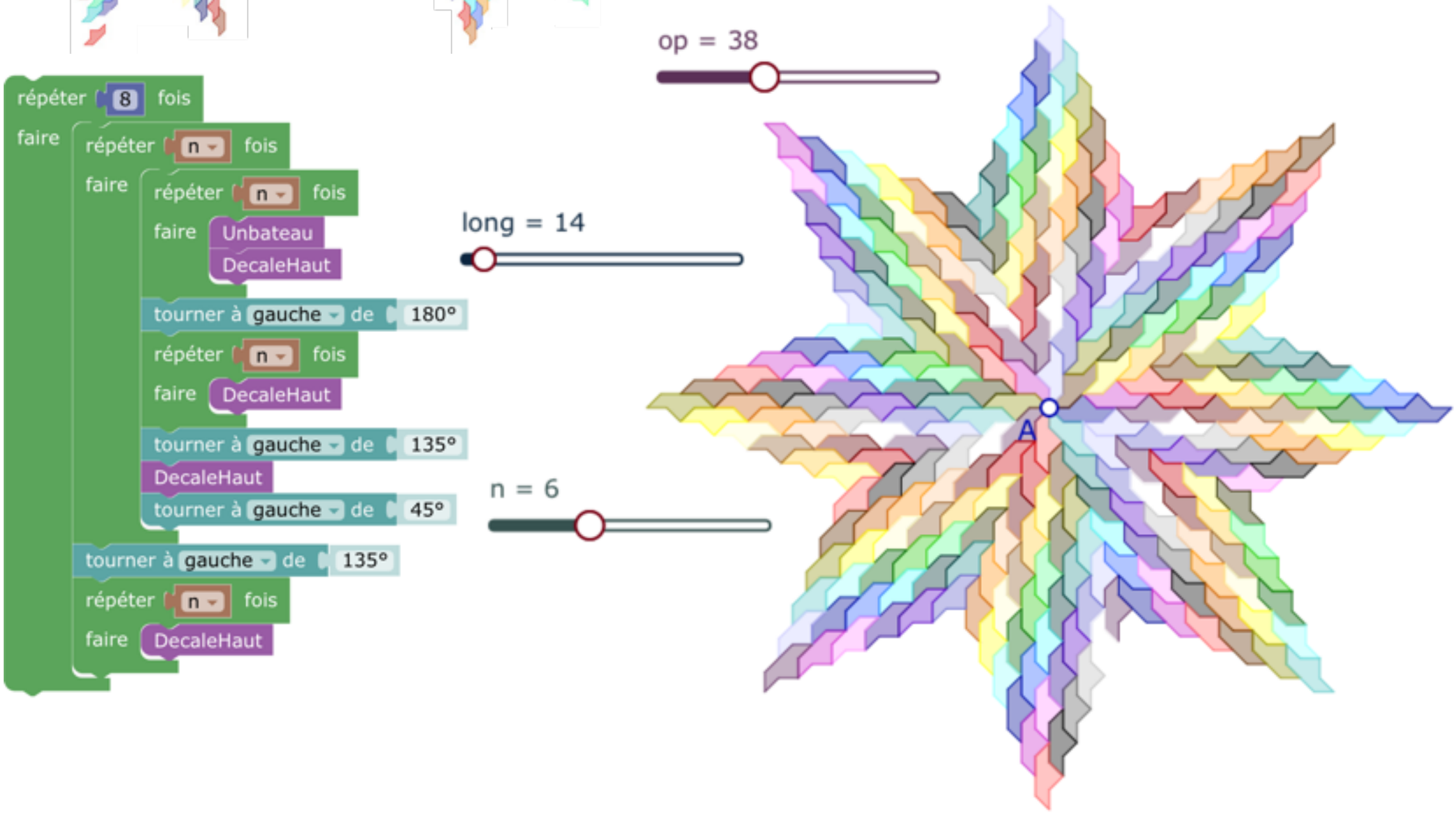
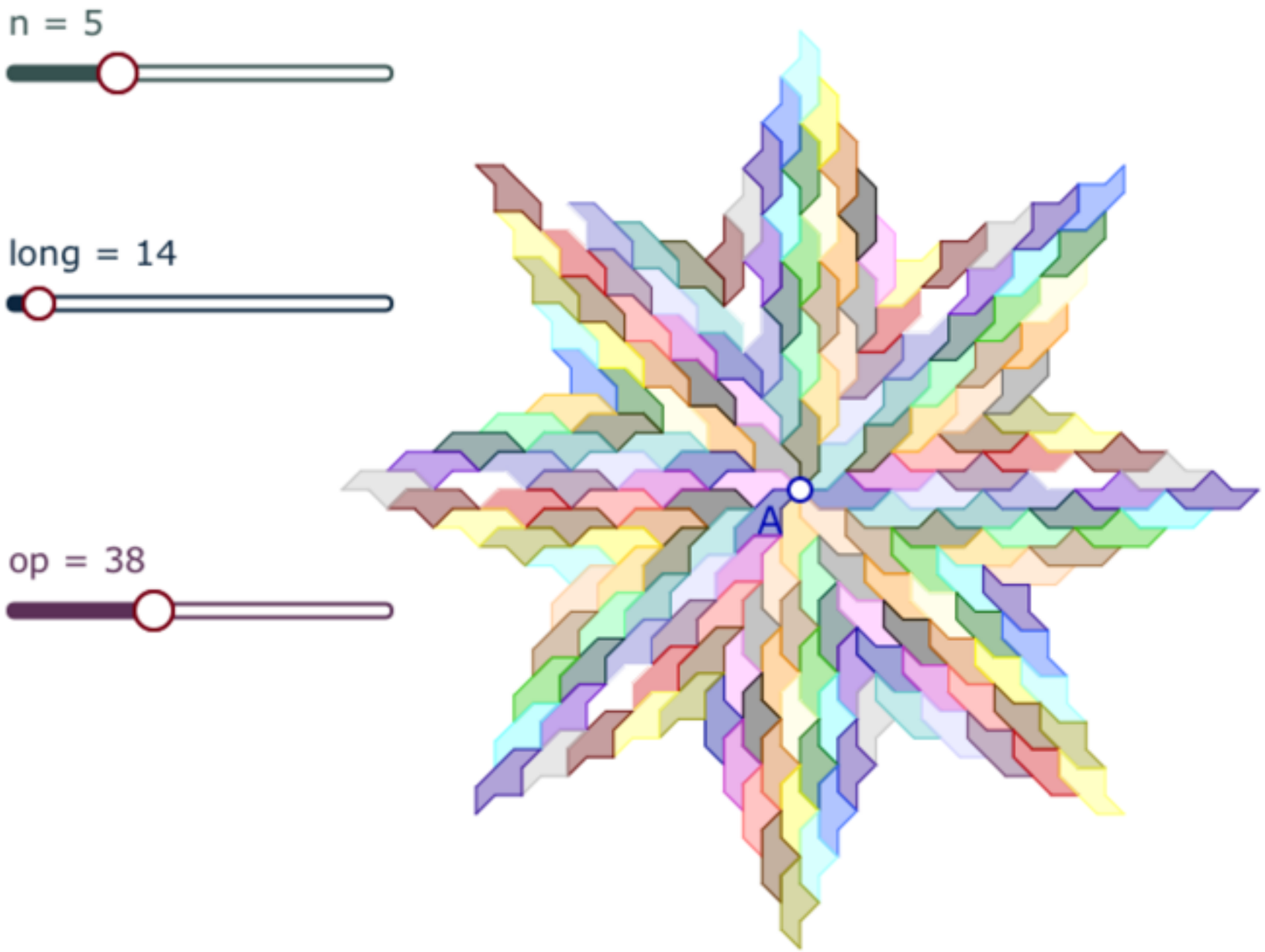
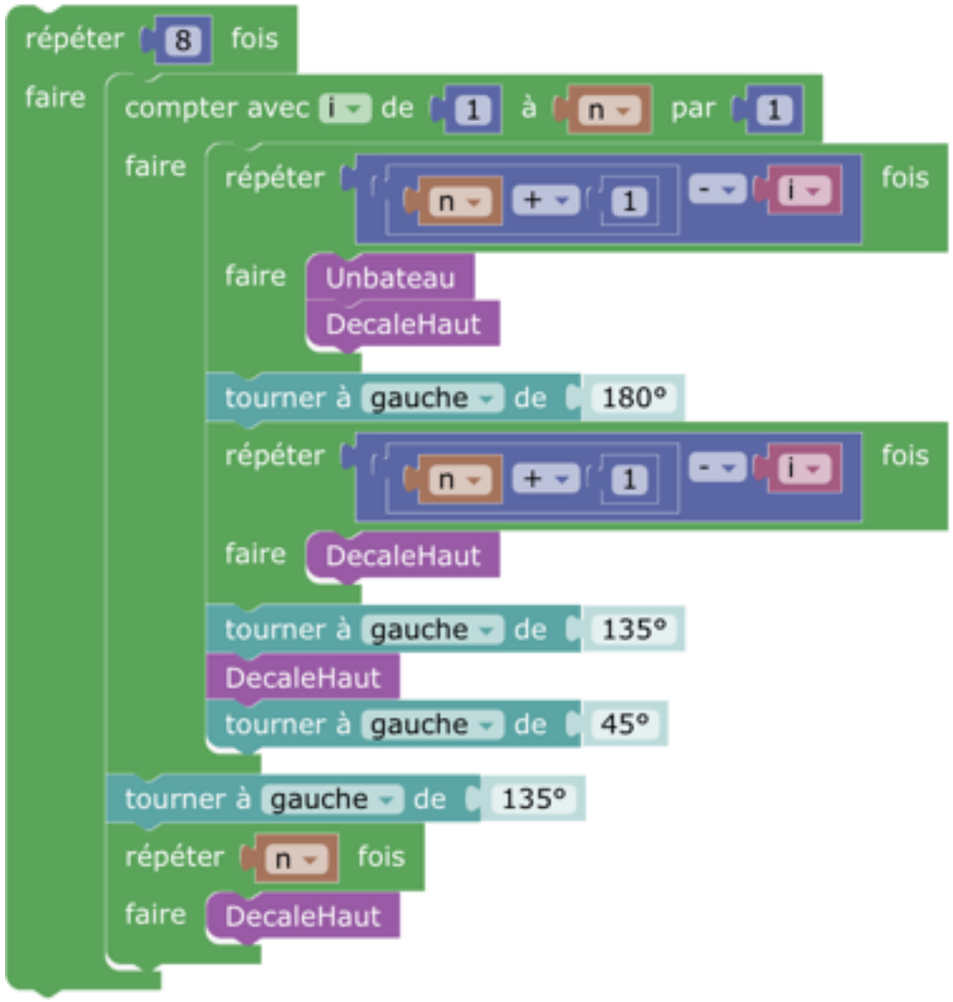


Figure dynamique 6.4 – Pavage par étoile sur le motif heptagone du bateau



*En mode **standard** à l'ouverture pour modifier le paramètre **n** et le remplacer par des entiers.*

Pavage par octogone : toujours avec les deux mêmes procédures de base, mais avec des compteurs. Pour obtenir les extrémités comme des côtés d'octogone, on peut produire un pavage du plan différent. Là encore, dans la figure suivante, en peut remplacer le curseur n par des paramètres numériques différents.



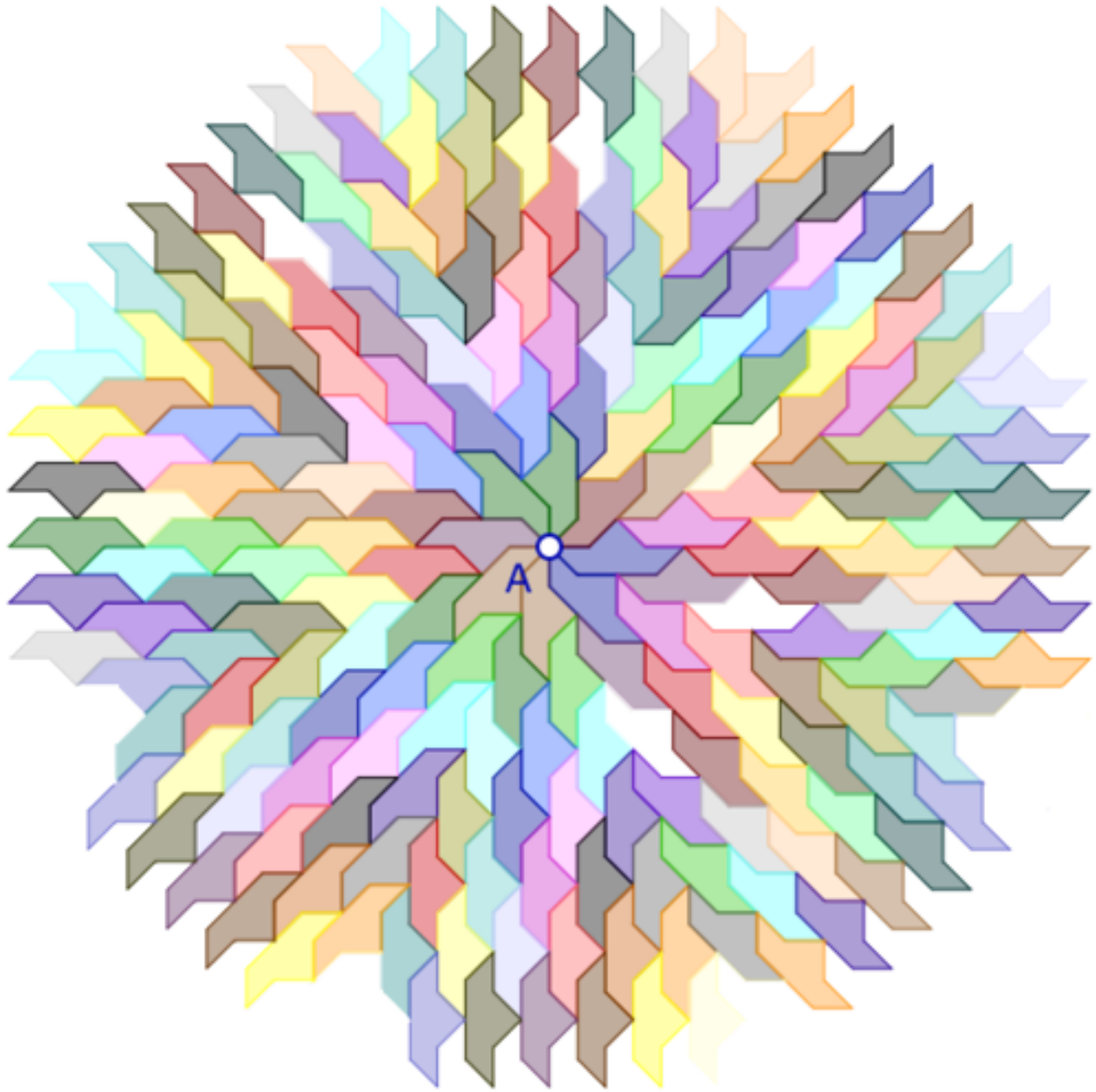
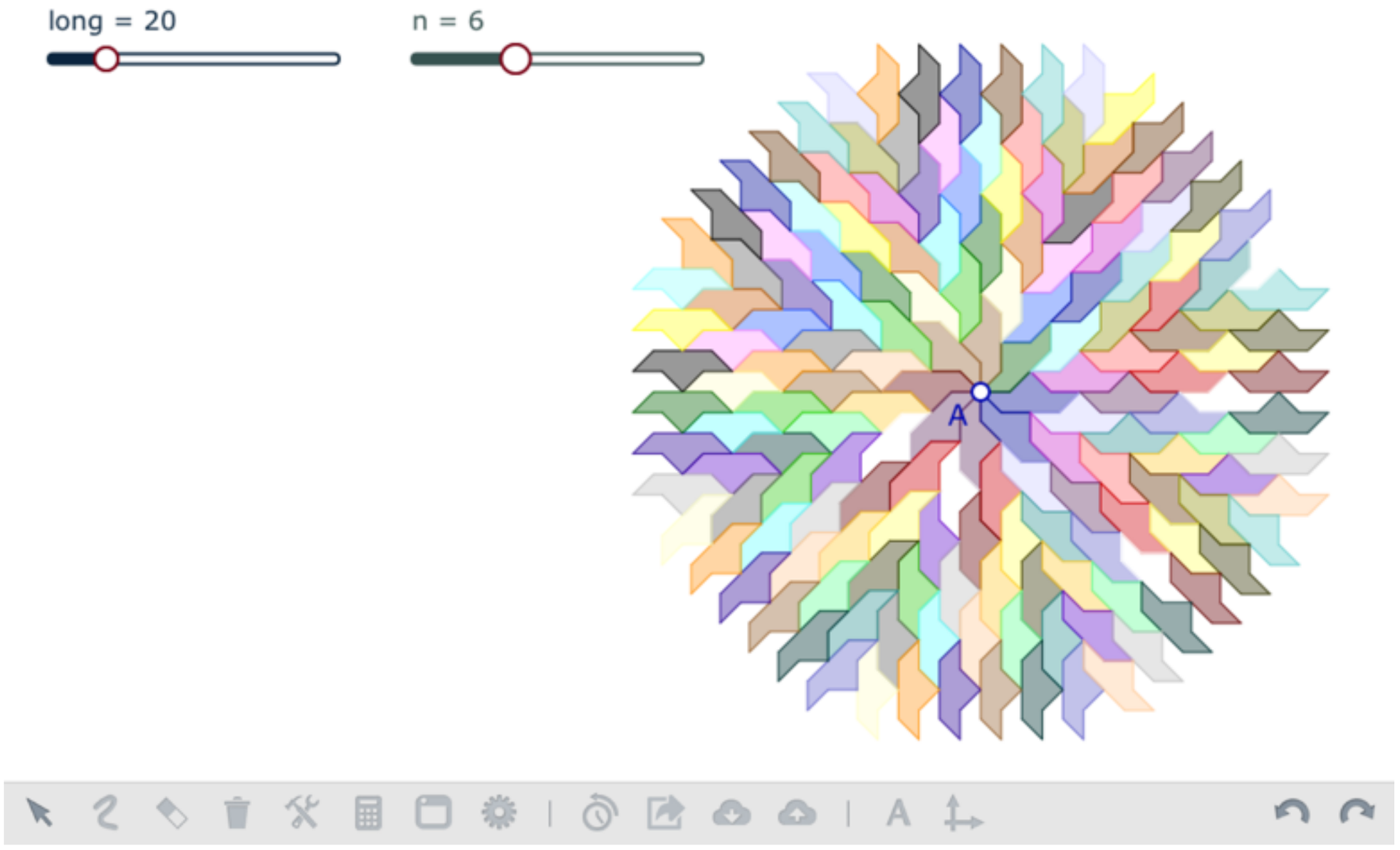


Figure dynamique 6.5 – Pavage par octogone avec le motif du bateau

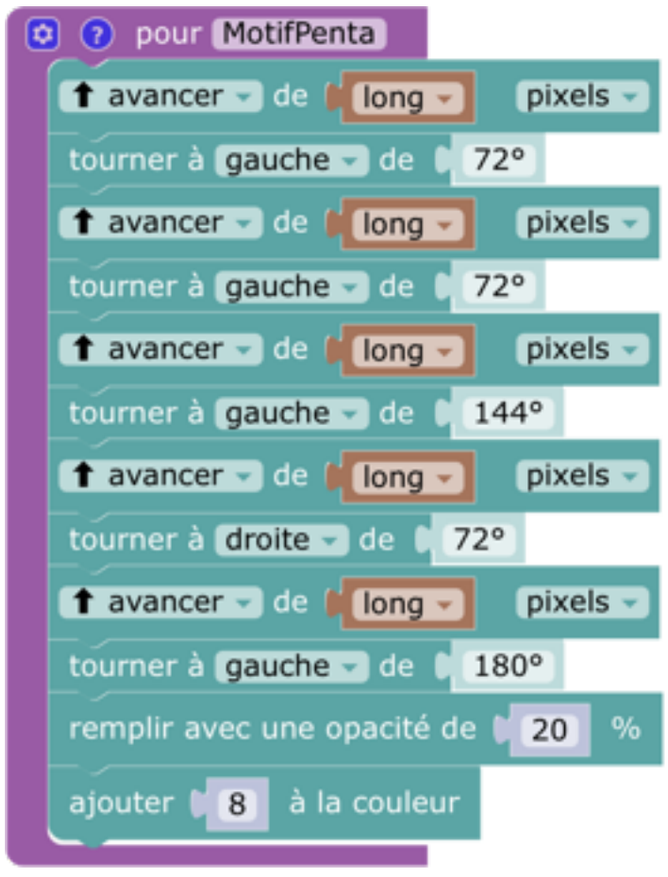


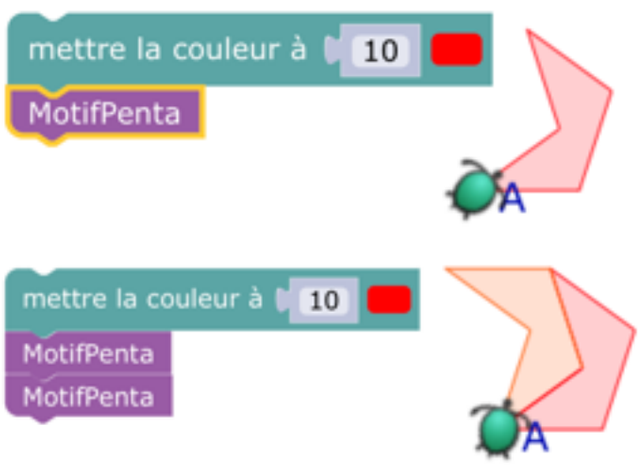
Même remarque que la figure précédente.

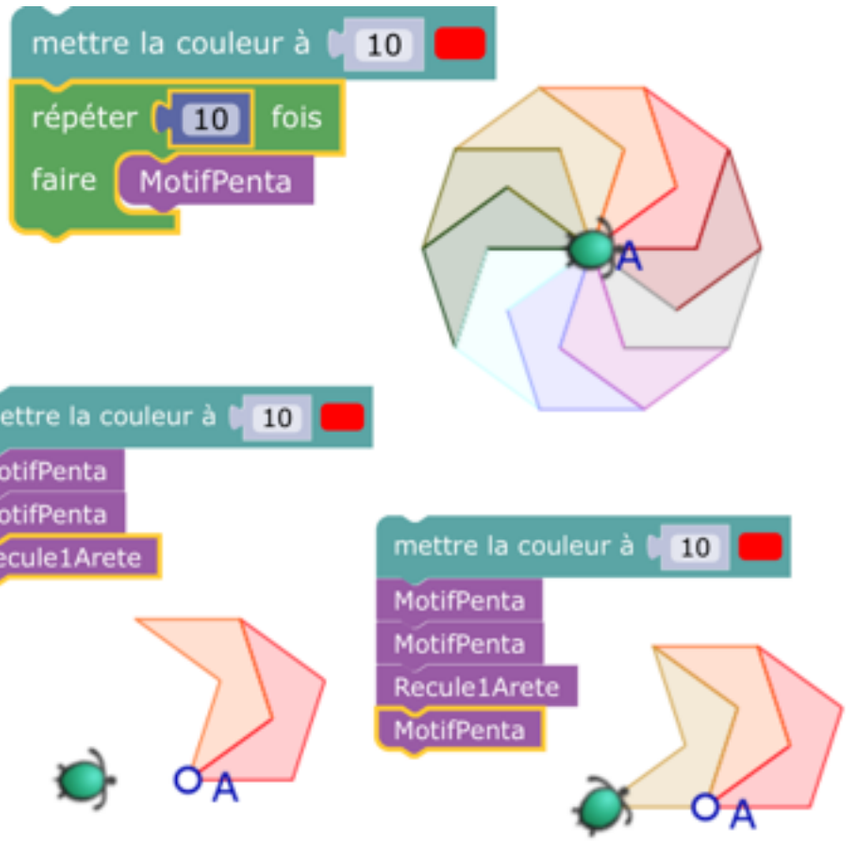
2 - Pavage par couronnes à base de pentagone

Là encore, le motif est construit à partir d'une longueur donnée. Plusieurs approches sont possibles pour découvrir l'angle du pentagone au sommet A. Soit on a une approche heuristique, en explorant avec une boucle **répéter** (après avoir expliqué pourquoi la tortue est sur le second côté de l'angle en A). On s'aperçoit alors qu'il faut 10 motifs pour faire le tour, d'où un angle de 36°, soit on propose une activité qui permet de le calculer avant, afin d'anticiper le nombre de motifs.

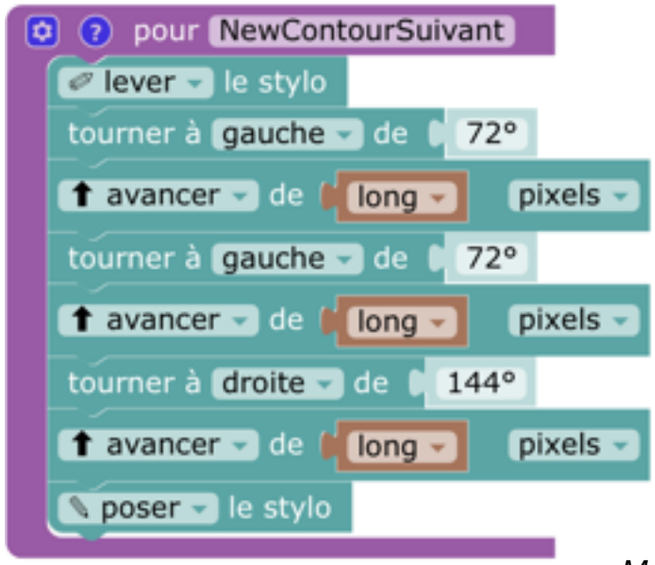
Premières explorations : on remarquera que la procédure **Recule1arete** n'est pas paramétrée par un angle, donc ne s'applique – pour être dans l'axe de l'arête dont on recule – que si deux motifs ont été construits.

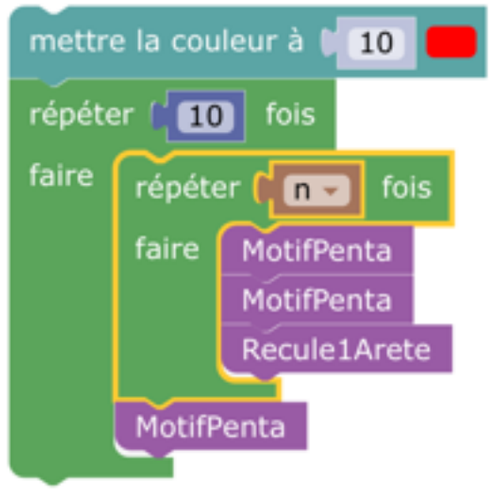


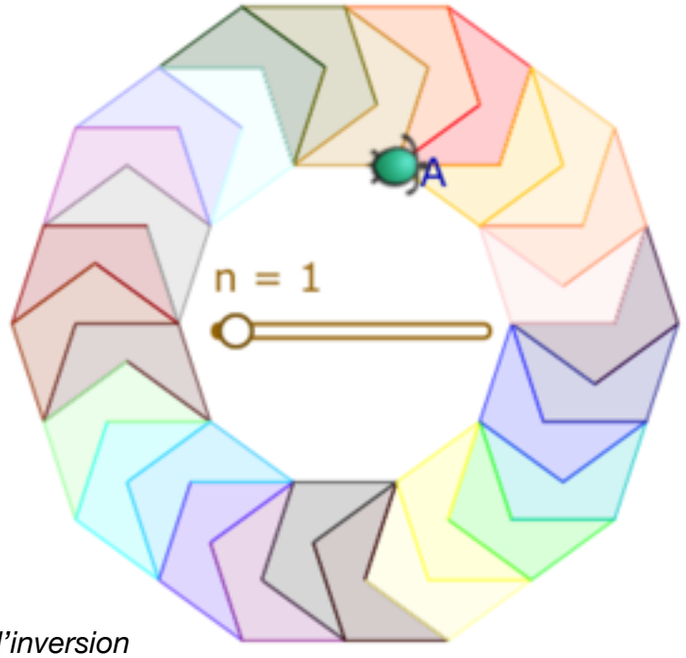




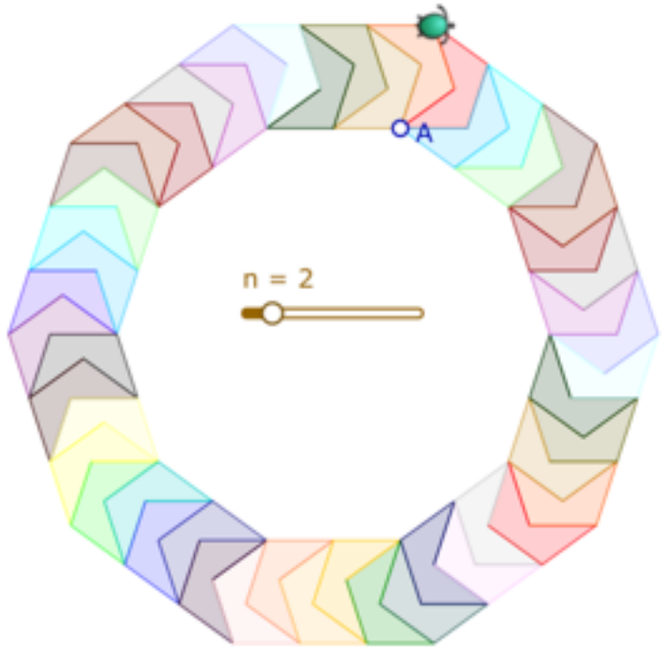
Remplissage par couronne : à l'image du cas précédent, deux procédures – le motif et un déplacement – suffisent à faire un motif significatif, celui de couronnes. Pour itérer – et donc paver – il suffit de replacer, par une troisième procédure, la tortue au bon endroit. Cette troisième procédure n'est pas optimisée, on pourrait n'avoir que deux déplacements.

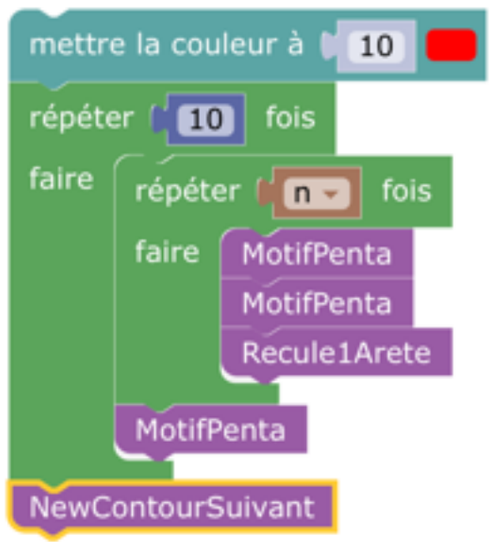






Merci à Patrice Débrayant pour cette idée d'inversion des «signes somme», ici des deux boucles.





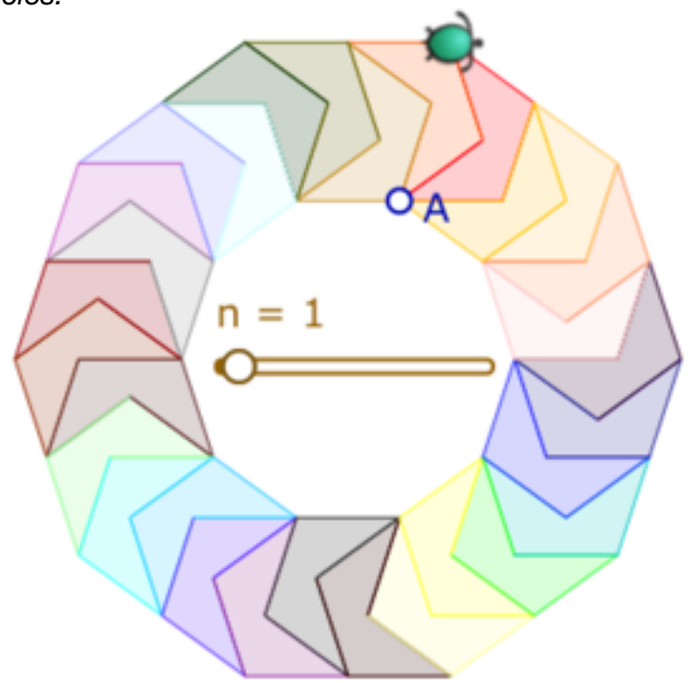
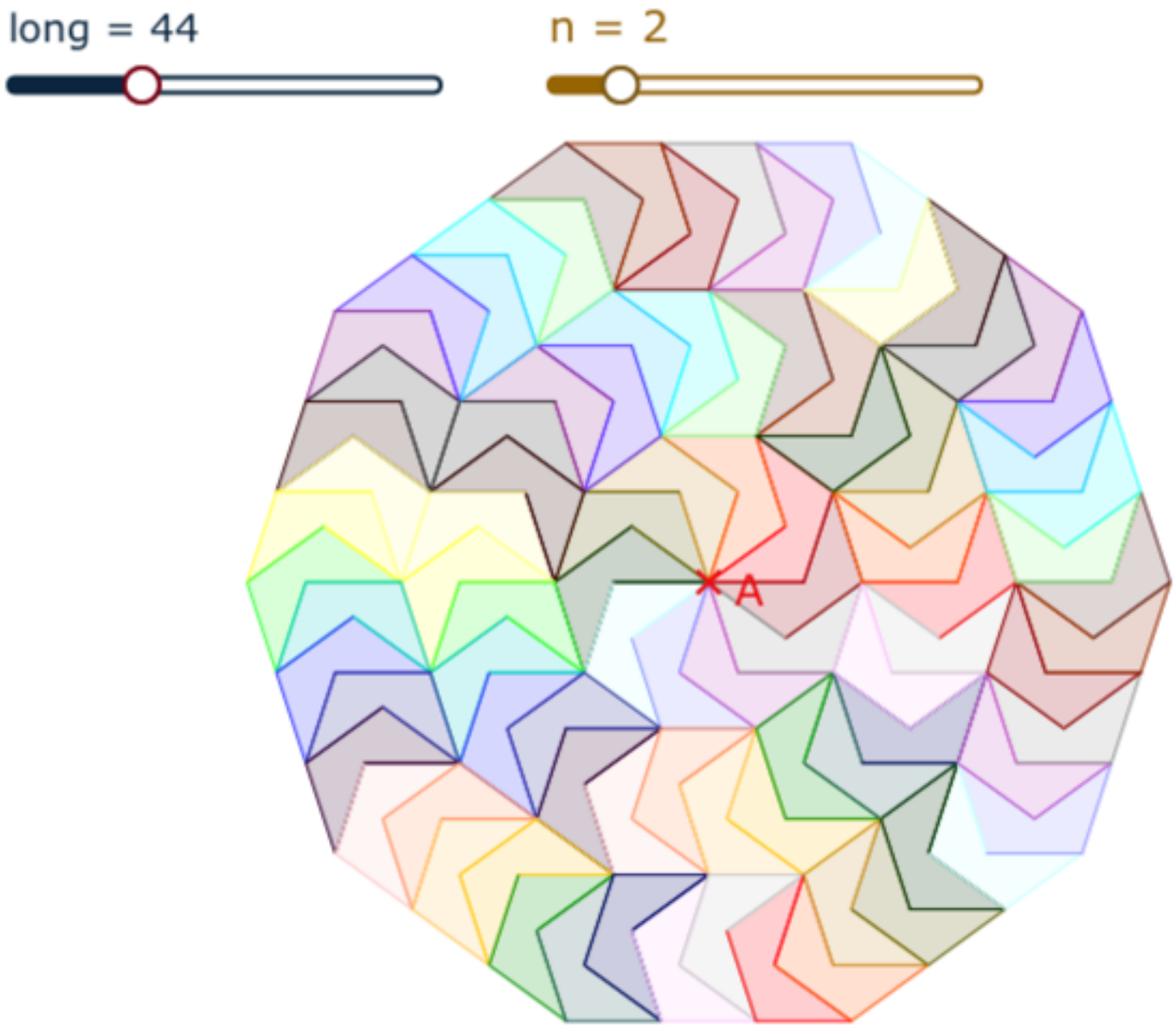
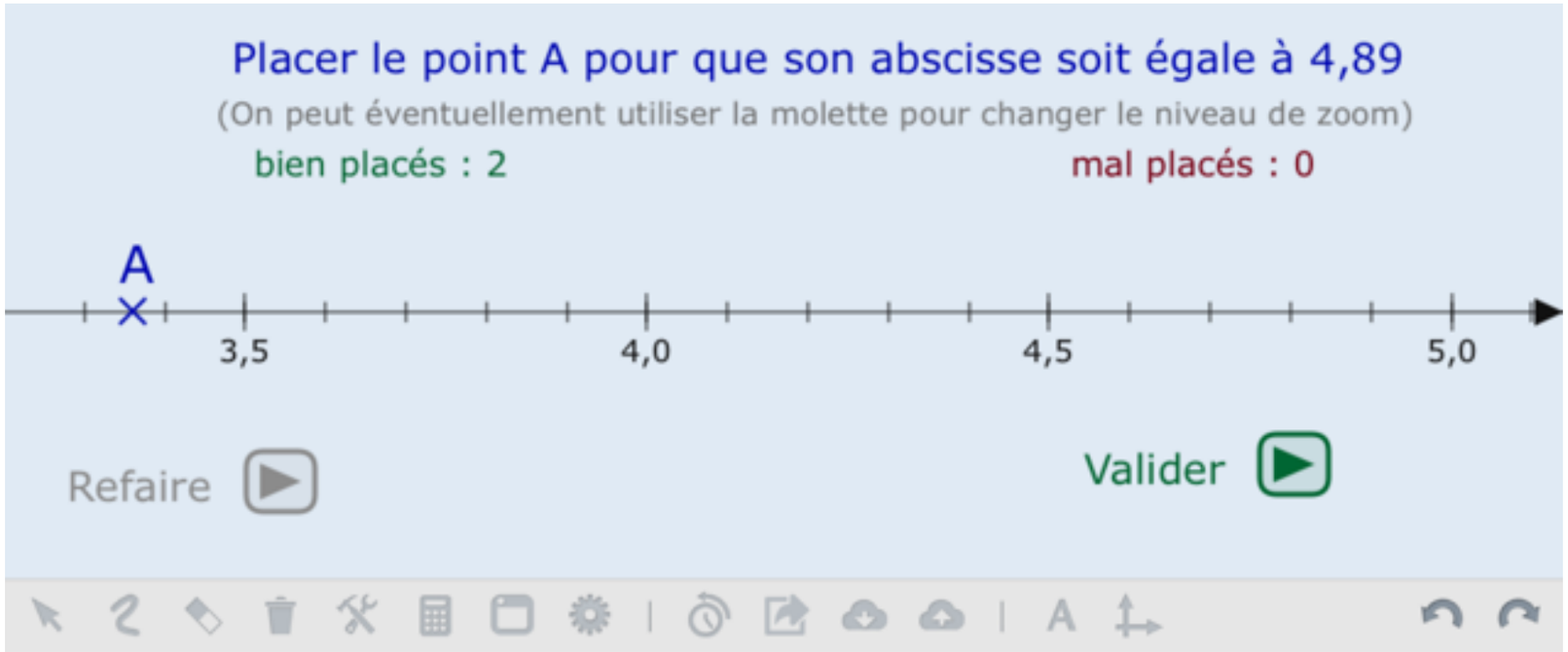


Figure dynamique 6.6 – Pavage par itération d'une couronne de pentagone



Quelques figures de Eric Hakenholz pour le collège

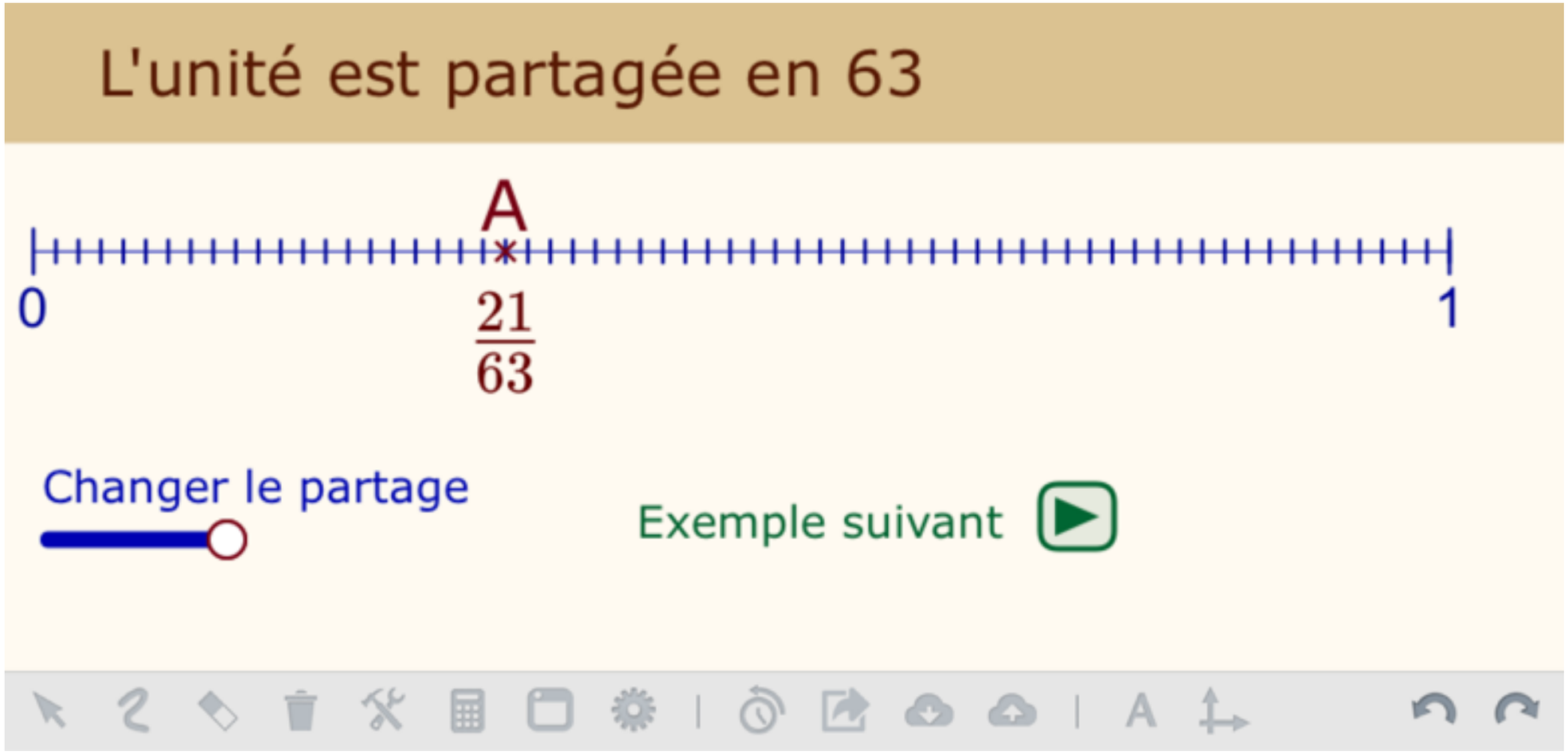
Figure dynamique 6.7 – Placer un point décimal



*Il faut être en mode **consultation** (sans outils) pour déplacer l'axe des abscisses. Le code est simple.*
La plupart des objets sont inertes pour que les élèves ne les déplacent pas, et donc non évitable (modifiable dans l'inspecteur).

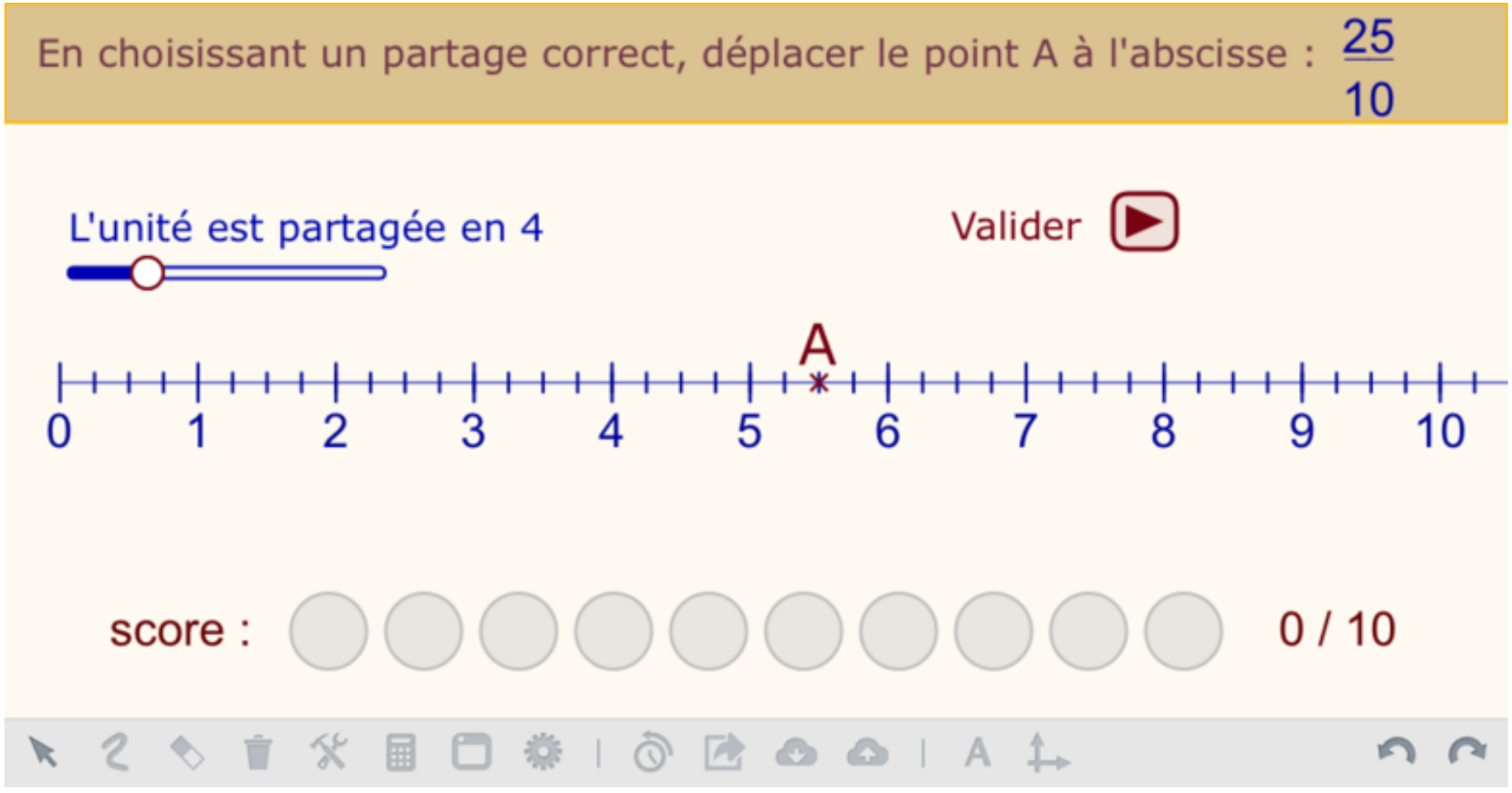
Cette figure aborde les centièmes, une autre ne traite que des dixièmes, et une troisième fait travailler les millièmes, il faut avoir plus l'habitude de zoomer.

Figure dynamique 6.8 – Visualisation du fractionnement de l'unité dans la simplification de fraction



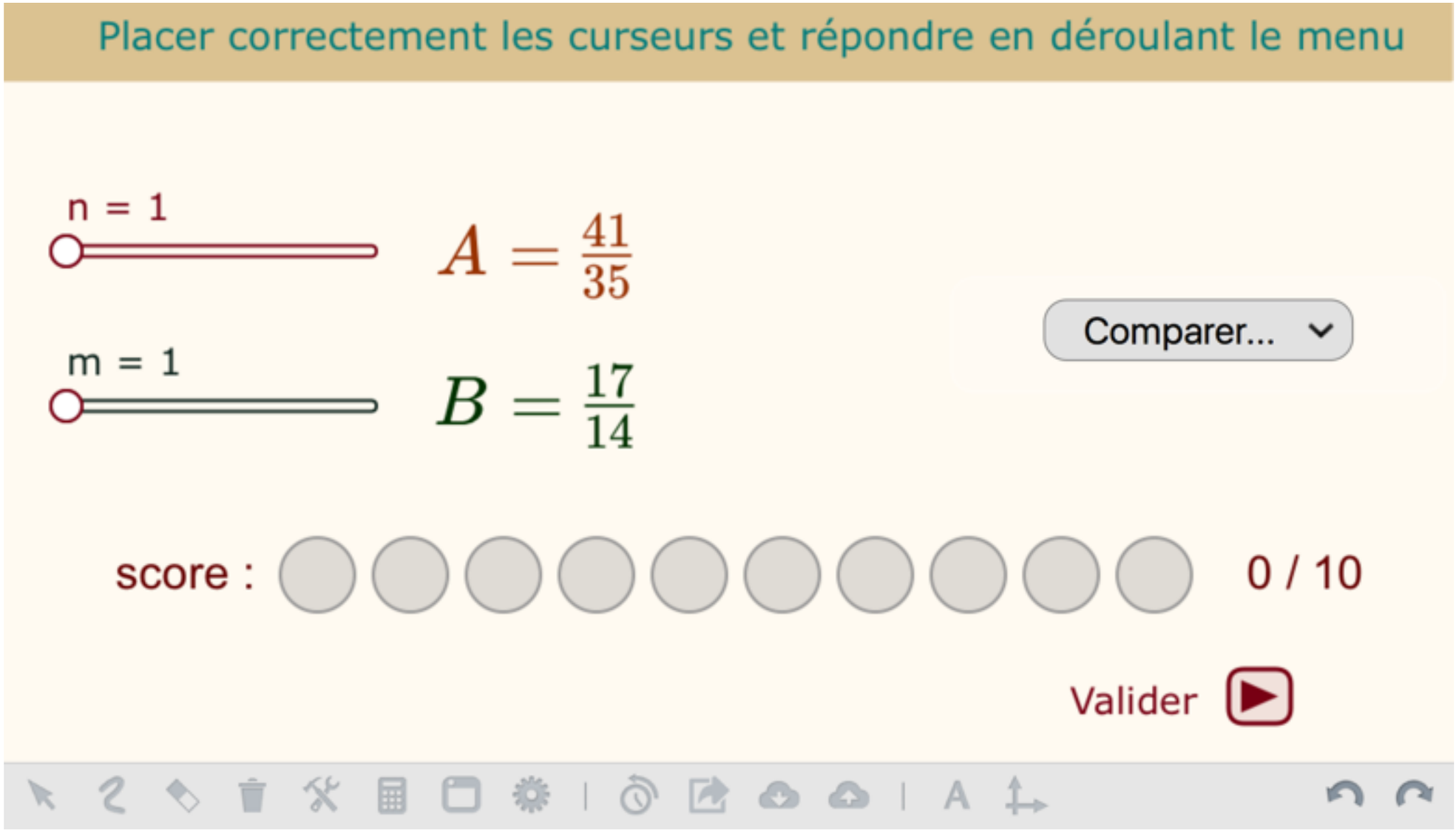
Cette situation où le point A et l'unité ne changent pas, et ce fractionnement de l'unité qui change autour de A quand on actionne le curseur, offre une construction dynamique saisissante des représentations du fractionnement chez les élèves.

Figure dynamique 6.9 – Choix de l'unité de partage par les élèves pour placer un point



Là encore si on a besoin de déplacer le repère, il faut passer en mode ***consultation****.*

Figure dynamique 6.10 – Comparer des fractions en réduisant au même dénominateur

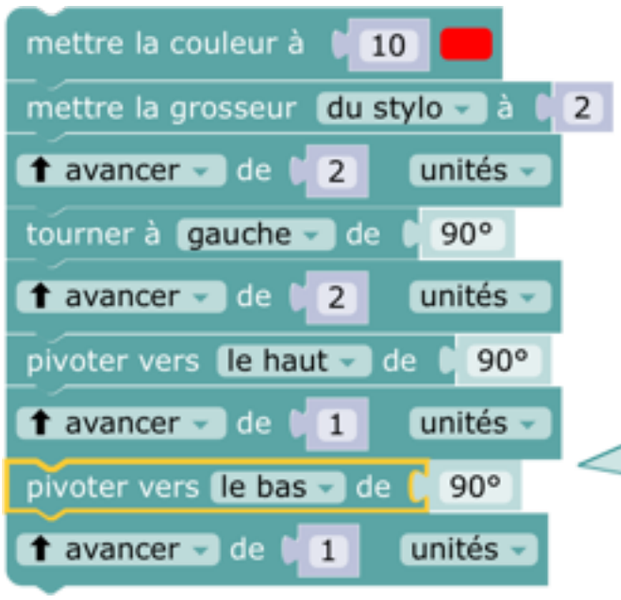


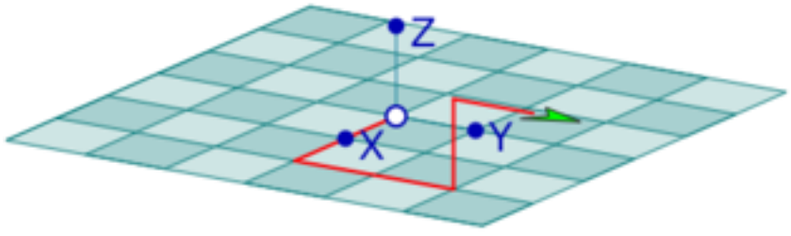
Ce qui est surprenant, c'est que cette figure est entièrement faite à la tortue. Une grande partie du code est dans le **DGScript** *Valider*. Une autre partie est dans l'expression **score**, inerte, qu'il faut rendre active pour pouvoir l'éditer. Le bouton *Comparer* est un **widget de DGPad** que l'on édite avec l'icône widget (entre la calculatrice et la roue de l'inspecteur).

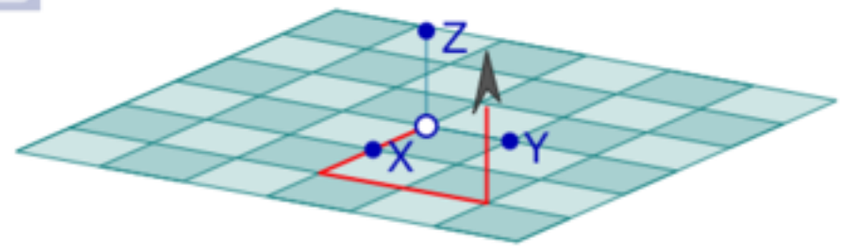
Tortue 3D au collège

Un autre apport essentiel de la tortue dynamique de DGPad réside dans ses outils 3D pour la constructions de polyèdres ou de patrons pliables. Certains polyèdres ou patrons sont très simples à construire de par le fait, tout à fait nouveau en milieu scolaire, que l'on travaille dans le repère mobile de la tortue, ce qui rend les procédures à mettre en place – pour des objets simples – à la fois courtes et conceptuellement réalistes en classe.
Une autre démarche didactique que nous allons envisager est une réflexion sur l'optimisation du code, que l'on peut alors – dans certains cas – rapprocher de la démarche de factorisation en algèbre. Là encore, même si c'est plus conceptuel, on peut faire un lien entre certaines attitudes algébriques et d'autres de programmation.

La tortue 3D est une flèche, le dos (dessus) est vert, le ventre (dessous) noir. Ci-contre, de par son parcours, on voit que par défaut la tortue est initialement dans le plan (xOy), sur, ou dans la direction de l'axe (Ox).

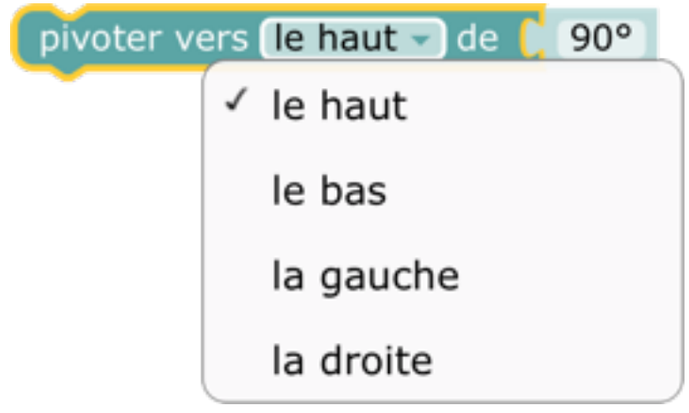






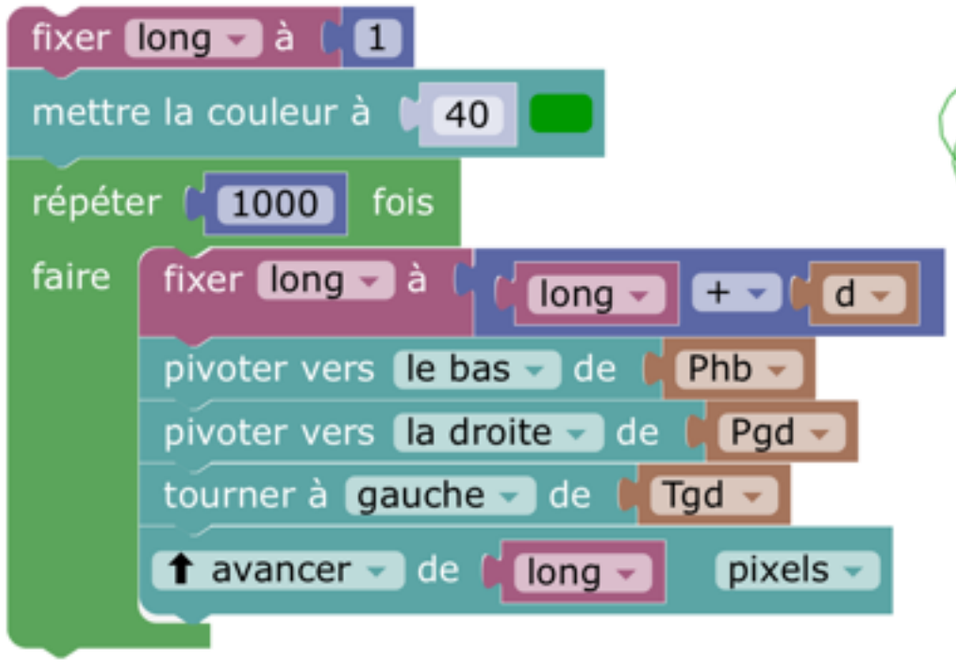
Toutes les instructions sont relatives à la tortue, ainsi le «pivoter vers le bas de 90°» ci-contre s'applique à la tortue comme on la voit dans l'illustration ci-dessus. Il en résulte qu'elle revient dans la direction (Oy), on voit à nouveau sa carapace (flèche verte ci-contre).

Exploration élémentaire pour appropriation des outils



Avant de commencer à travailler «scolairement», voyons deux figures proches, en version 3D, des spirales que nous avons déjà exploré en 2D. Tout d'abord celle ci-dessous, à 4 curseurs, où le préfixe **P** signifie **Pivoter** et **T** signifie **Tourner**. La taille est pilotée par le curseur **d**. On peut aussi modifier la borne de la boucle.

Dans la page suivante, vous pouvez tester les variantes **haut/bas**, **droite/gauche** et même ne pas respecter les suffixes **hb** et **gd**.



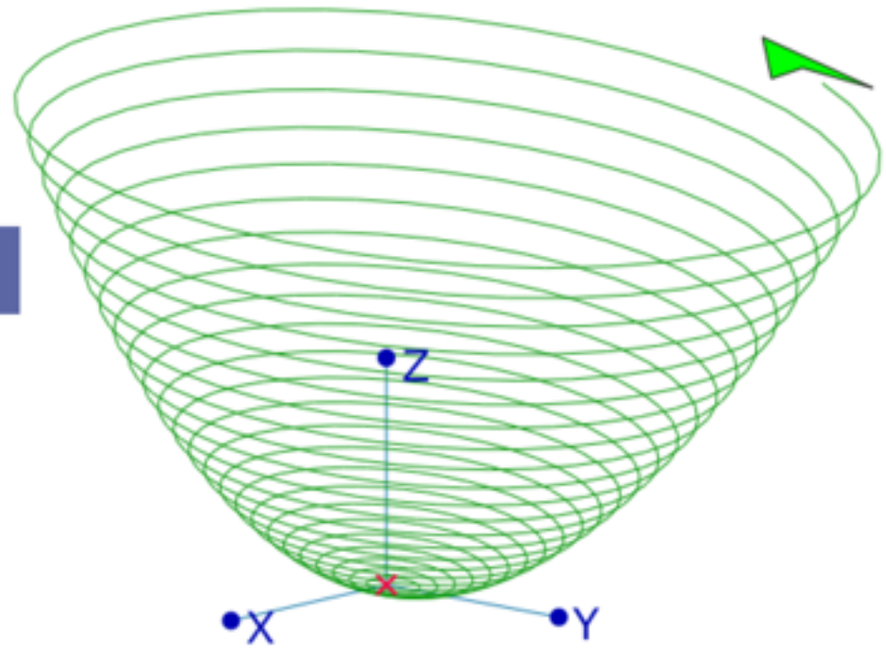
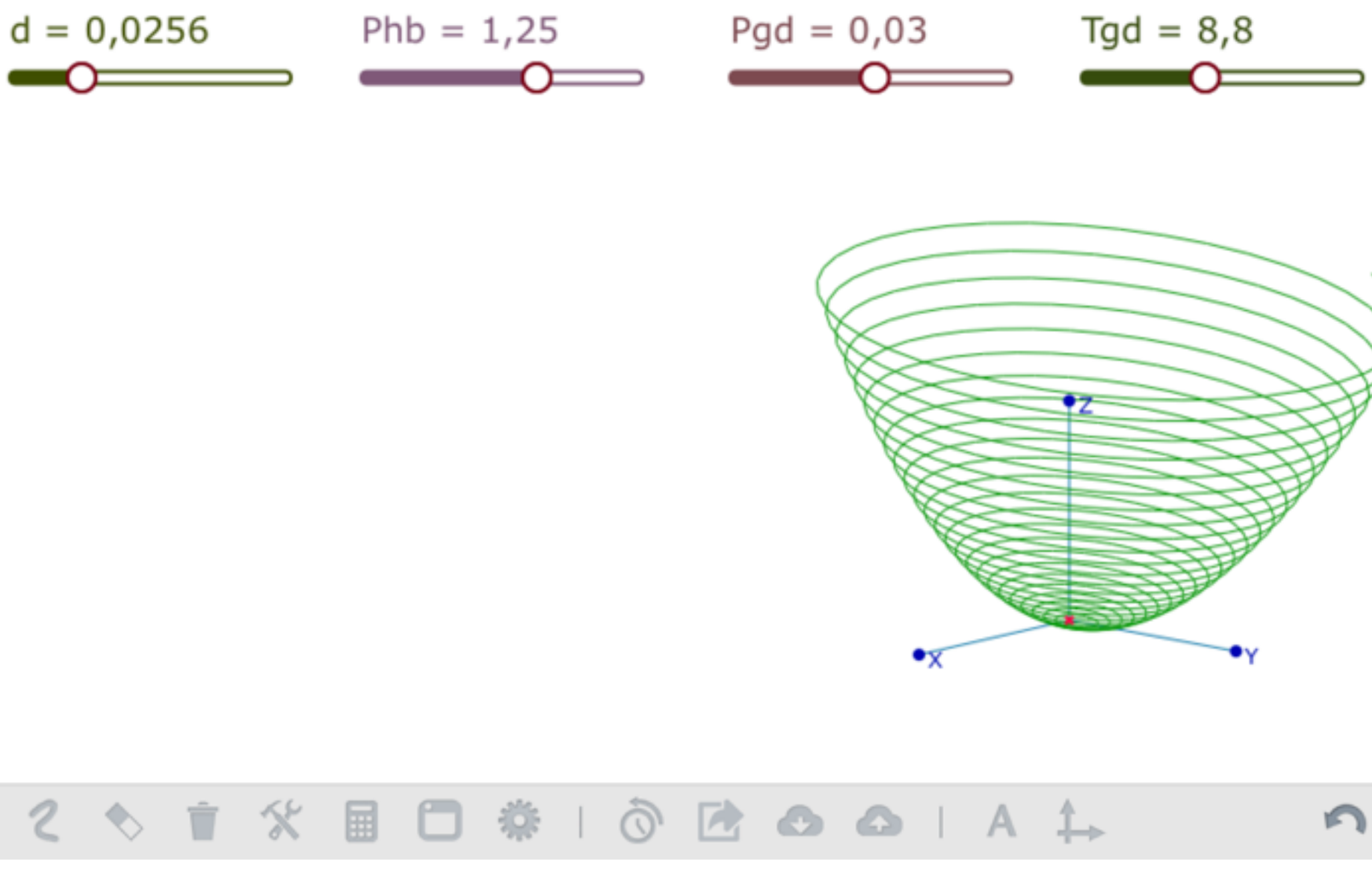


Figure dynamique 7.1 – Explorer les outils de la tortue 3D avec des curseurs



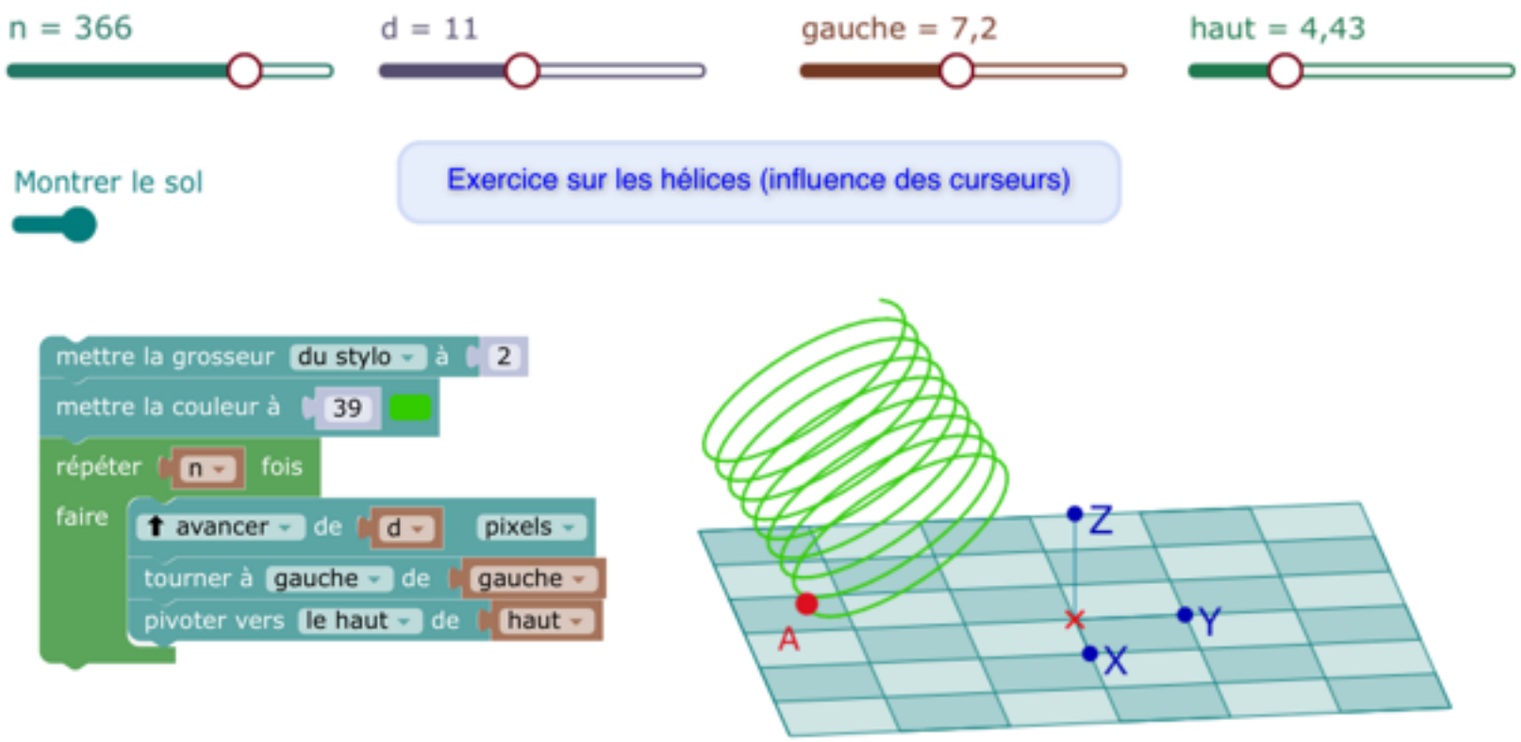
*Le code est dans l'origine du repère (être en mode **standard**). Il est plus facile de manipuler le trièdre en mode **consultation**.*

Quelles sont nos capacités d'anticipation du résultat dans un repère mobile 3D ?

Nous poursuivons sur une figure cette fois de type **hélice**, avec deux instructions : une instruction du plan (**tourner**) et une de l'espace (**pivoter**).

Tester les influences des deux paramètres.

Nous vous invitons ensuite à faire la modification suivante, et à tester quelques variantes (placer une instruction avant le avancer par exemple).



Encadrer une instruction 2D de la même instruction 3D...

... revient à définir un plan (avec une boucle). Tester alors l'influence des paramètres. Dans la page suivante, l'instruction est déjà placée mais inactive, il suffit de l'activer. Dans les pages suivantes, on justifie le résultat.

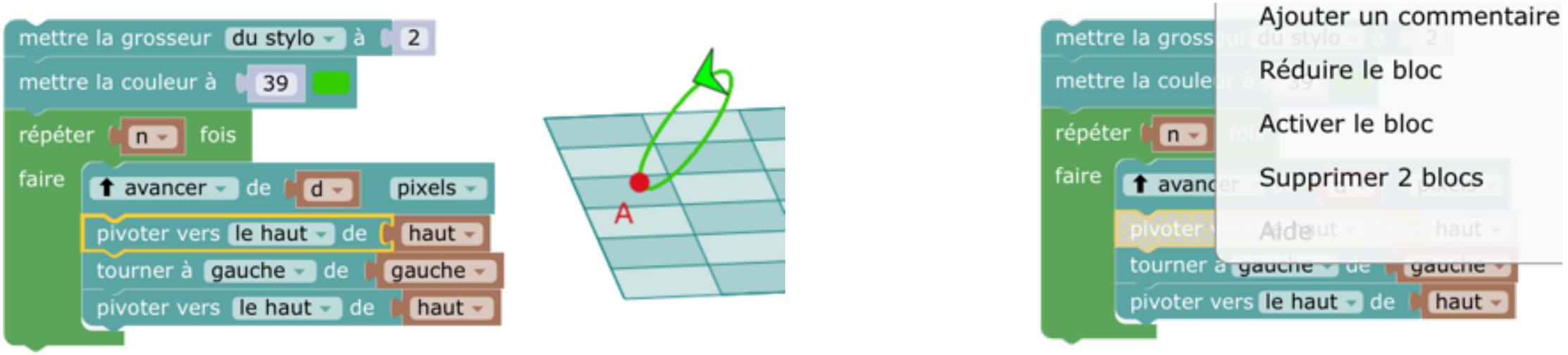
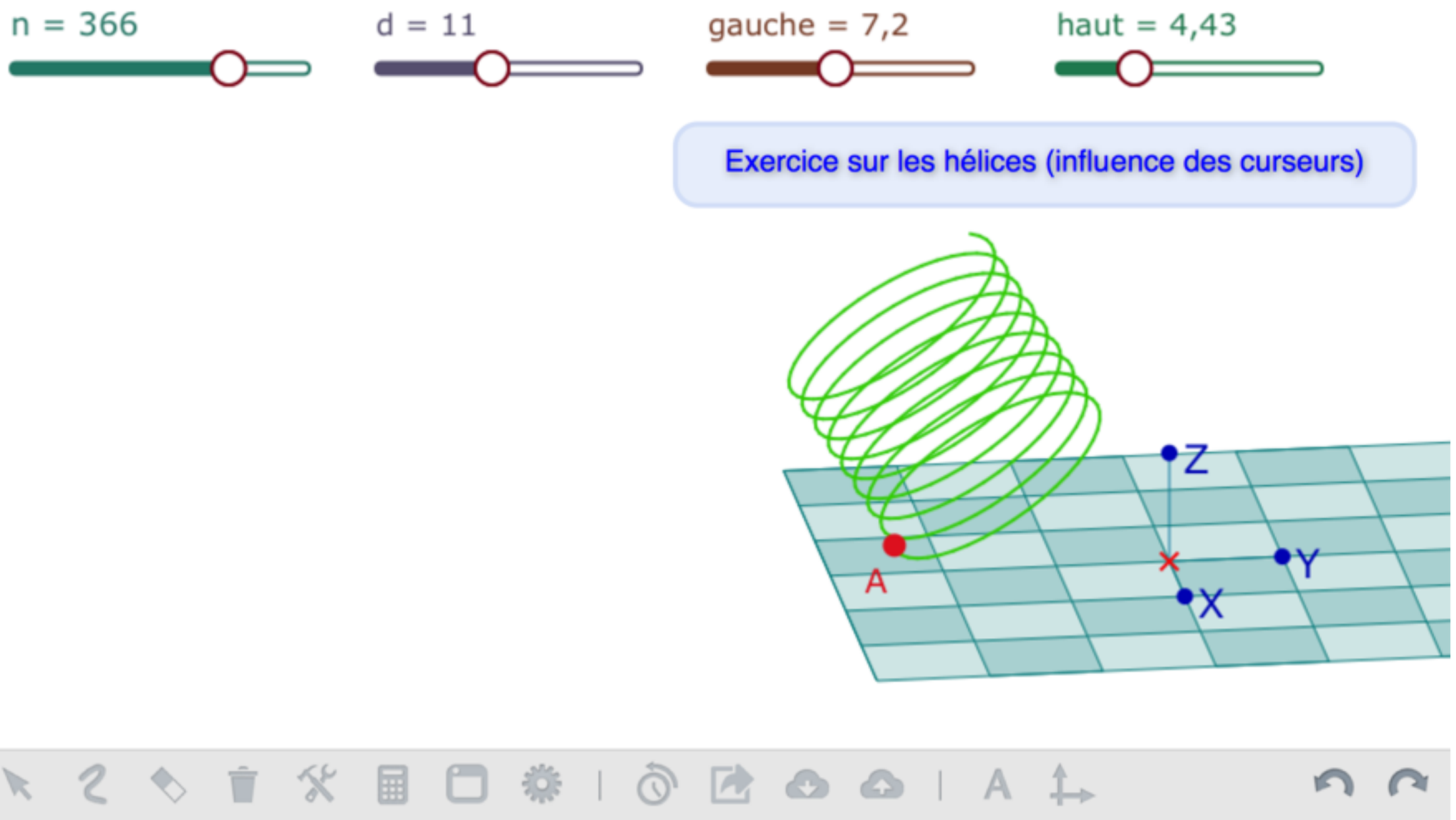


Figure dynamique 7.2 – Explorer les items 3D sur une figure élémentaire – Hélice



Le code est en A. Activer le code inactif. On peut ensuite déplacer des blocs et les activer/désactiver.
*Ne pas oublier la dualité mode **standard** (pointeur actif) / mode **consultation** (sans outil actif).*

Justification de la spirale 3D se réduisant dans un plan dans la manipulation précédente et construction du plan

On peut être surpris du résultat. Est-on vraiment dans un plan, ou bien est-ce une approximation ? Et si oui quelle est sa direction ? Merci à Dominique Tournès, le directeur de notre IREM, de s'être penché sur cette question. Voici son analyse.
On se place dans le repère initial de la tortue, en A, avec ses directions $\left(\vec{i}, \vec{j}, \vec{k}\right)$.

La tortue avance d'abord de d dans la direction $\vec{i}$, et donc fait une translation de vecteur $d\vec{i}$. Elle fait ensuite successivement trois rotations «sur place» soit donc trois rotations vectorielles. La première, d'angle de mesure $-h$, autour de $\vec{j}$, transforme le repère $\left(\vec{i}, \vec{j}, \vec{k}\right)$ en $\left(\vec{i_1}, \vec{j_1}, \vec{k_1}\right)$ par la matrice $M_1 = \begin{pmatrix} \cos h & 0 & -\sin h \\ 0 & 1 & 0 \\ \sin h & 0 & \cos h \end{pmatrix}$. La deuxième, d'angle de mesure g, autour de $\vec{k}_1$, transforme $\left(\vec{i_1}, \vec{j_1}, \vec{k_1}\right)$ en $\left(\vec{i_2}, \vec{j_2}, \vec{k_2}\right)$ par la matrice $M_2 = \begin{pmatrix} \cos g & -\sin g & 0 \\ \sin g & \cos g & 0 \\ 0 & 0 & 1 \end{pmatrix}$. Enfin, la troisième, d'angle de mesure $-h$, autour de $\vec{j_2}$ transforme $\left(\vec{i_2}, \vec{j_2}, \vec{k_2}\right)$ en $\left(\vec{i_3}, \vec{j_3}, \vec{k_3}\right)$ par la matrice initiale M_1. La composition des trois, le **palindrome** $M_1M_2M_1$, a pour matrice

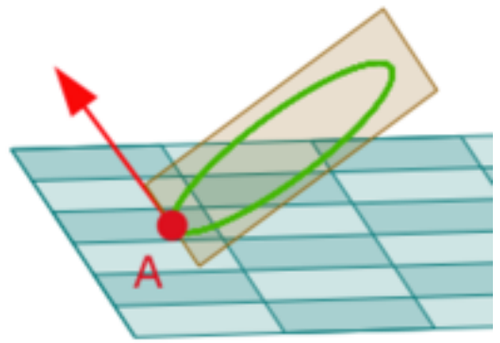
$$M = \begin{pmatrix} \cos^2 h \,.\, (1+\cos g) - 1 & -\sin g \cos h & -\sin h \cos h \,.\, (1+\cos g) \\ \cos h \sin g & \cos g & -\sin h \sin g \\ \sin h \cos h \,.\, (1+\cos g) & -\sin g \sin h & 1 - \sin^2 h \,.\, (1+\cos g) \end{pmatrix}$$, expression de $\left(\vec{i_3}, \vec{j_3}, \vec{k_3}\right)$ rapporté à $\left(\vec{i}, \vec{j}, \vec{k}\right)$.

C'est une isométrie vectorielle positive, de direction propre la droite vectorielle $\begin{cases} X = 0 \\ (\cos g - 1)Y - \sin h \sin g Z = 0 \end{cases}$.

C'est donc une rotation vectorielle autour de l'axe $\left(0, \sin h \sin g, \cos g - 1\right)$. Le plan vectoriel P orthogonal à l'axe contient $\vec{i}$ et est stable par la transformation. La tortue avance donc dans le plan de direction P passant par A.
Après un mouvement élémentaire, son orientation est donnée par le vecteur $\vec{i_3}$, qui appartient à P comme image de $\vec{i}$.
On se retrouve alors dans la situation initiale et donc, par itération, la tortue ne sort pas du plan de direction P passant par A.

L'extrémité du vecteur normal au plan, en A est donné par l'expression suivante, le coefficient 100 étant arbitraire

```
[X,Y]= A-100*[0,sin(haut)*sin(gauche),cos(gauche)-1]
```

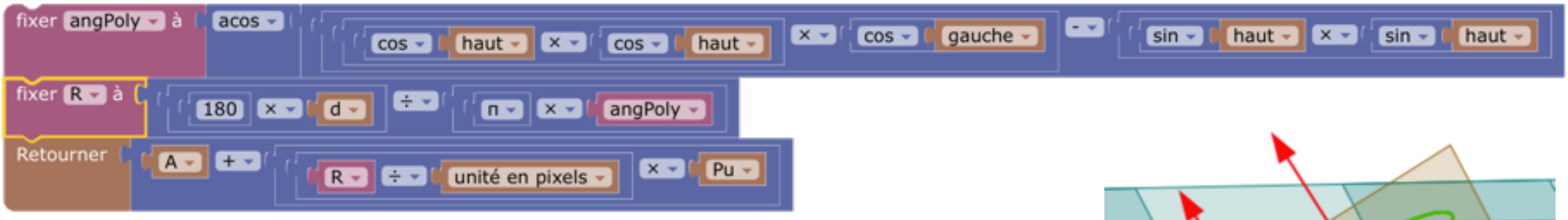


Centrer le vecteur normal

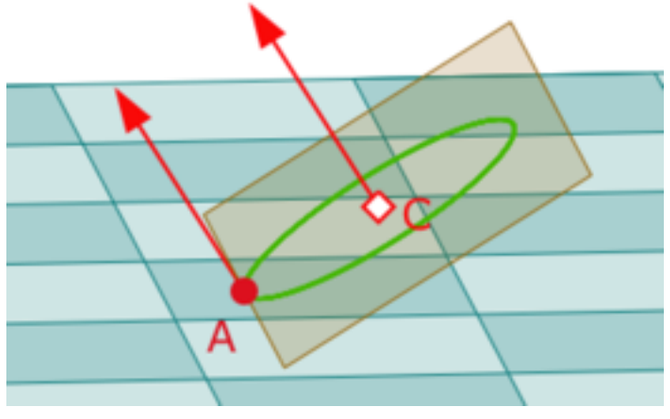
La trace de la tortue n'est bien-sûr pas un cercle, seulement une ligne brisée. C'est la valeur du paramètre **d**, petite au regard de la finesse de la résolution des écrans, qui peut donner l'illusion d'un cercle. Travaillant sur les spirolatères différentiels au moment de la première publication de ce livre numérique, c'est à Patrice Debrabant – plusieurs mentionné dans cet ouvrage – que l'on doit d'avoir finalisé cette question. Il s'agit de calculer une valeur approximative du rayon de cette ligne brisée.

La composante $\vec{i}$ du vecteur $\vec{i_3}$ peut aussi s'écrire $(\cos^2 h)(1+\cos g)-1=\cos^2 h\cos g-\sin^2 h$. Et donc l'angle de la ligne brisée est $\alpha=\cos^{-1}\left(\cos^2 h\cos g-\sin^2 h\right)$. Et ainsi, le rayon du cercle approché est $R=\frac{360}{\alpha}.\frac{d}{2\pi}=\frac{180d}{\alpha\,.\,\pi}$ si l'angle est en degrés.

Il faut toutefois adapter l'unité puisque d est donnée en pixels, cela peut se faire ainsi, en Blockly : on remarquera le bloc qui permet d'utiliser l'unité en pixels (**Pu** est le vecteur unitaire dans la direction qu'il convient).

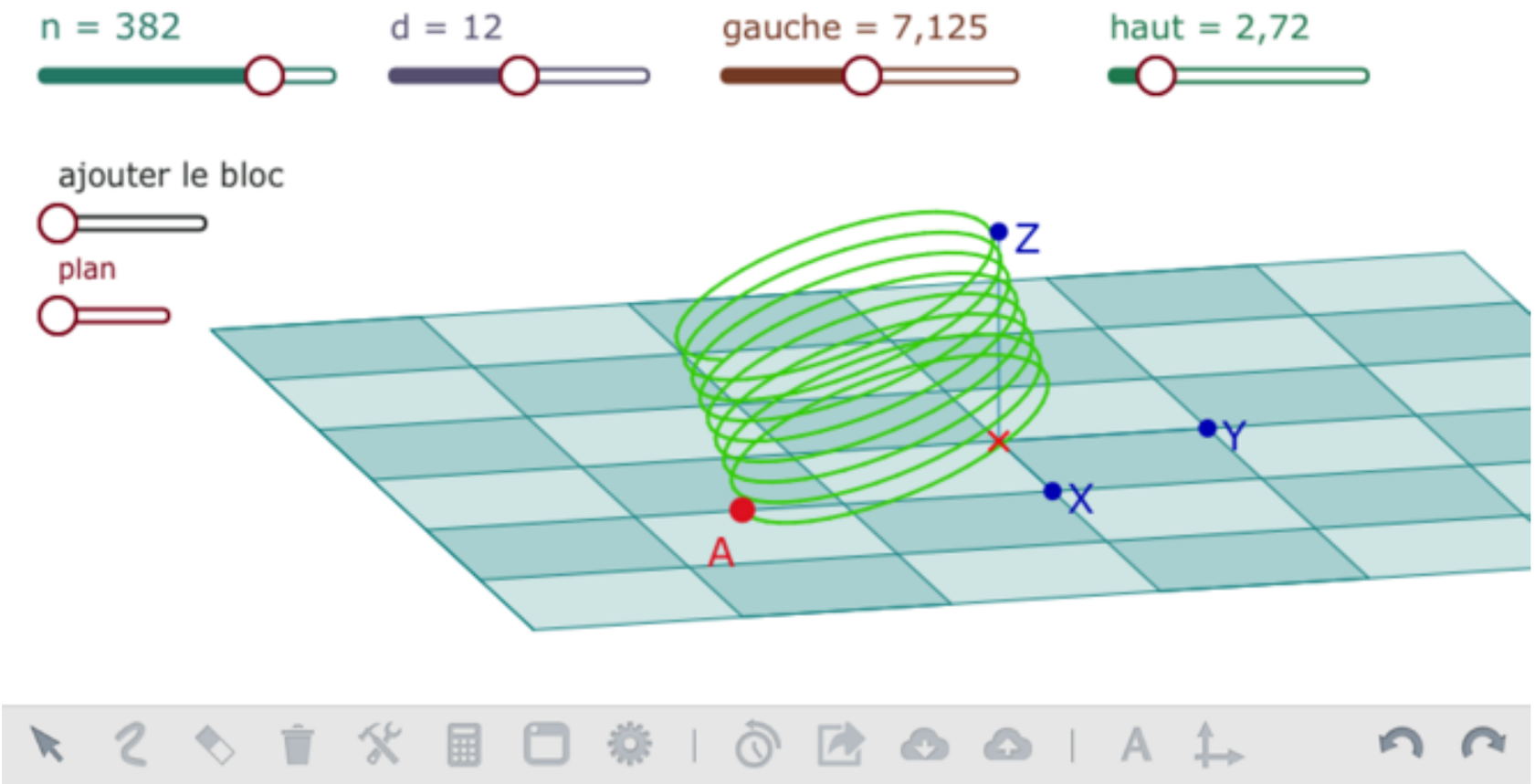


Dans un autre très bel article, de août 2017, sur les spirolaèdres, Patrice précise que c'est parce que l'on applique un **palindrome** de type $T=BAB$ à la tortue 3D que l'itération de la transformation reste coplanaire. C'est parce que $T^{-1}=B^{-1}A^{-1}B^{-1}$ utilise la même suite de rotations (même ordre), avec des angles opposés que le résultat est coplanaire, ce qui ne peut être le cas avec $T=AB$ et $T^{-1}=B^{-1}A^{-1}$.



Dans cette figure, on évite au lecteur d'ajouter lui-même le bloc qui produit un palindrome. Cela a déjà été fait dans la figure précédente pour expérimenter soi-même le résultat. Aussi cette fois, on le fait plus simplement en manipulant le curseur « **ajouter le bloc** ». De même le plan de réduction et le centre approximatif de la ligne brisée s'obtiennent par le curseur « **plan** ». Penser à jouer aussi sur les autres curseurs.

Figure dynamique 7.3 – Visualisation du plan de réduction des manipulations précédentes.



Introduction scolaire à la tortue 3D

On reprend l'activité proposée par Eric Hakenholz sur sa chaine YouTube pour introduire les outils 3D. On y a juste ajouté des curseurs. Il s'agit d'ouvrir un livre de format carré.

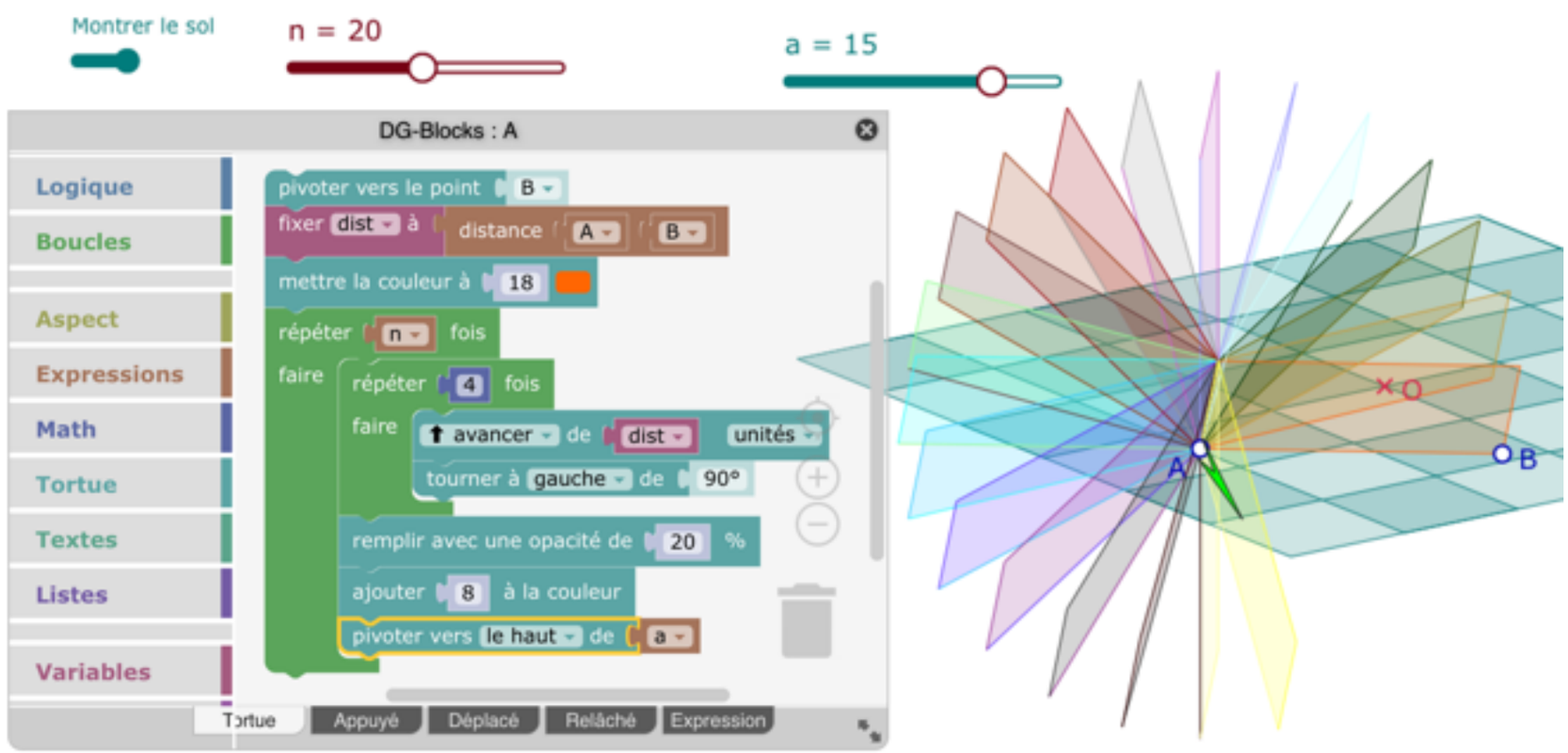
Pour cela on utilise une seule instruction 3D : **pivoter vers le haut**.

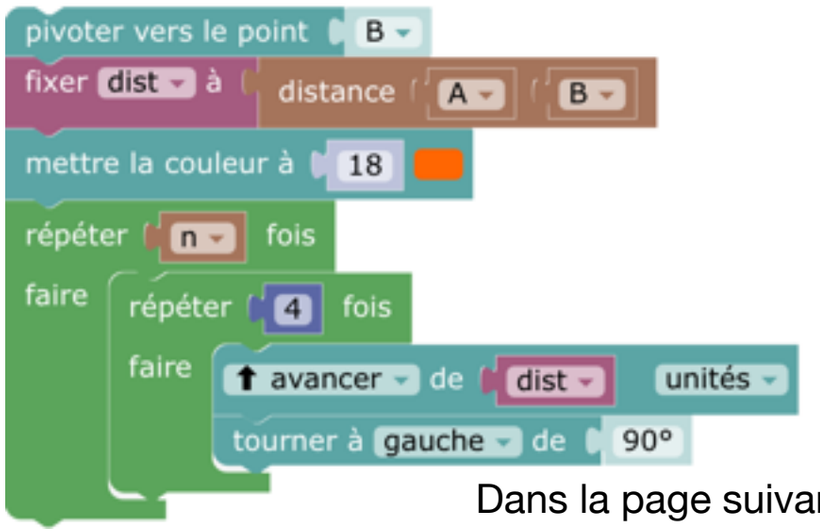
Dans le début de la construction ci-dessous, on a répété n fois le même carré de côté [AB].

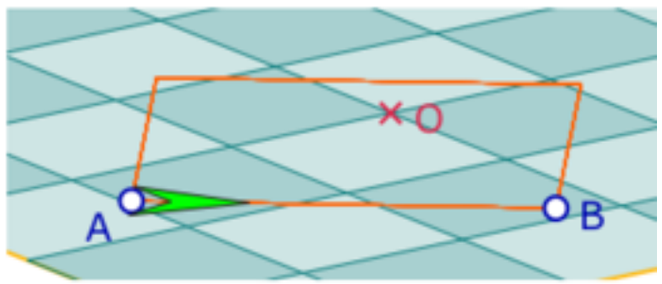
On remarque que la tortue est sur une arête.

Soit on tourne selon l'arête [AB] et on utilisera un **pivoter vers la droite**.

Soit on tourne selon l'arête précédente, dont la tortue est actuellement orthogonale et on utilisera un **pivoter vers le haut**.

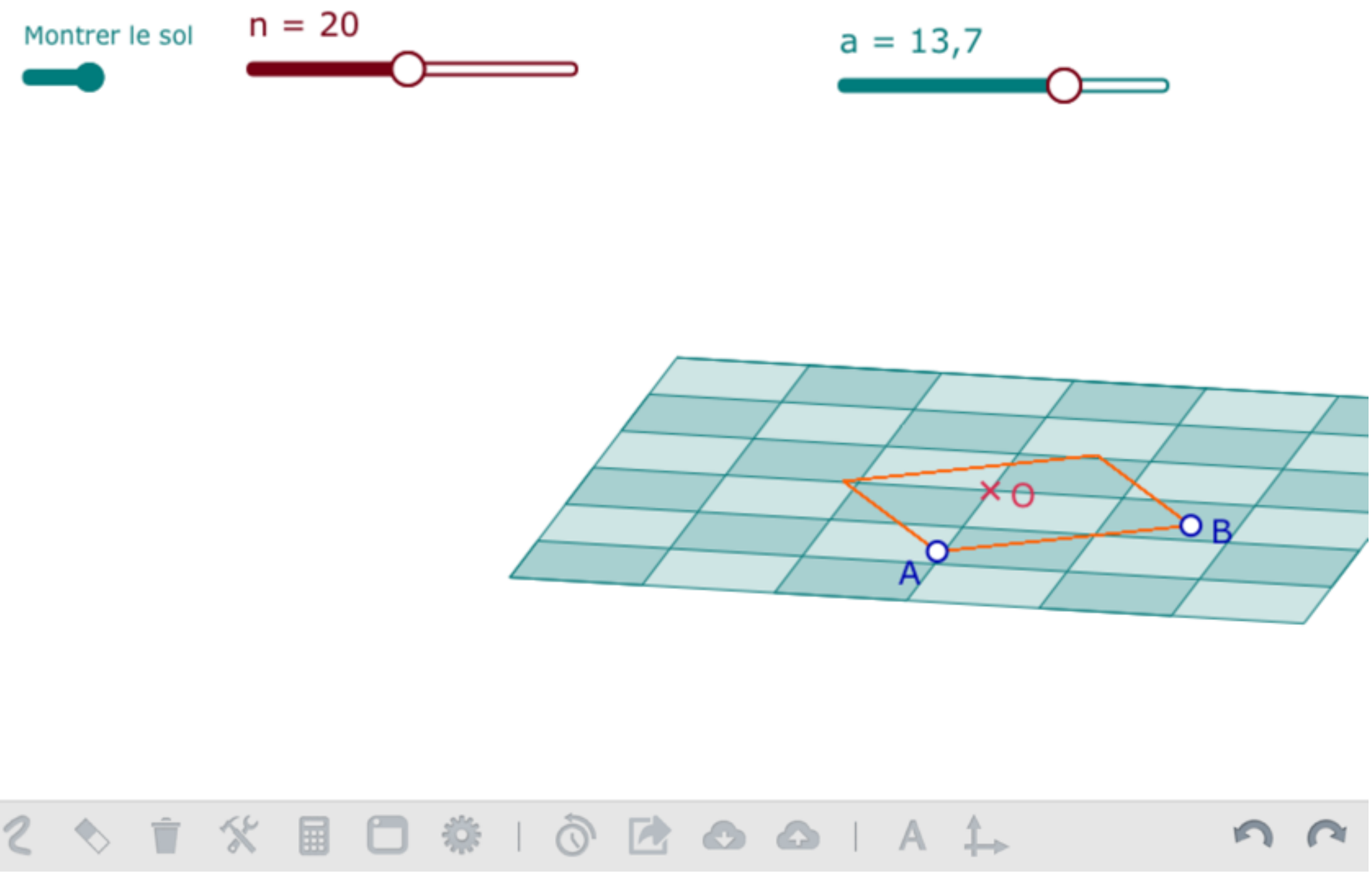






Dans la page suivante **Activer** une à une les 3 lignes désactivées, puis **modifier l'orientation de la rotation** dans le **pivoter**.

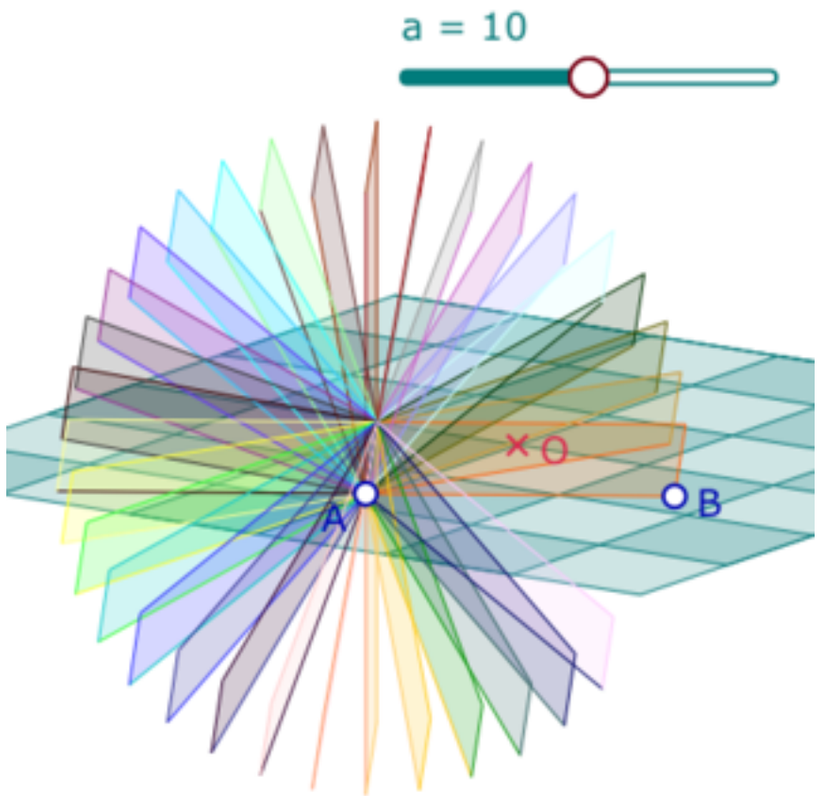
Figure dynamique 7.4 – Pratique initiale de la 3D – Ouverture d'un livre

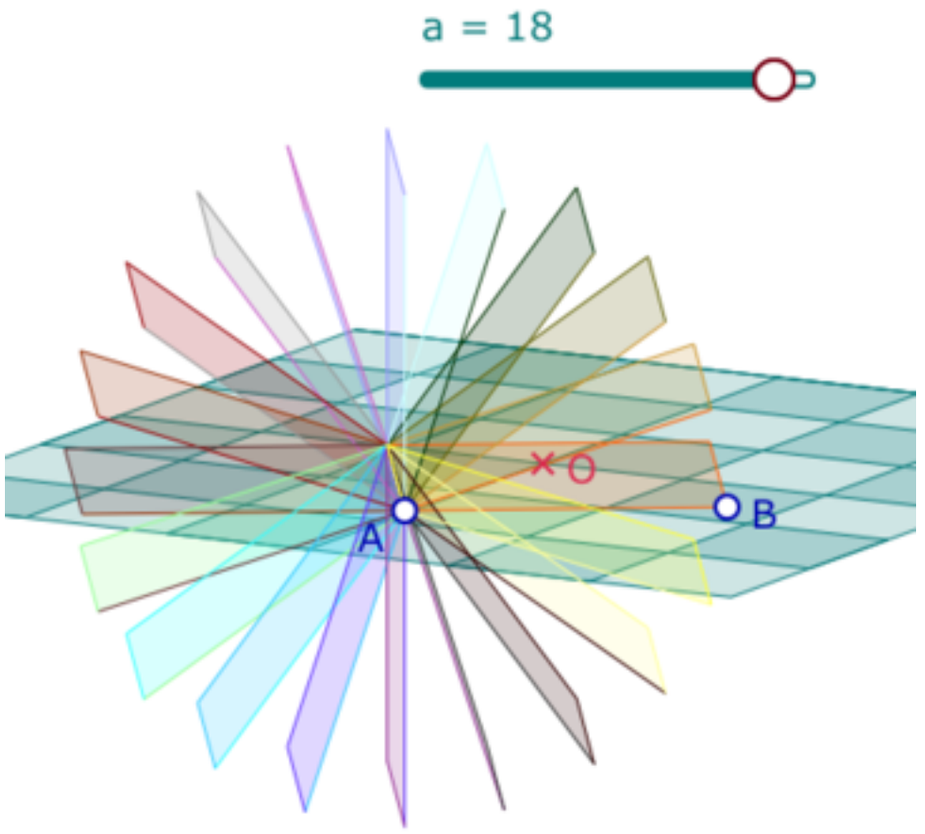


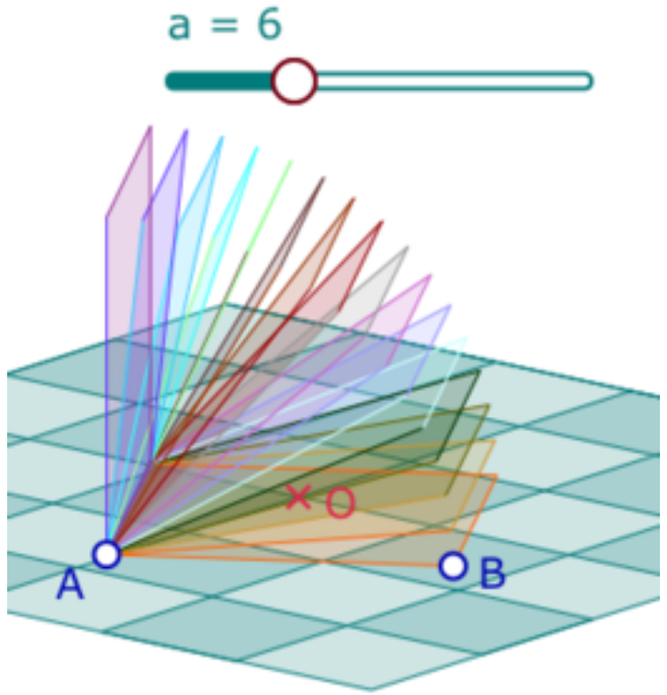
Le code est en A. Activer les trois lignes désactivées. Modifier la direction du pivot (de haut par droite).

Utilisation arithmétique de cette activité

On peut s'intéresser aux angles qui permettent qu'une feuille soit perpendiculaire au sol. Comment le visualiser ? Voici quelques exemples.







Les activités proposées dans la suite de ce chapitre

Trois polyèdres à construire de manière filaire : la tortue ne trace alors pas nécessairement des faces coloriables.

Patrons pliables du cube. Cette fois en traçant complètement les faces, que l'on peut ainsi colorier.

Compléments (non scolaires) sur d'autres patrons par **pivoter gauche**.

Réalisation d'un cube à la tortue DGPad

Cube par 4 faces

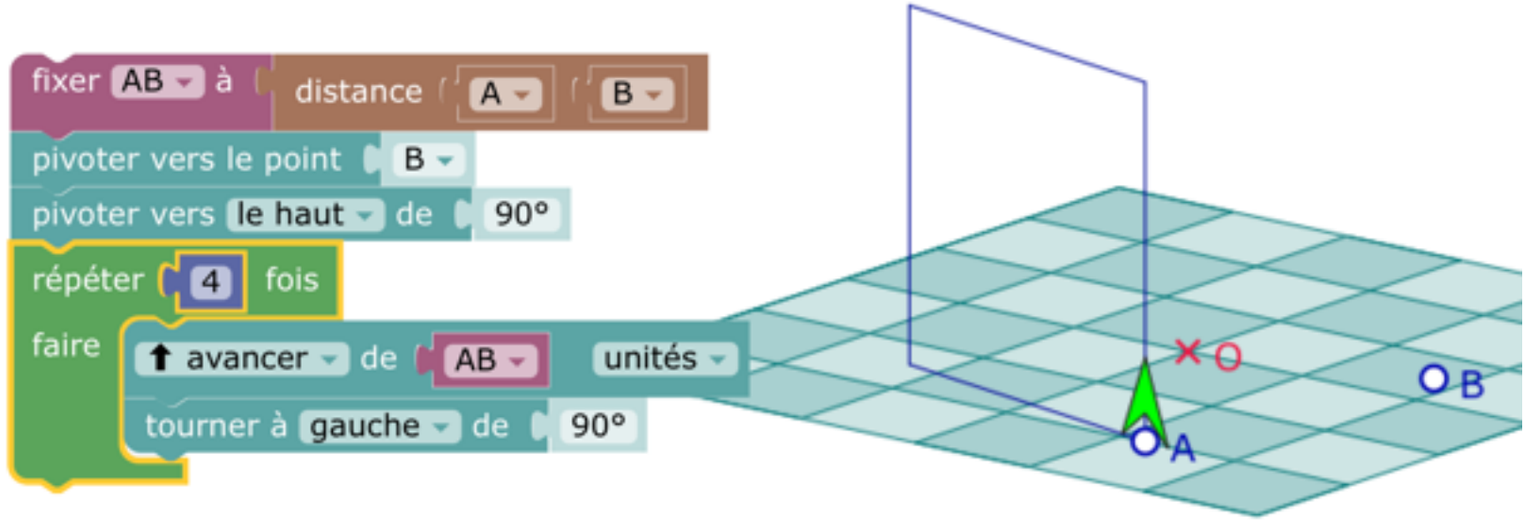
Ci-contre on se donne deux points A et B du plan du sol, et on commence par construire une face verticale, celle orthogonale à [AB] en A.

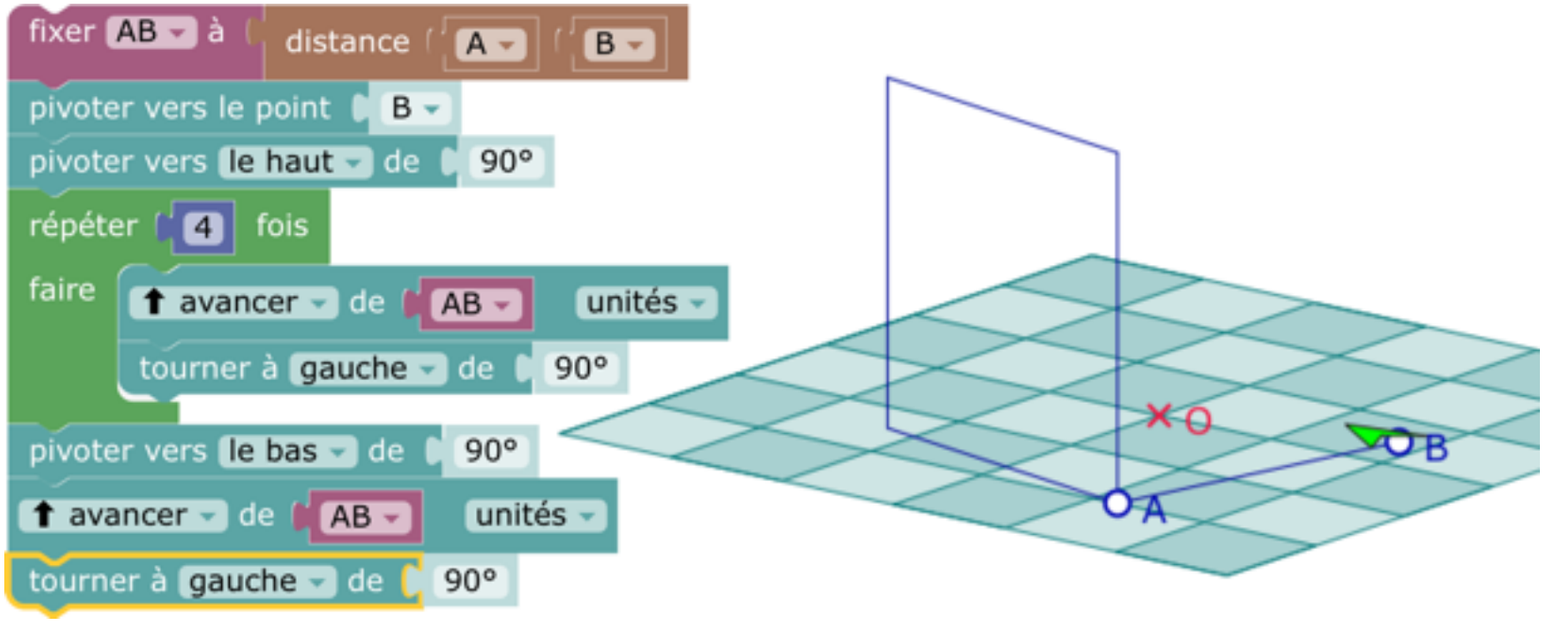
On va ensuite reproduire 4 fois cette construction pour dessiner le cube à partir de 4 faces seulement.

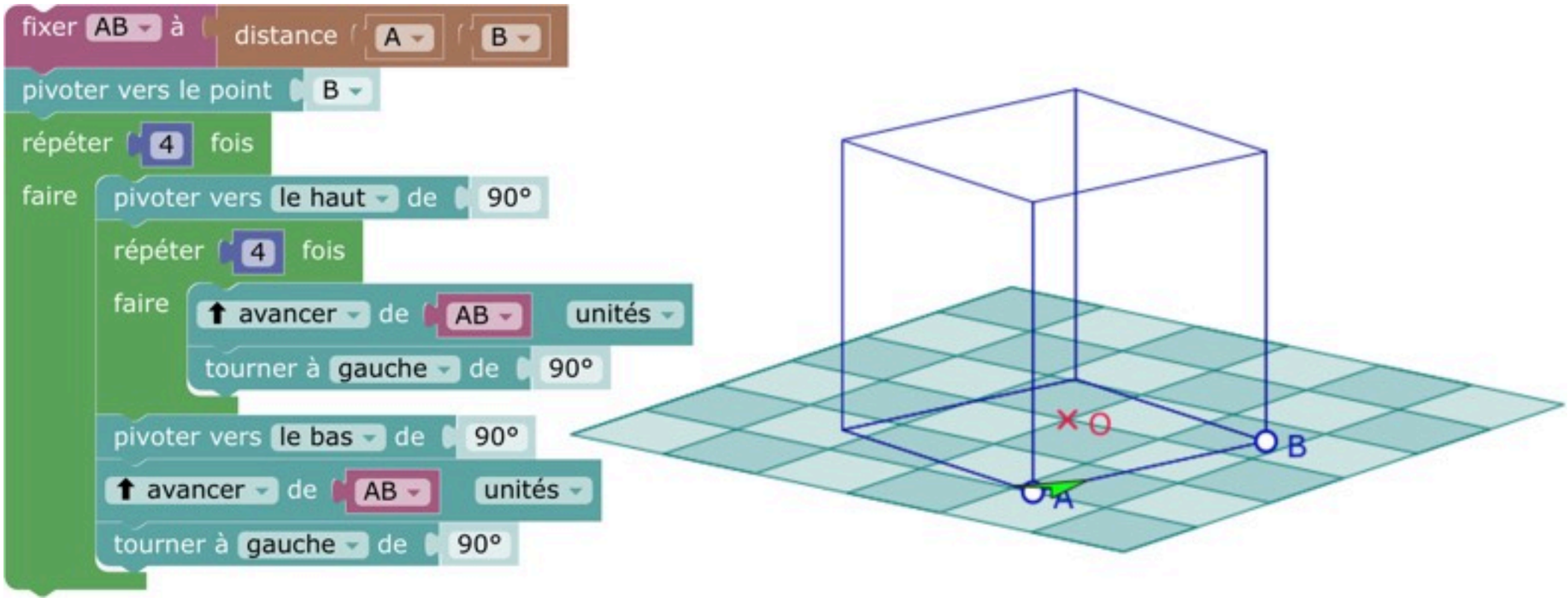
On voit bien qu'en faisant cela on va dessiner les 12 arêtes, mais on n'aura pas les 6 faces au sens de 6 carrés que l'on pourrait remplir.

Pour cela, à partir de l'étape initiale, il suffit d'ajouter les trois lignes ci-contre pour se retrouver comme dans la condition initiale des deux premières lignes : juste après la direction vers B.

On peut alors faire une boucle de cette construction (page suivante).







Finalement on dessine 20 arêtes, pour 12 nécessaires. Mais il faudrait 24 arêtes pour avoir 6 carrés complets.

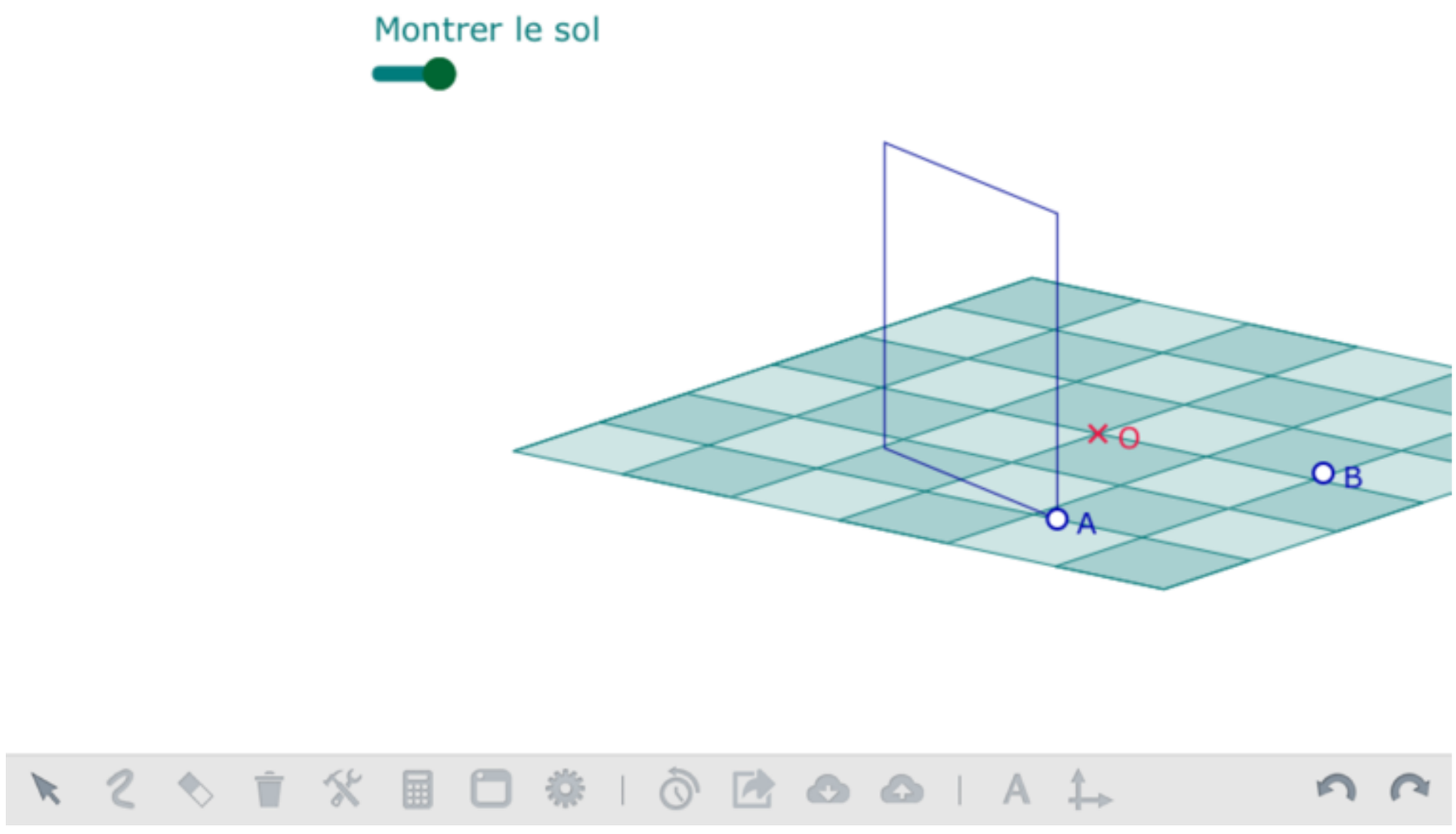
On peut demander aux élèves quelles sont les 8 arêtes tracées en deux fois.

Dans la figure suivante, seule la phase du premier carré est construite. Vous êtes invité à

1- achever la construction comme ici.

2 - remplir les faces pour voir que seules 4 des 6 faces sont entièrement construites.

Figure dynamique 7.5 – Cube par 4 faces – Figure à compléter

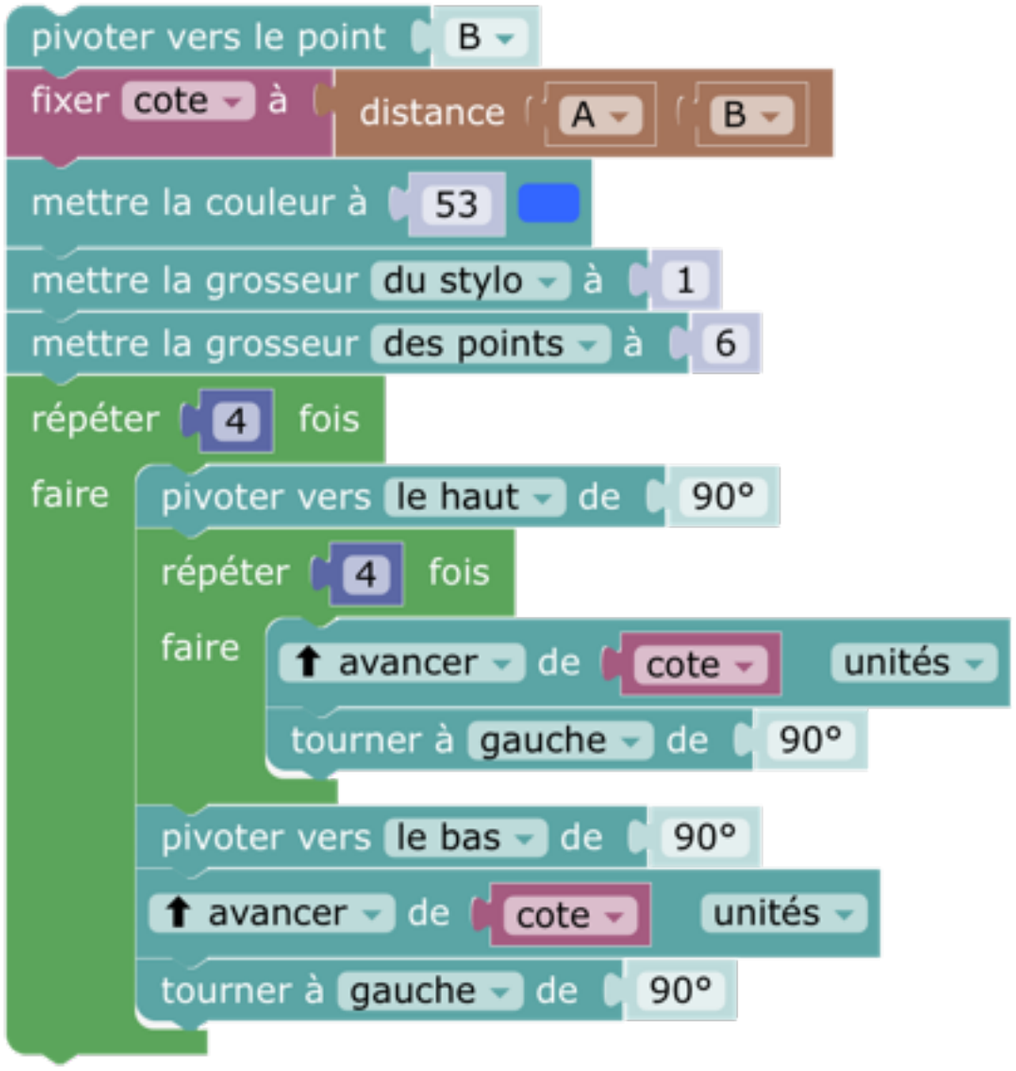


1 - Compléter par les 3 lignes manquantes puis achever par une boucle de 4 telles faces.
*2 - Remplir à 20 % les carrés et manipuler la figure (de préférence en mode **consultation** en désactivant le pointeur) pour voir que le cube a deux trous en haut et en bas.*

Une illustration de la spécificité de l'orientation de la tortue en 3D

Dans la figure suivante, on se donne deux points A et B sur deux plans orthogonaux, dont A sur le plan du sol. Alors, l'orientation de la tortue vers B oriente aussi le plan dans lequel elle va se déplacer.

Il en résulte que le même code construit un cube d'arête [AB] dont la face de base comporte l'autre arête issue de A dans le plan du sol.



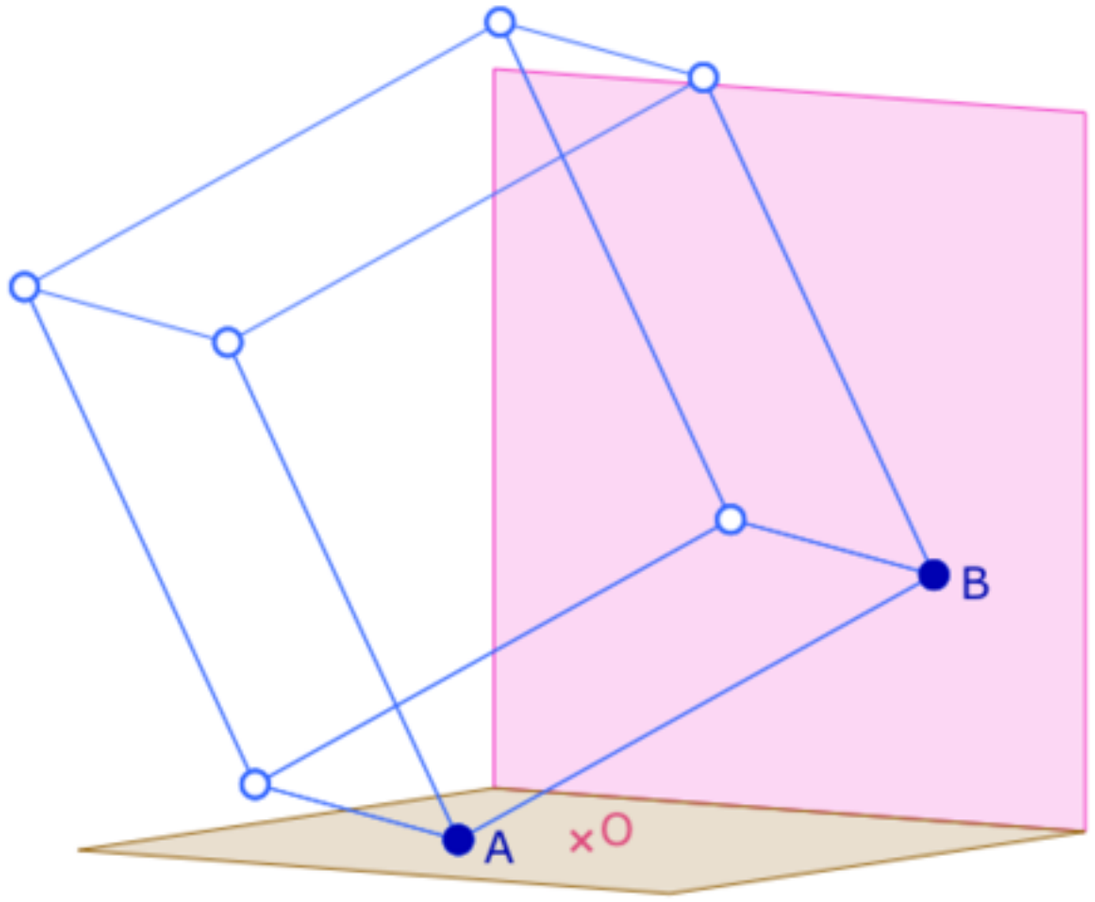
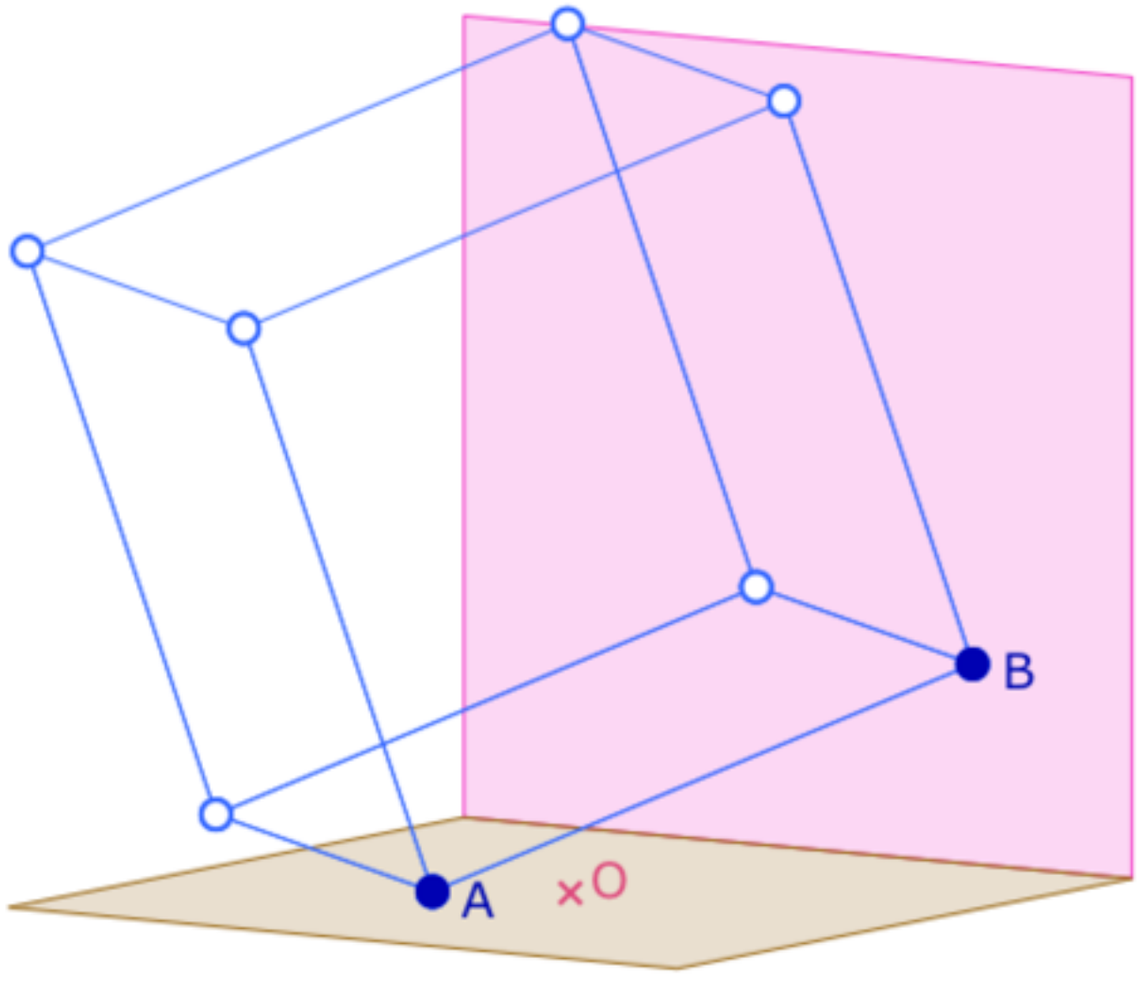


Figure dynamique 7.6 – Application du code précédent avec deux sommets dans deux plans orthogonaux



*Par défaut, la figure est en mode **consultation** pour faire tourner le trièdre et voir qu'une arête issue de A est dans le plan du sol.*
*Activer le mode **standard** (par le pointeur) pour voir le code placé dans le point A.*

Du cube au cuboctaèdre

Cuboctaèdre par ses 6 faces carrées, premier exemple de code optimisé

A la sortie du cube de la construction précédente, la tortue est en A sur l'arête [AB] en direction de B. On peut facilement faire le carré des milieux de la face verticale de côté [AB], c'est une face carré du cuboctaèdre.

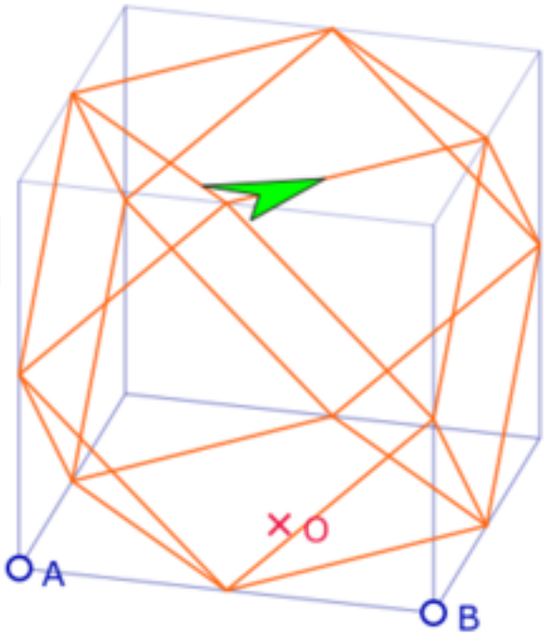
Les premières lignes du code à droite tracent le cube et positionnent la tortue au milieu de l'arête [AB].

Voyons maintenant le début du code pour le cuboctaèdre.

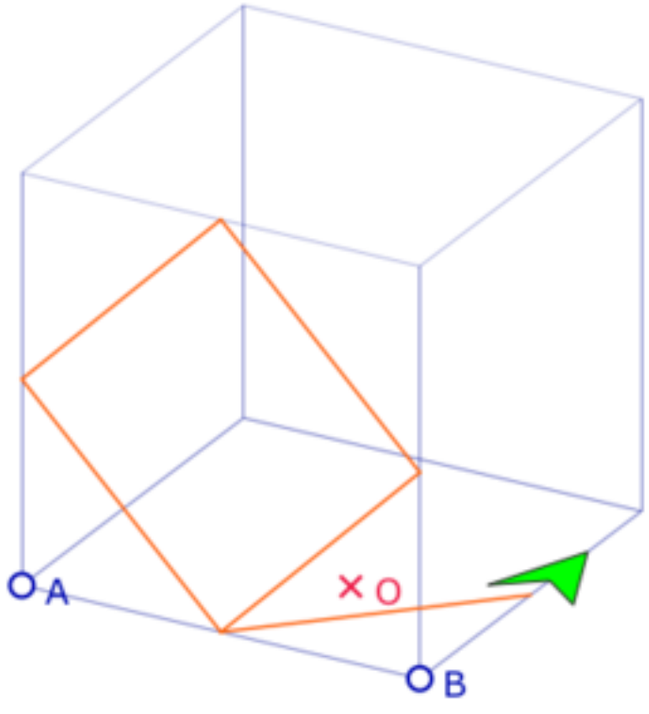
Pour cette illustration, on a mis la boucle **répéter** à une seule itération pour voir que la démarche peut être optimisée : le cuboctaèdre a 24 arêtes et nous allons construire ces 24 arêtes *une seule fois*. On commence donc par tracer un carré et une arête du carré du sol. Répétant ceci 4 fois, on construit 5 des 6 carrés et 20 des 24 arêtes.

Il suffit d'ajouter un carré sur la face du haut (code non reproduit ici).

```
fixer AB à distance A B
pivoter vers le point B
mettre la grosseur du stylo à 0.3
Cube avec : cote AB
fixer cCubo à AB ÷ racine carrée 2
lever le stylo
avancer de AB ÷ 2 unités
poser le stylo
mettre la grosseur du stylo à 1.5
mettre la couleur à 18
Cuboctaedre avec : cote cCubo
```

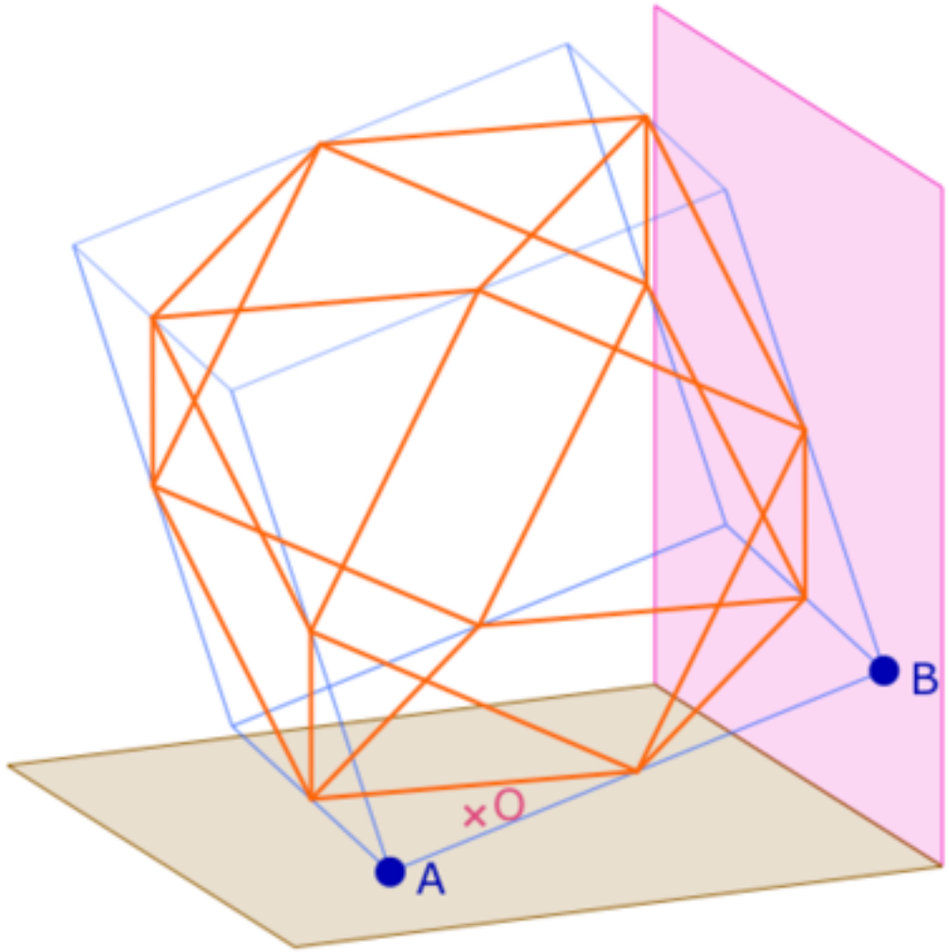



```
pour Cuboctaedre avec : cote
répéter 1 fois
faire
    pivoter vers le haut de 45°
    répéter 4 fois
    faire
        avancer de cCubo unités
        pivoter vers le haut de 90°
    pivoter vers le bas de 45°
    tourner à gauche de 45°
    avancer de cCubo unités
    tourner à gauche de 45°
```



Comme pour le cube, le même code s'applique à deux points A et B dans deux plans orthogonaux.

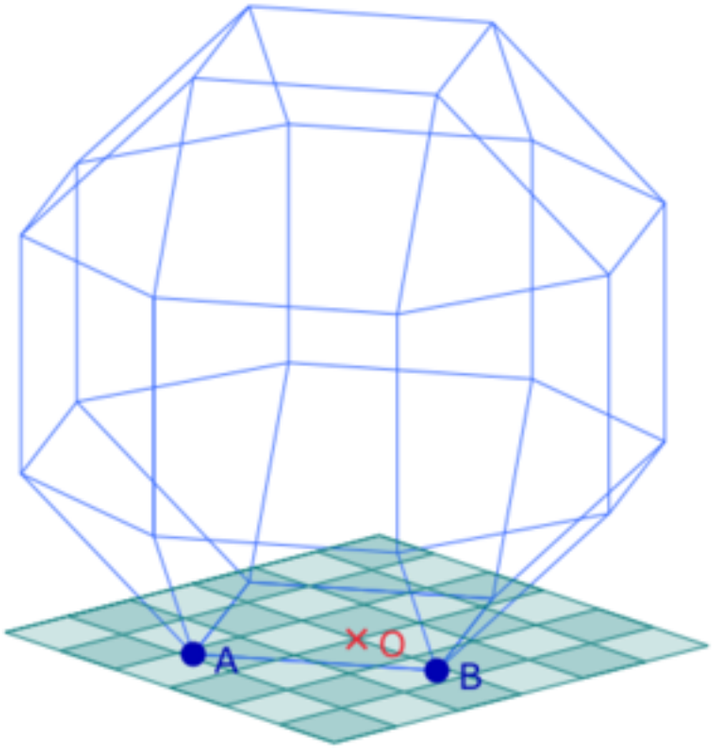
Figure dynamique 7.7 – Cuboctaèdre dans un cube d'arête [AB]



*Par défaut la figure est en mode **standard**, pour visualiser le code, placé dans le point A.*
*Désactiver le pointeur pour passer en mode **consultation** et plus facilement faire tourner le trièdre.*

Construction filaire du rhombicuboctaèdre

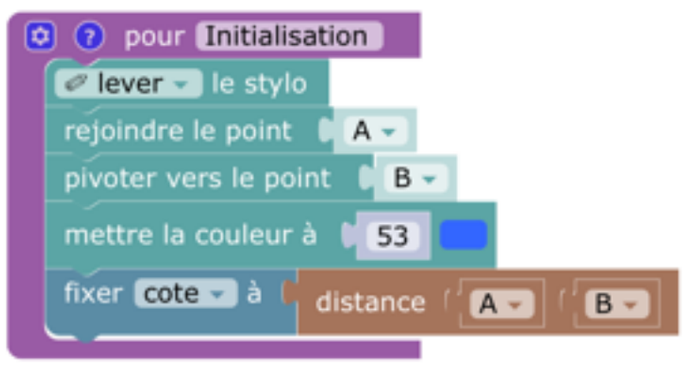
Le rhombicuboctaèdre comme 6 octaèdres réguliers

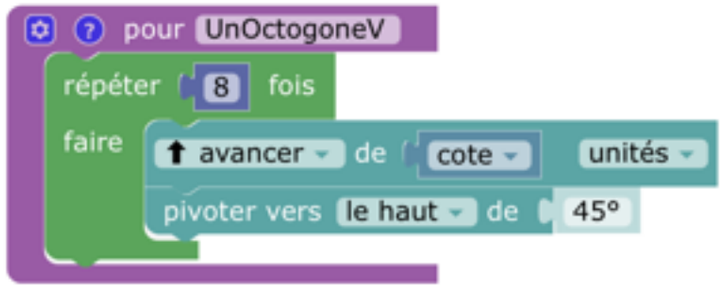


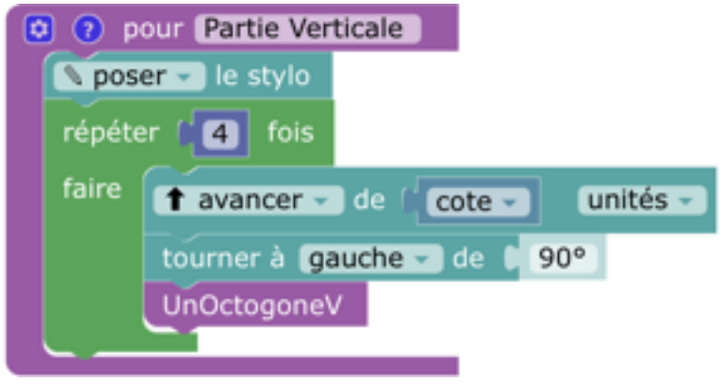
Du point de vue standard des polyèdres semi-réguliers, il est composé de **8 triangles** (un à chaque sommet d'un cube que l'on imagine tronqué) et de **18 carrés** : 6 correspondant à chacune des 6 faces d'un cube initial et 3 fois 4 carrés (ci-contre haut, milieu, bas) pour rejoindre les carrés entre eux.
Il a aussi **48 arêtes**.

Réaliser une figure – si possible un peu optimisée – à la tortue est l'occasion de déconstruire cette représentation et d'en mettre une autre à la place. Ci-contre on peut voir aussi **4 octogones réguliers verticaux**, chacun passant par une arête du carré du sol de côté [AB], et **2 octogones réguliers horizontaux**.
Les octogones étant obtenus par des **tourner de 45°**, ces constructions sont tout à fait accessibles en collège. En proposant un changement de cadre à la fois nouveau et opérationnel, on permet d'aborder autrement la géométrie dans l'espace et certains polyèdres.

Dans cette approche, on a choisi de différencier la partie verticale de la partie horizontale, pour découper la construction en étapes. On pourrait probablement conceptualiser un peu plus. En faisant 8 segments pour l'octogone, la tortue revient à sa position de départ, d'où la nécessité d'avancer dans la procédure de la **Partie Verticale**.

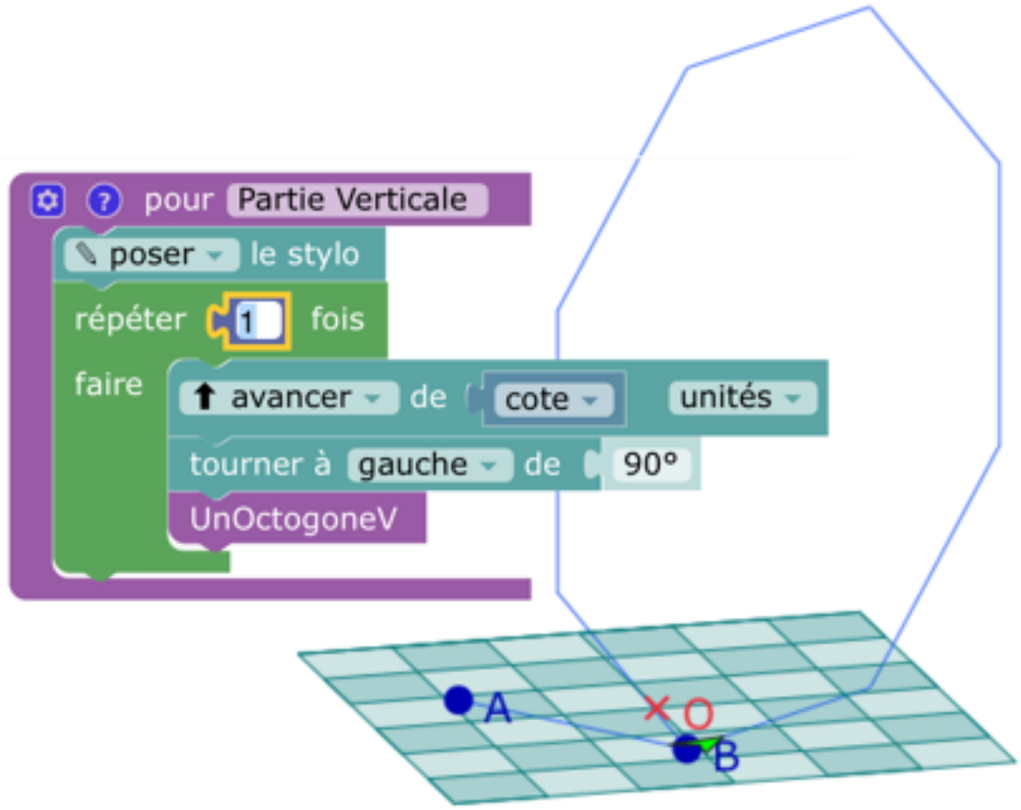






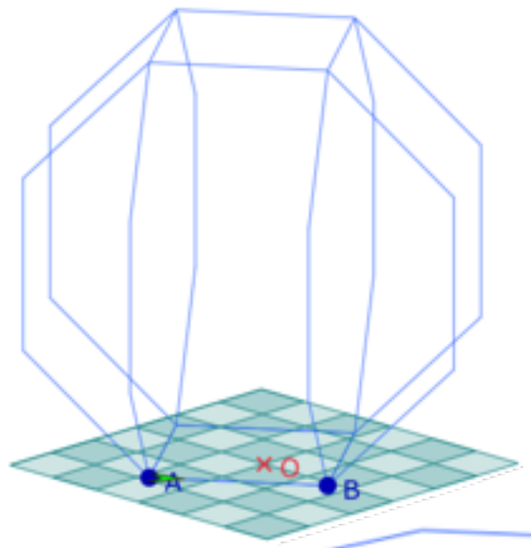
Partie verticale 1 pas

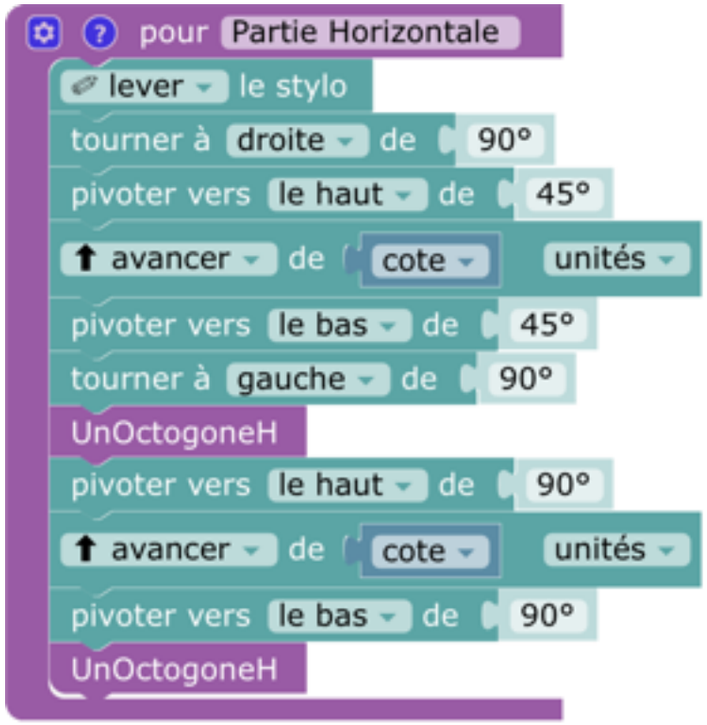
La rotation à 90° qui permet de parcourir le carré du sol et donc de tracer les 4 octogones. Le **répéter 4 fois** peut-être interprété, dans le champ du code, comme une factorisation de la tâche à accomplir.

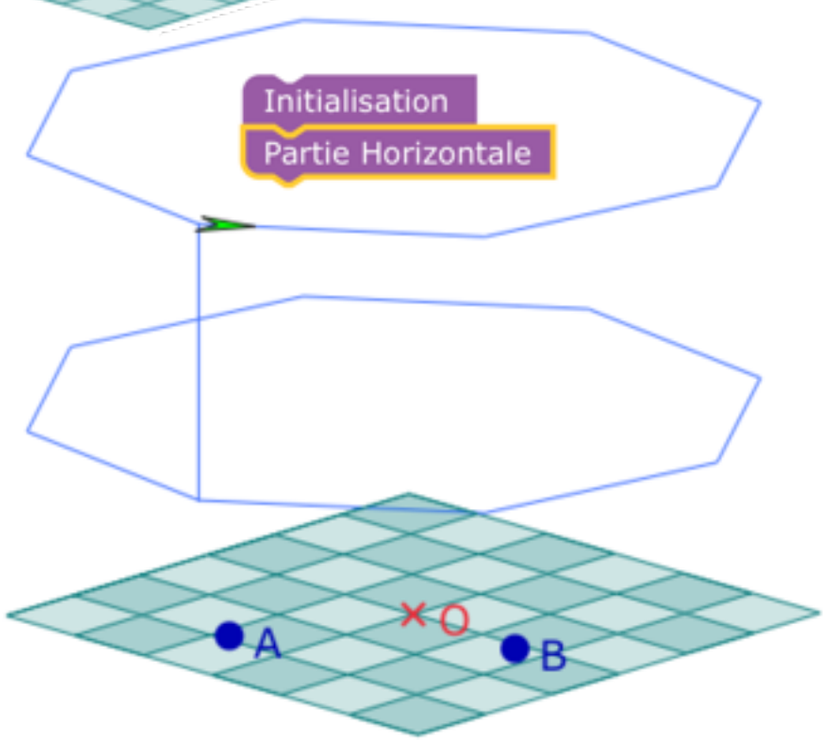


Partie horizontale

La tortue arrivant après la partie verticale (ci-contre) comme au départ, on peut même illustrer la partie horizontale seule comme ci-dessous. Il s'agit de placer la tortue au bon endroit, sans calcul de distance ,simplement en parcourant les arêtes.







Octogone horizontal

Il est dans un plan orienté comme le sol donc il utilise une commande plane **tourner à gauche**. On devrait pouvoir utiliser la même procédure pour les deux octogones.

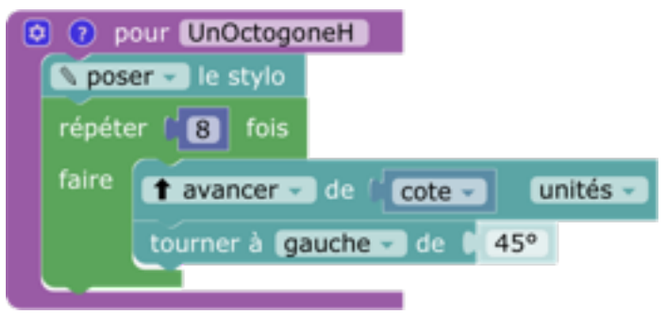
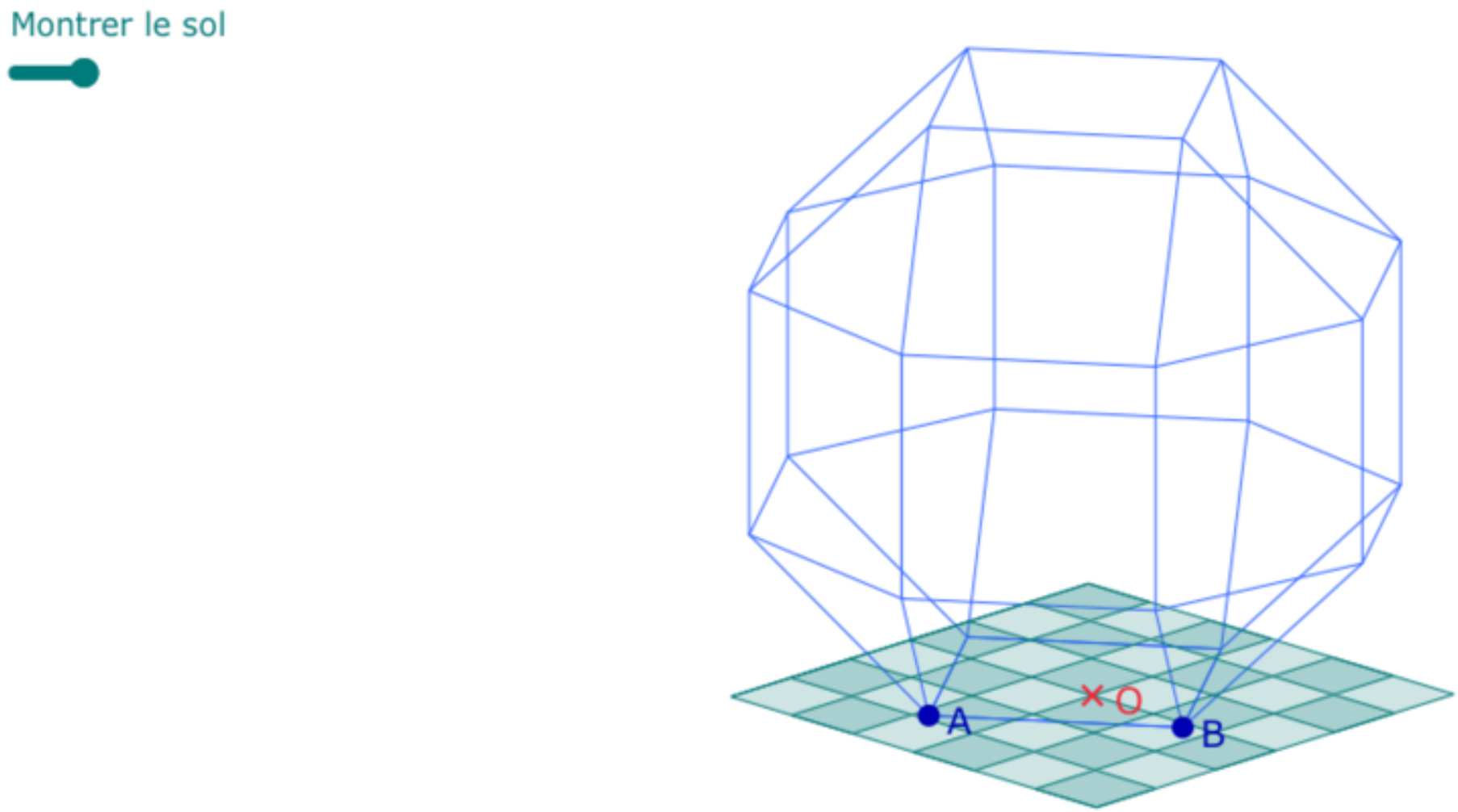


Figure dynamique 7.8 – Cuboctaèdre défini par deux points du sol A et B – Par 6 octogones

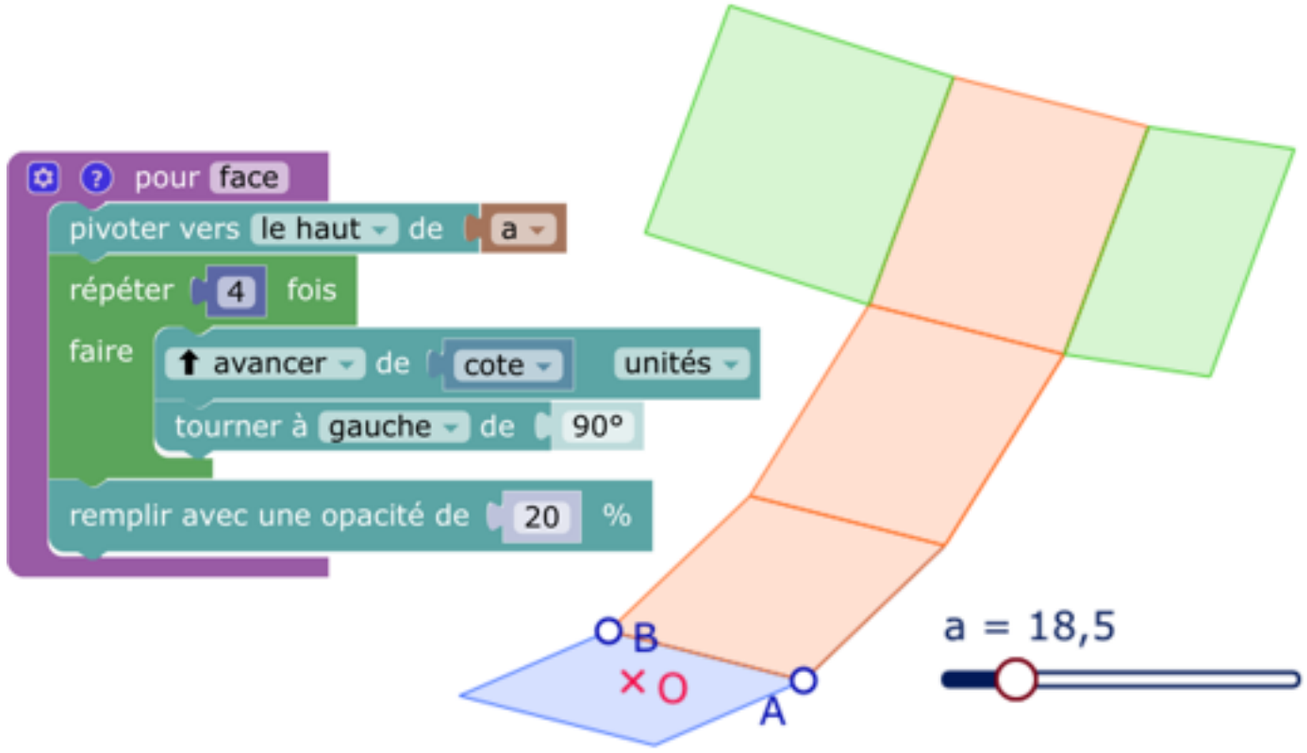


*Dans cette figure, le code est en O. Être en mode **standard** pour le code, en mode **consultation** pour modifier pour le trièdre.*

Patron du cube dynamique à la tortue

Principe général - La procédure face

On se donne un curseur pour l'angle de pliage. La procédure **face** trace un carré, la tortue reprenant la position initiale. Elle commence par changer de plan en faisant pivoter la tortue vers le haut : la tortue doit ainsi être orthogonale à l'arête autour de laquelle elle tourne.

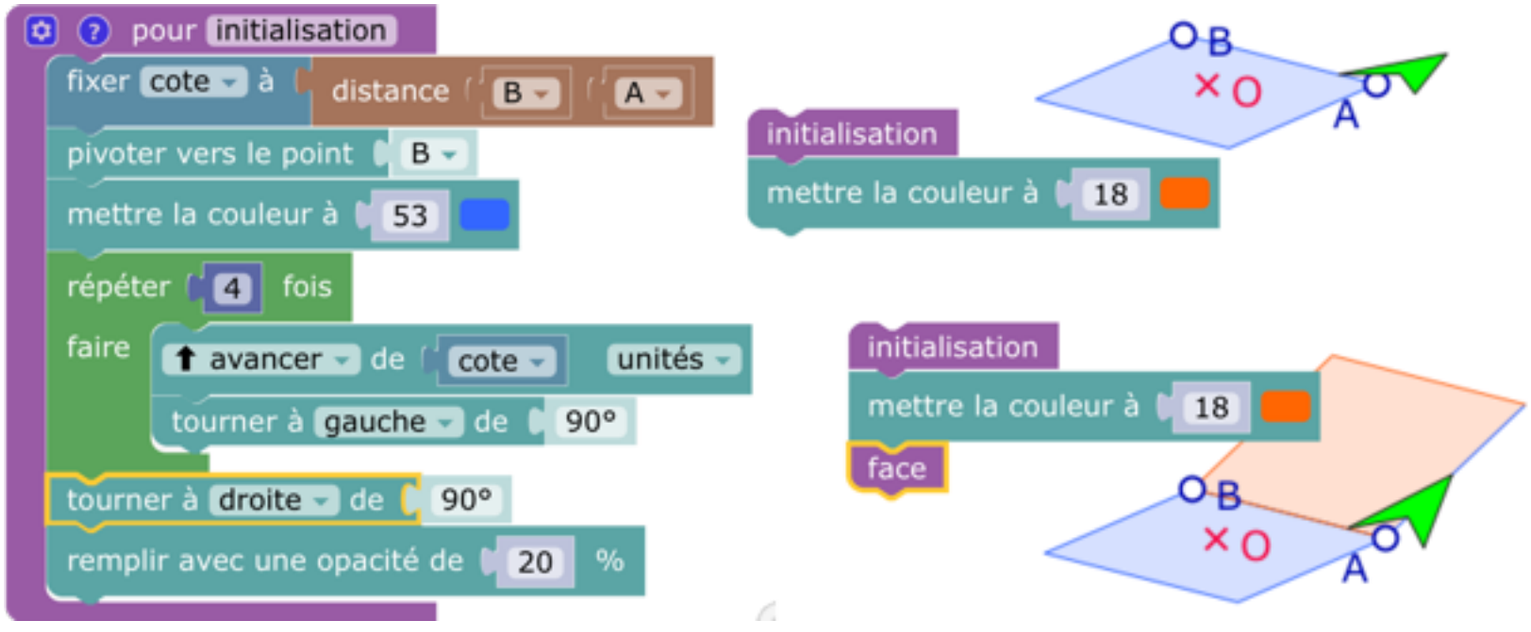


La procédure d'initialisation

Elle fixe la variable globale, trace le carré du sol, et surtout place la tortue perpendiculairement à l'arête [AB] autour de laquelle va tourner la face suivante.

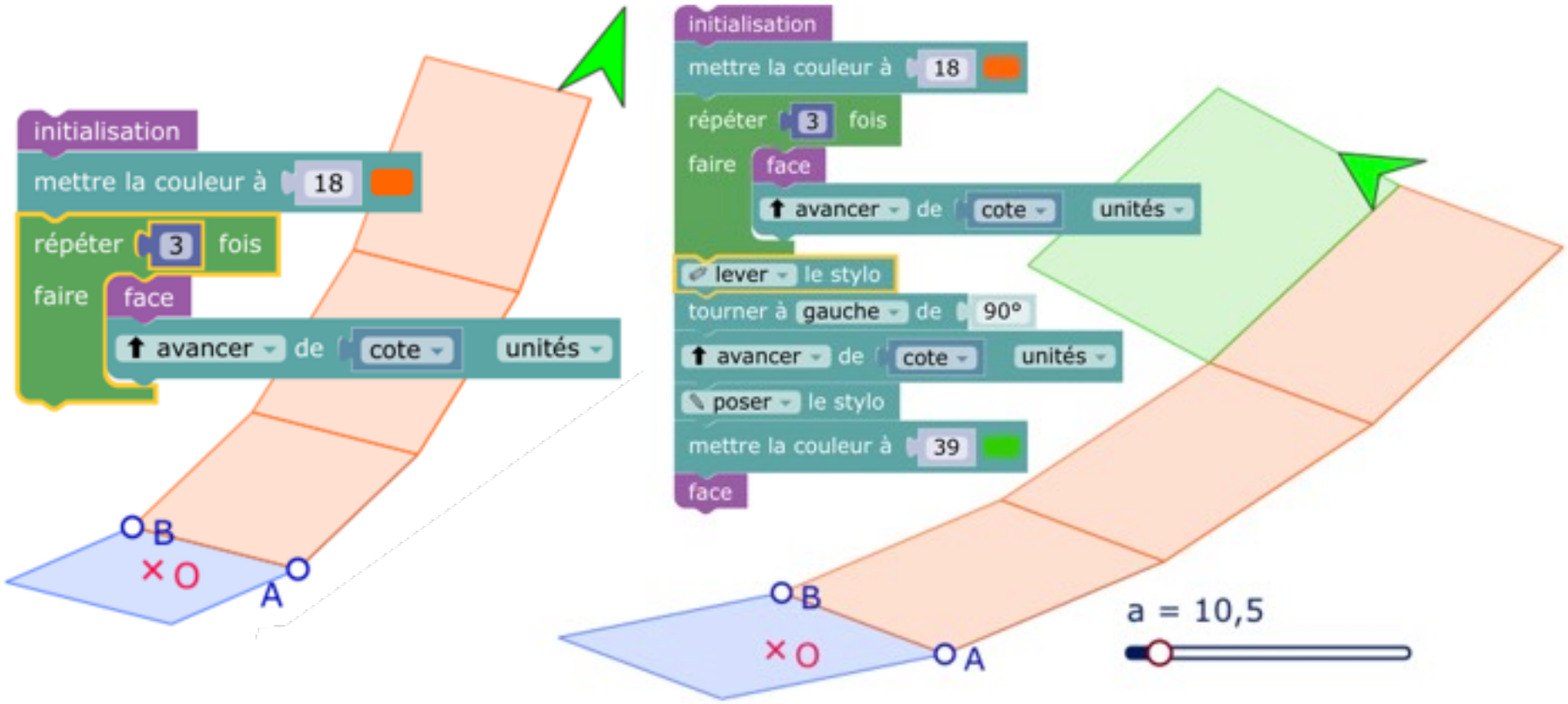
Exemple de la première face

Comme on le voit sur les différentes illustrations, la procédure **face** laisse la tortue à la sortie au même endroit qu'au début mais dans le plan de la nouvelle face. Il faudra donc juste avancer d'une arête pour placer la face suivante. C'est le choix retenu ici parmi d'autres options possibles.



Les faces latérales

Comme mentionné à la page précédente, il suffit d'itérer la nécessité d'avancer d'une arête après chaque face.
La tortue est alors dans le plan de la dernière face comme ci-contre.



La face suivante

Il suffit de placer la tortue au bon endroit et de tracer à nouveau une face (illustration en haut à gauche).

La dernière face

Ci-contre, on a repris le code de la face 5 et positionné la tortue au bon endroit pour appliquer une dernière fois **face**. Il faut revenir dans le plan de la dernière face latérale orange et placer la tortue dans la position que l'on voit ci-contre, puis appliquer la procédure face.

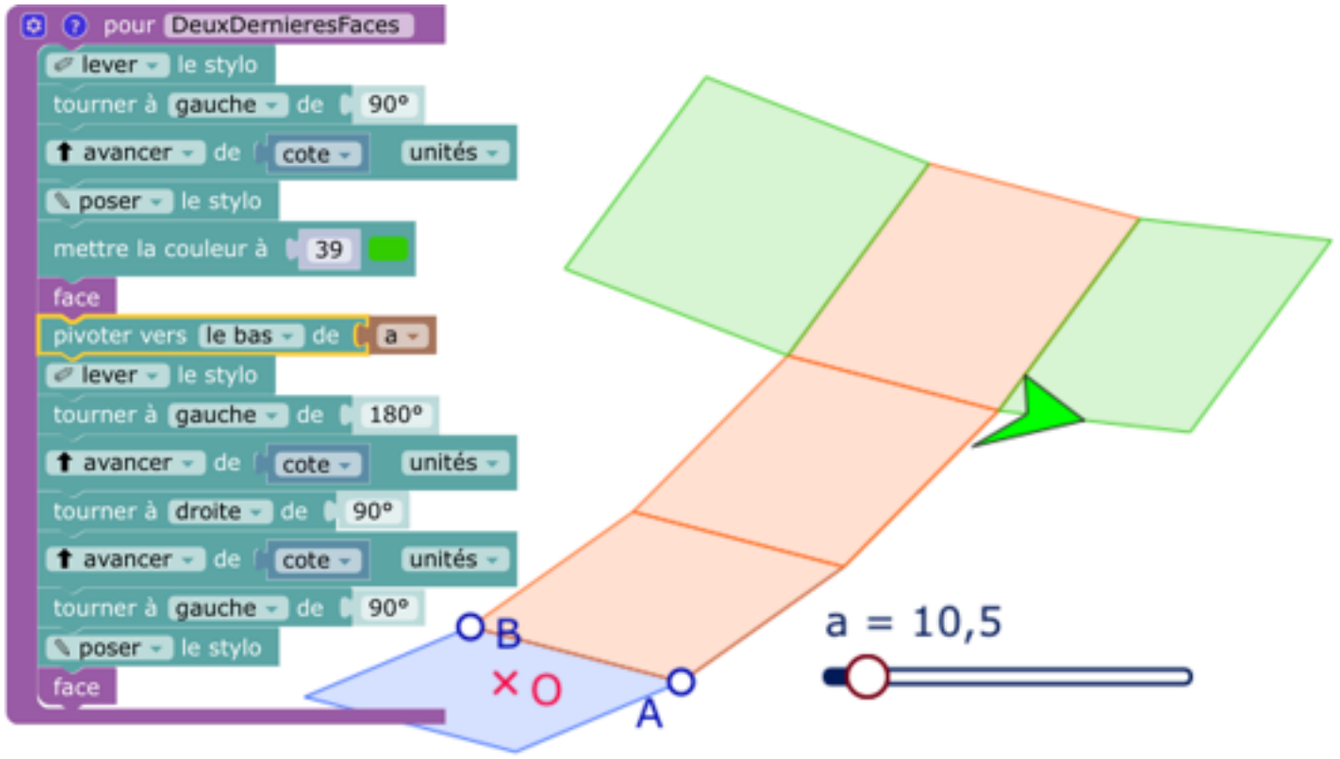
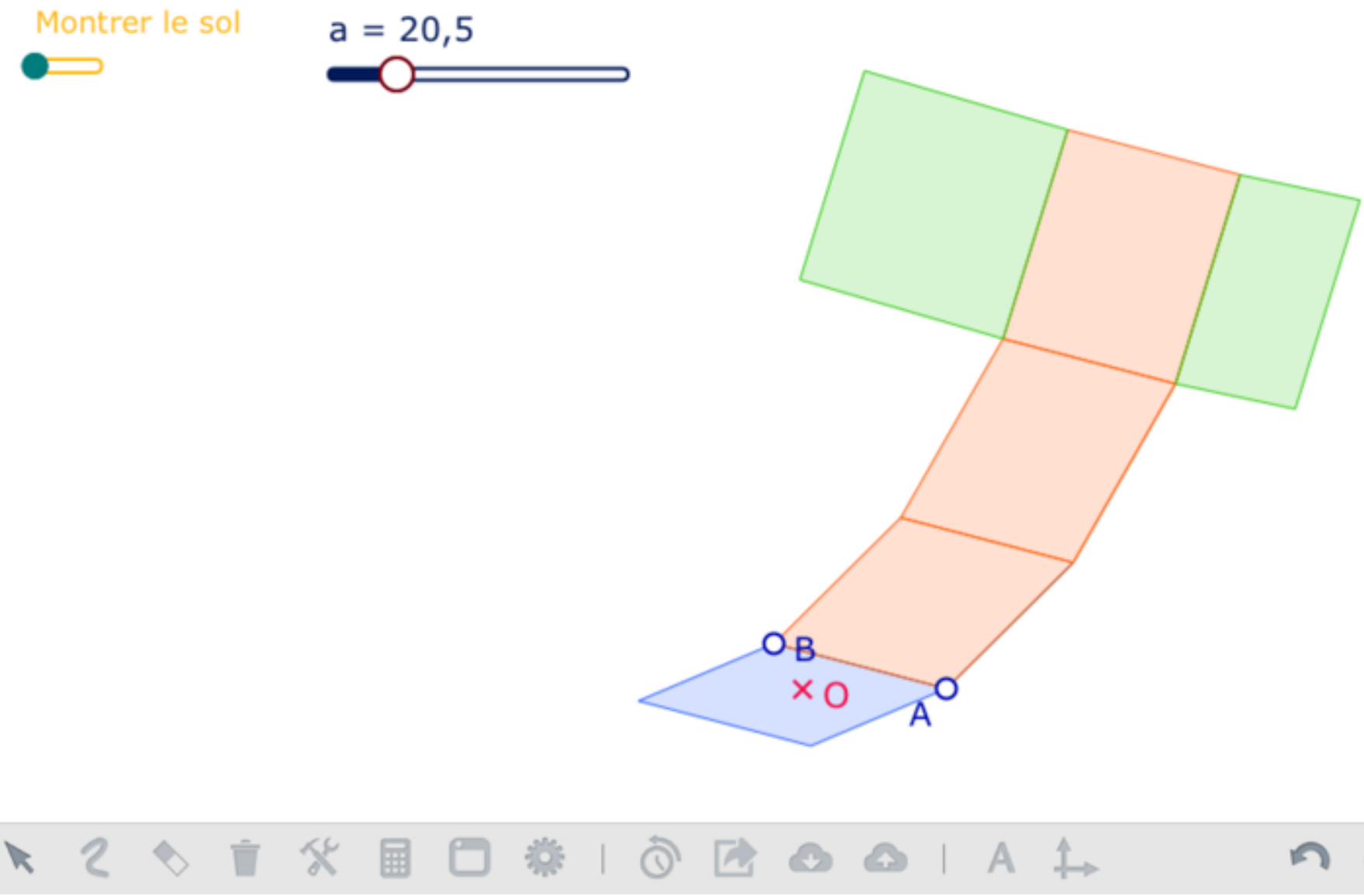


Figure dynamique 7.9 – Patron de cube standard réalisé à la tortue

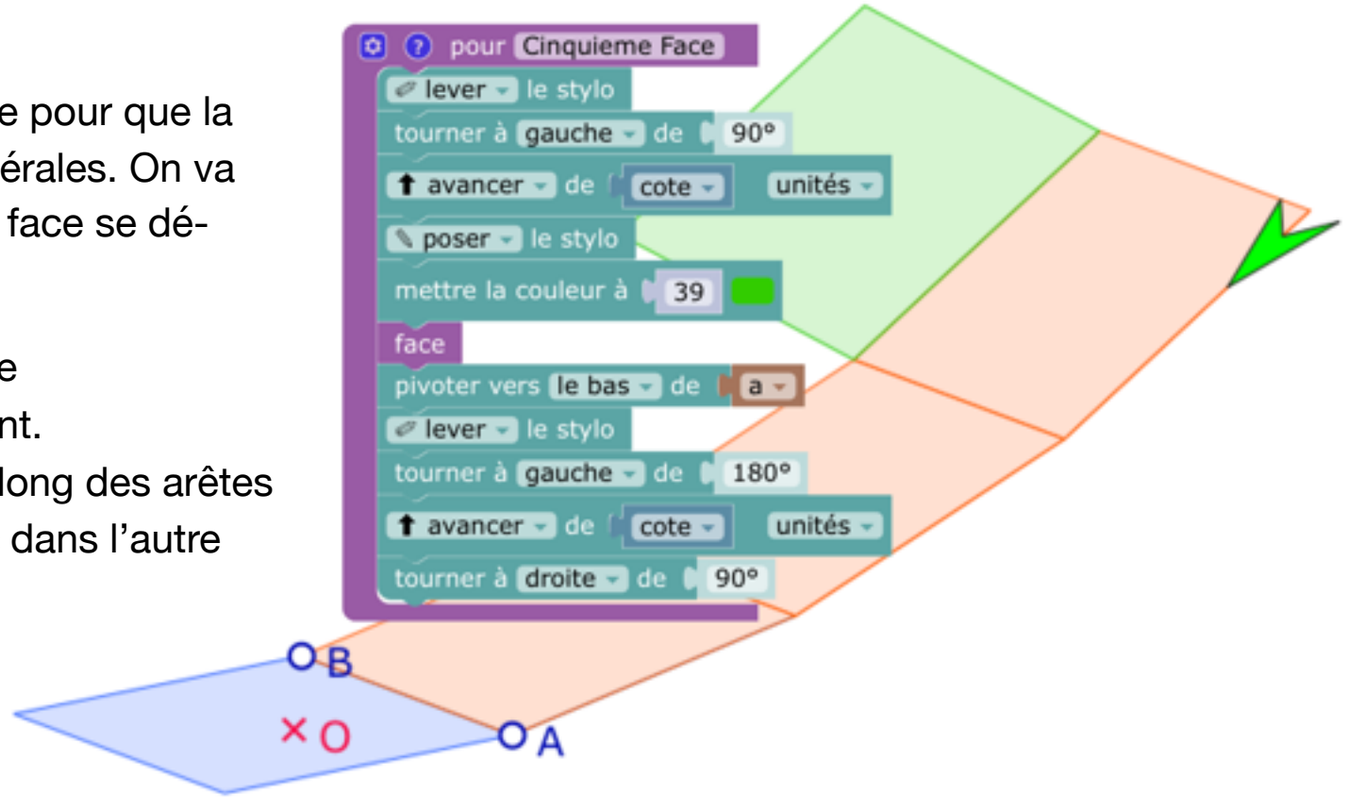


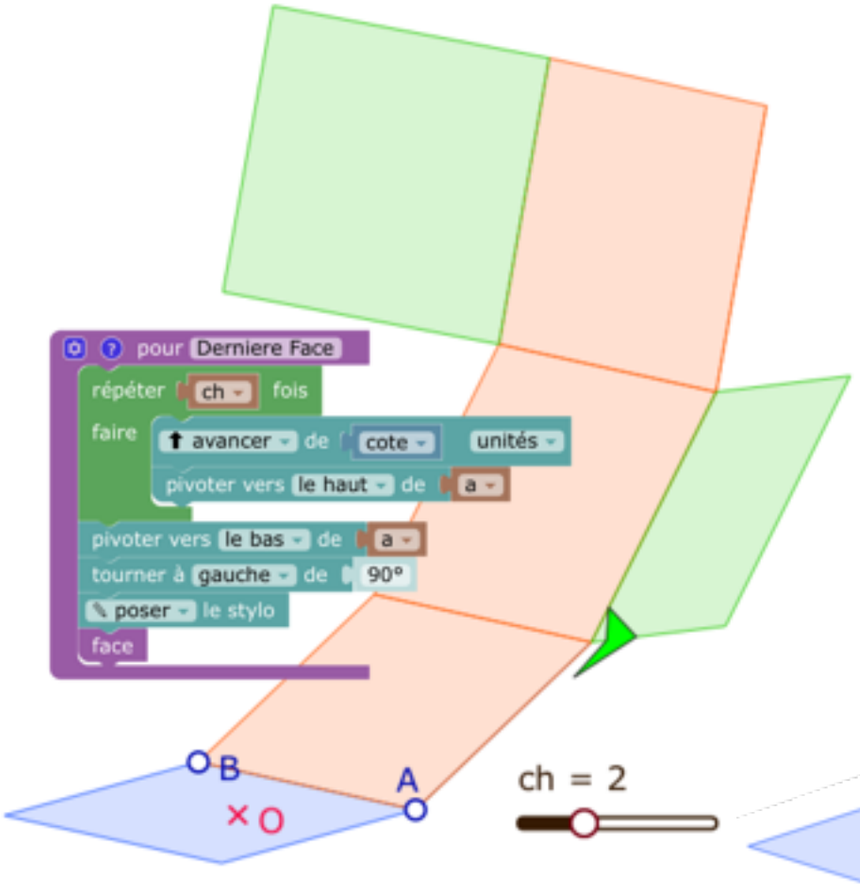
*La figure est en mode **consultation** à l'ouverture (aucun outil sélectionné) pour manipuler le trièdre.*
*Activer le pointeur de gauche pour passer en mode **standard** pour tester des modifications du code (en A).*

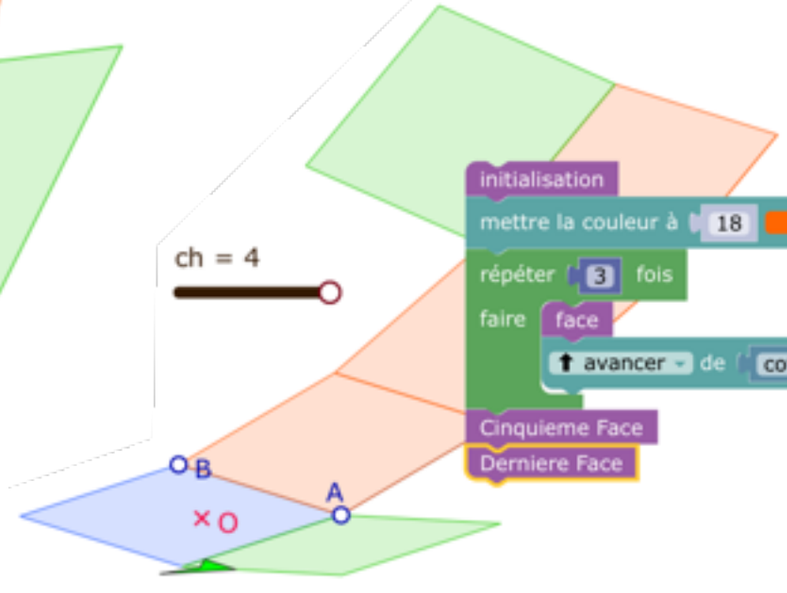
Ajout de trois autres patrons (hors contexte scolaire)

On commence par modifier le code de **la face 5** juste pour que la tortue soit le long des arêtes contigües des faces latérales. On va ainsi construire les patrons pour lesquels la dernière face se déplace sur ces arêtes.

On ajoute un curseur **ch** allant de 1 à 4 pour choisir le patron. La procédure **Dernière Face** s'écrit simplement.
On remarquera le **pivoter par le haut** pour circuler le long des arêtes et **vers le bas** pour le pliage car la tortue est orientée dans l'autre sens que dans la procédure **face**.







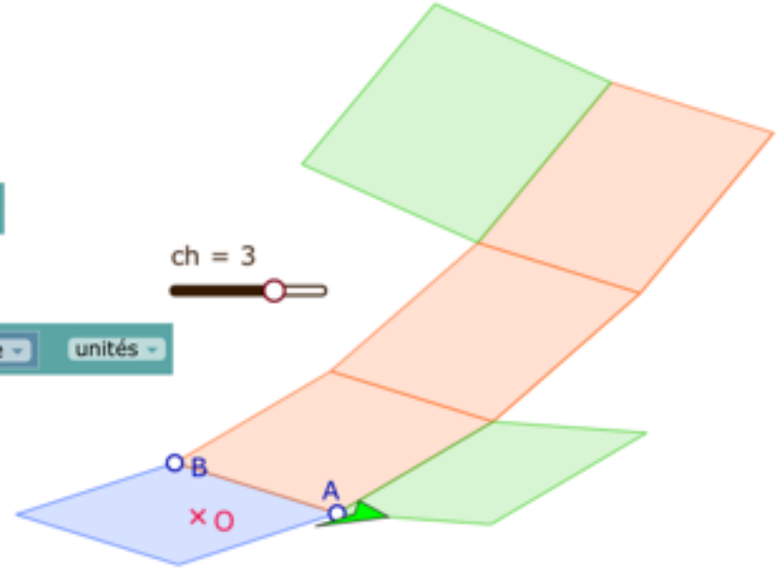
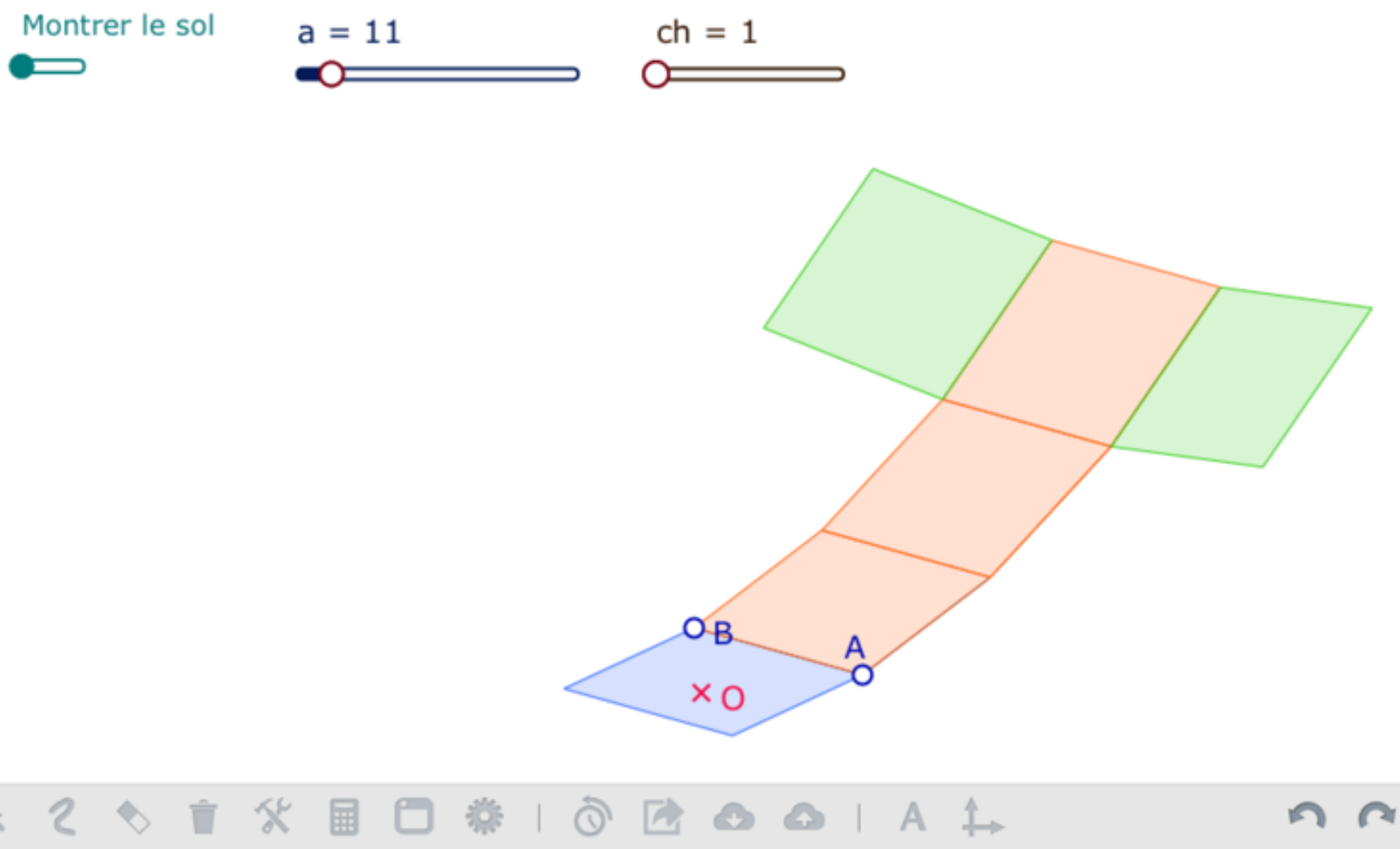


Figure dynamique 7.10 – Quatre patrons de cube comme trace dynamique de la tortue



*Le code est toujours dans A - Vérifier si vous êtes en mode **consultation** (à l'ouverture) ou **standard** (Blockly).*

Compléments - utilisation du « pivoter gauche »

Dans les activités précédentes, on a choisi de privilégier deux seules orientations pour pivoter : haut et bas. Cela a suffit car on travaillait essentiellement sur des faces carrées. Cela correspond aussi au mouvement naturel de la tête : si on veut regarder vers le haut, on pivote la tête vers le haut. L'axe symbolique (d'une arête de polyèdre) correspondant pourrait être celui des épaules. On ne tourne la tête pour regarder le ciel que quand on est allongé, un axe symbolique associé serait alors le corps tout entier.

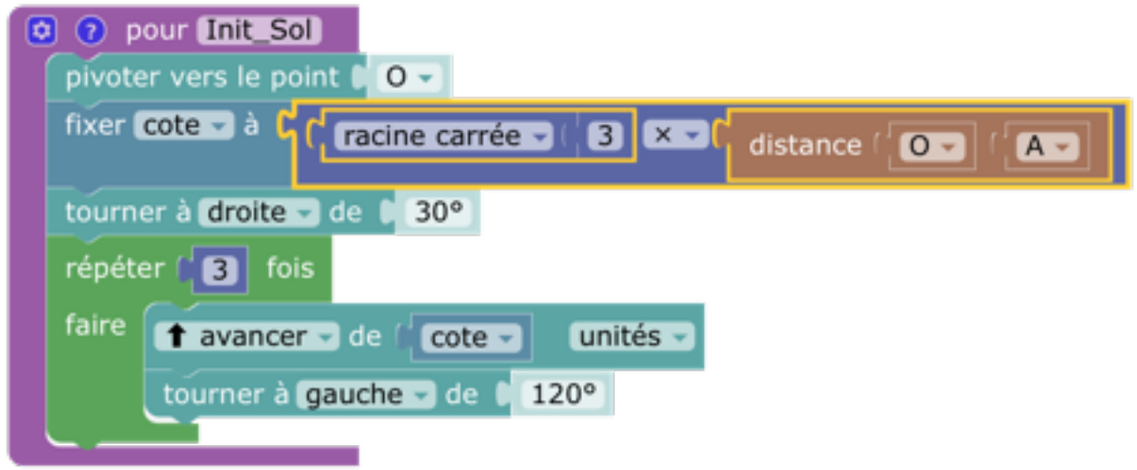
Mais en géométrie, dès que l'on ne travaille plus sur des faces à angle droit, le pivoter vers le haut n'est plus l'outil le plus naturel, il est bien plus simple de placer la tortue sur l'arête du pivot, la faire pivoter et ensuite construire les faces voulues dans le plan de la tortue ainsi défini. Nous allons voir cela sur quelques exemples, avec des faces triangulaires.

Par ailleurs, comme les faces vont être triangulaires, on se limitera à des tracés filaires : une arête étant préexistante, dessiner un triangle reviendra à dessiner deux arêtes. L'avantage est que l'on est d'avance sur le sommet de la face suivante. L'inconvénient est que les triangles n'étant pas fermés, on ne peut pas les colorier.

Nous commencerons par deux exemples détaillés de patrons du tétraèdre régulier, puis nous verrons une application plus avancée avec 6 patrons de l'octaèdre.

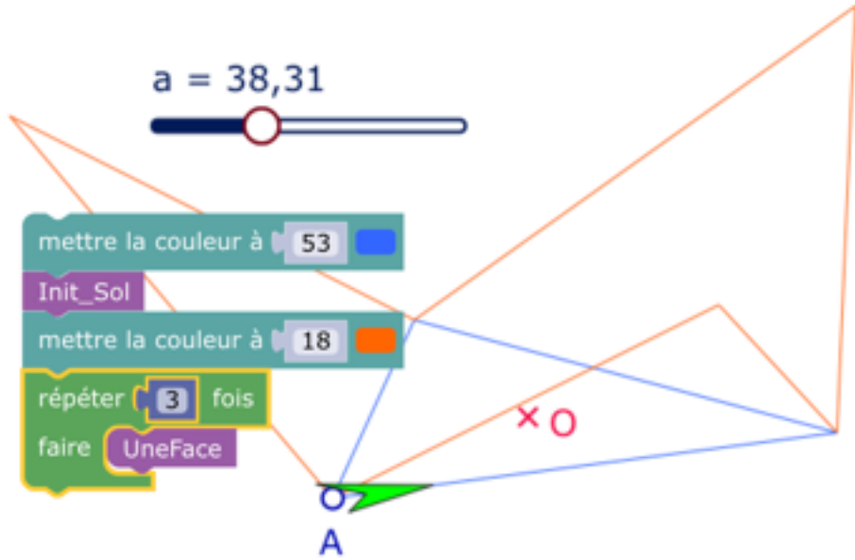
Tétraèdre régulier - Patron 1

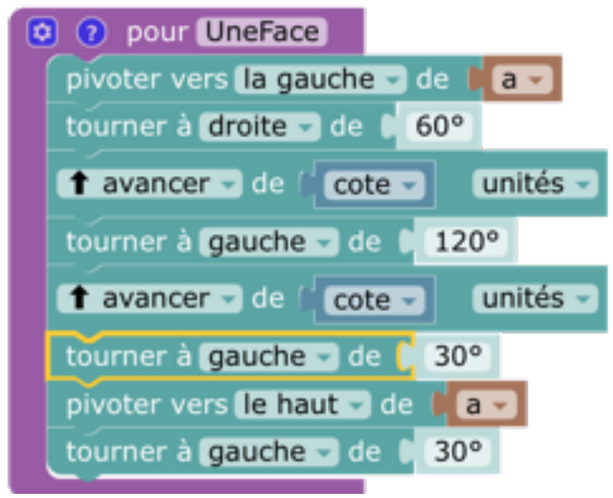
Il s'agit du patron standard de 3 triangles autour du triangle «du sol».
L'avantage immédiat est que les triangles ont tous **la même orientation,** d'où la factorisation élémentaire ci-contre.

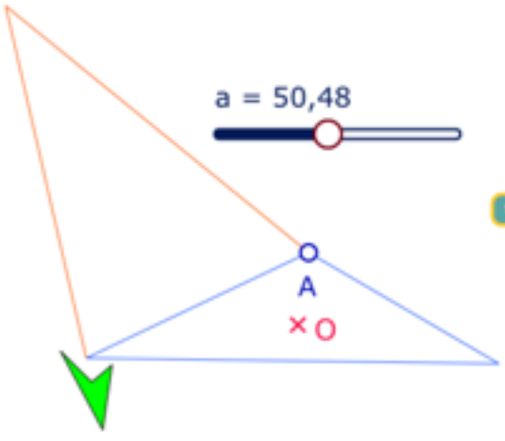


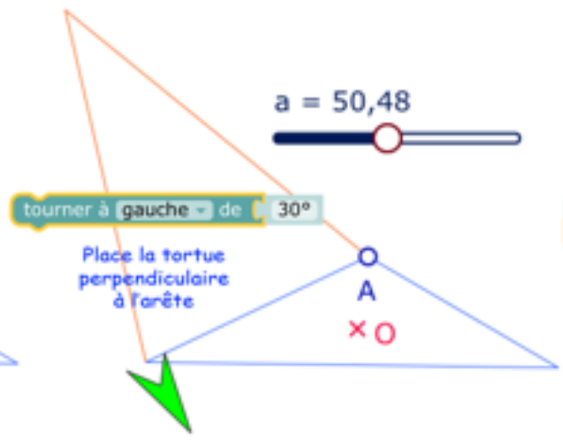
Une face par « pivoter vers la gauche »

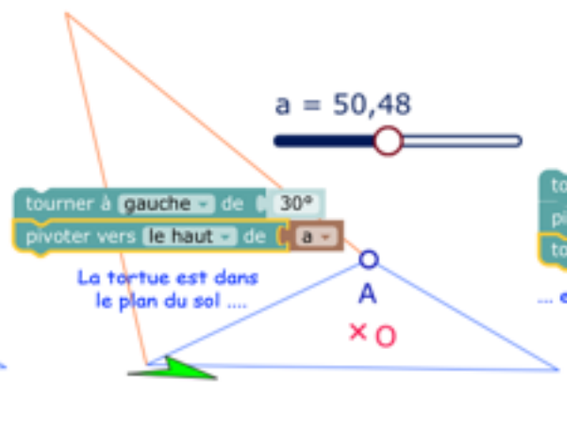
On voit qu'à la sortie de la procédure d'initialisation, la tortue est sur une arête.
On pivote donc vers la gauche pour être dans le plan de pliage.
Dans ce plan on trace la face (4 lignes, dont 2 **avancer**). L'important ici est dans les trois dernières lignes, pour revenir dans le plan du sol. Voici le détail :











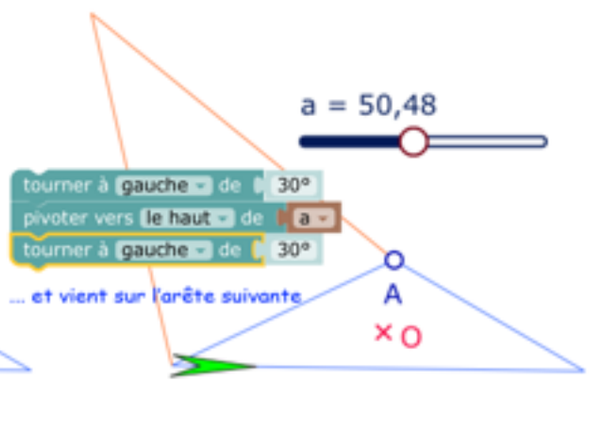
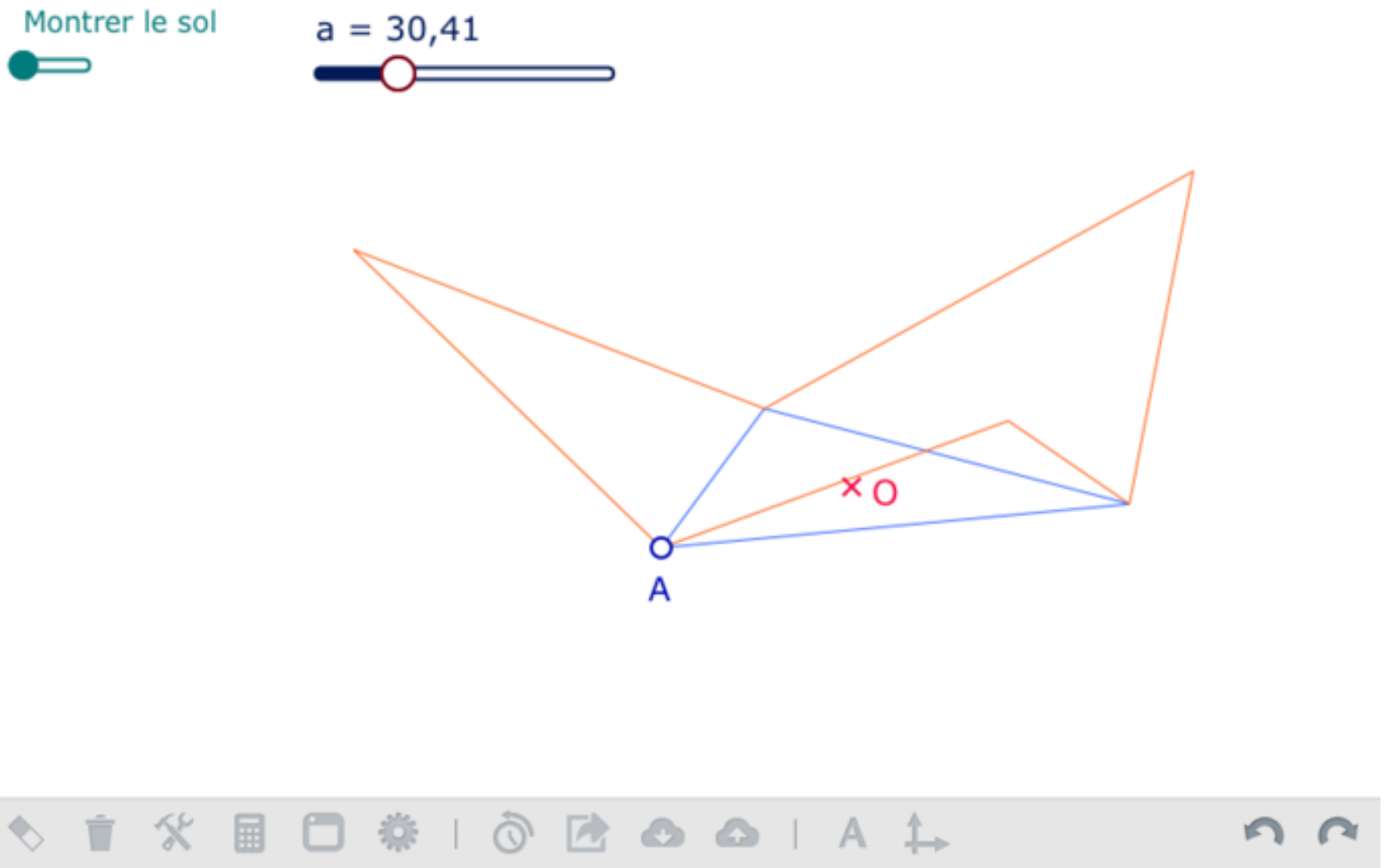


Figure dynamique 7.11 – Tétraèdre régulier – Patron filaire 1 (les faces ont toutes même orientation)



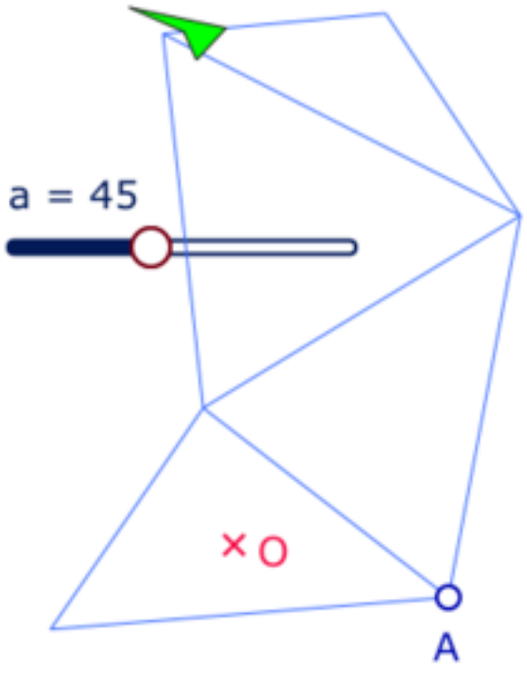
*Le code est en A - Au départ la figure est en mode **consultation**.*

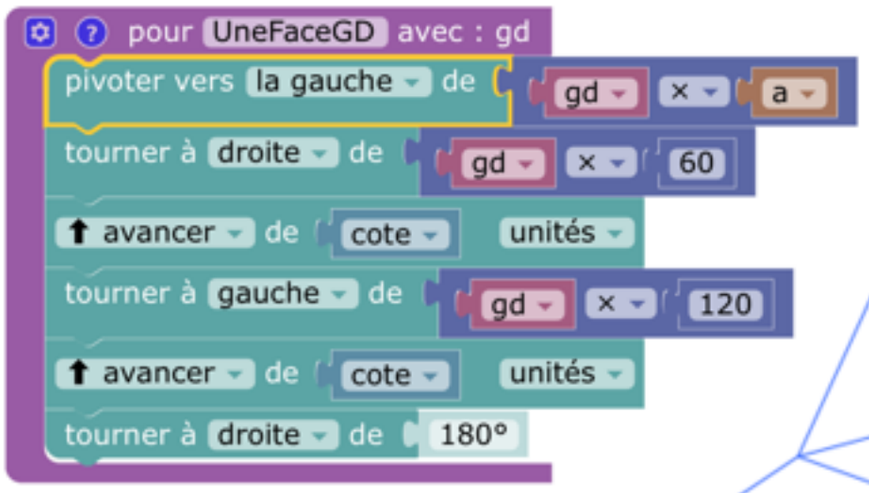
Tétraèdre régulier - Patron de « peau d'orange »

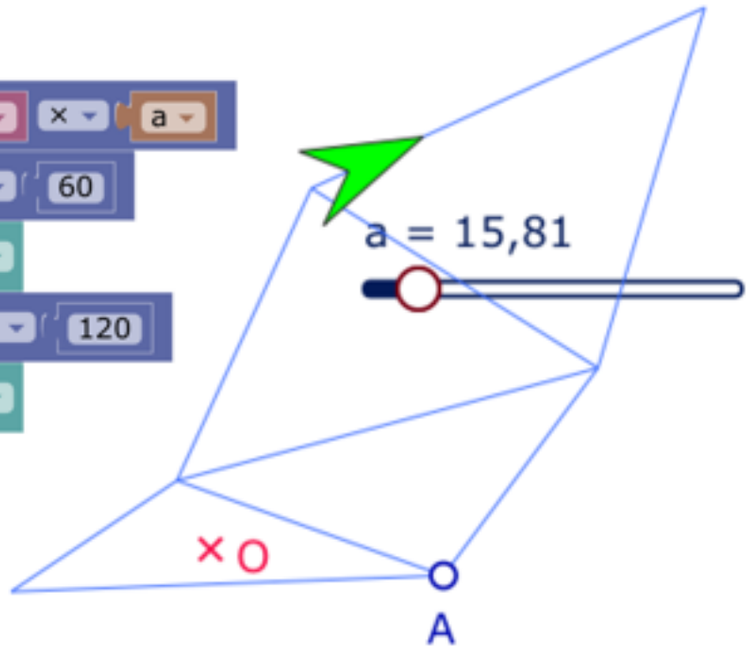
L'avantage de ce patron, pour tous les polyèdres pour lesquels on peut le faire, est que l'on **« continue de monter »**, c'est-à-dire que l'on change de plan à chaque face, sans avoir à revenir à une face précédente. Cela fait un patron très simple à écrire.

Mais – en général – dans ce cas, il y a des faces « gauches » et des faces « droites », deux orientations des faces, en fonction de la position de la tortue. On peut faire bien entendu une procédure **FaceGauche** et une **FaceDroite**, mais on peut aussi **paramétrer la procédure**. Même si cela la rend un peu moins lisible au départ, c'est un procédé très standard.

L'intérêt est donc qu'on n'utilise qu'**une seule fois** – par appel – **le changement de plan**, ici un pivoter gauche.







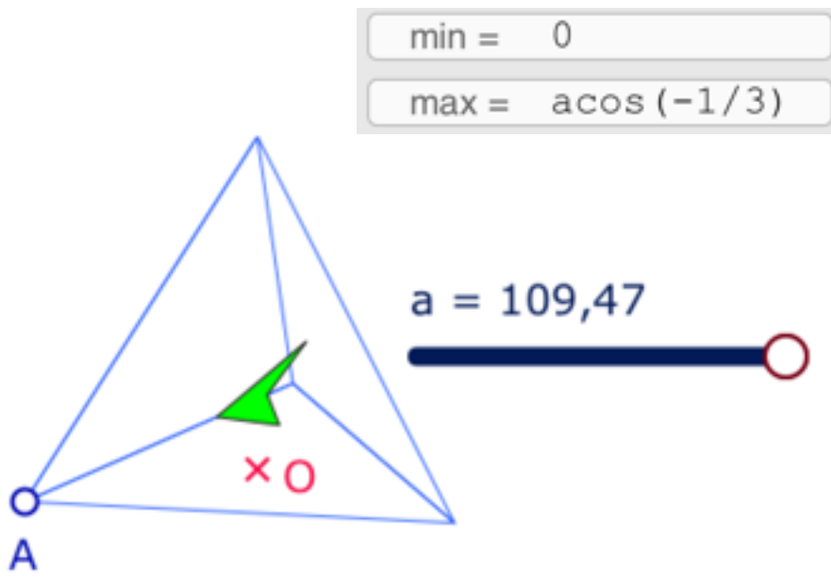
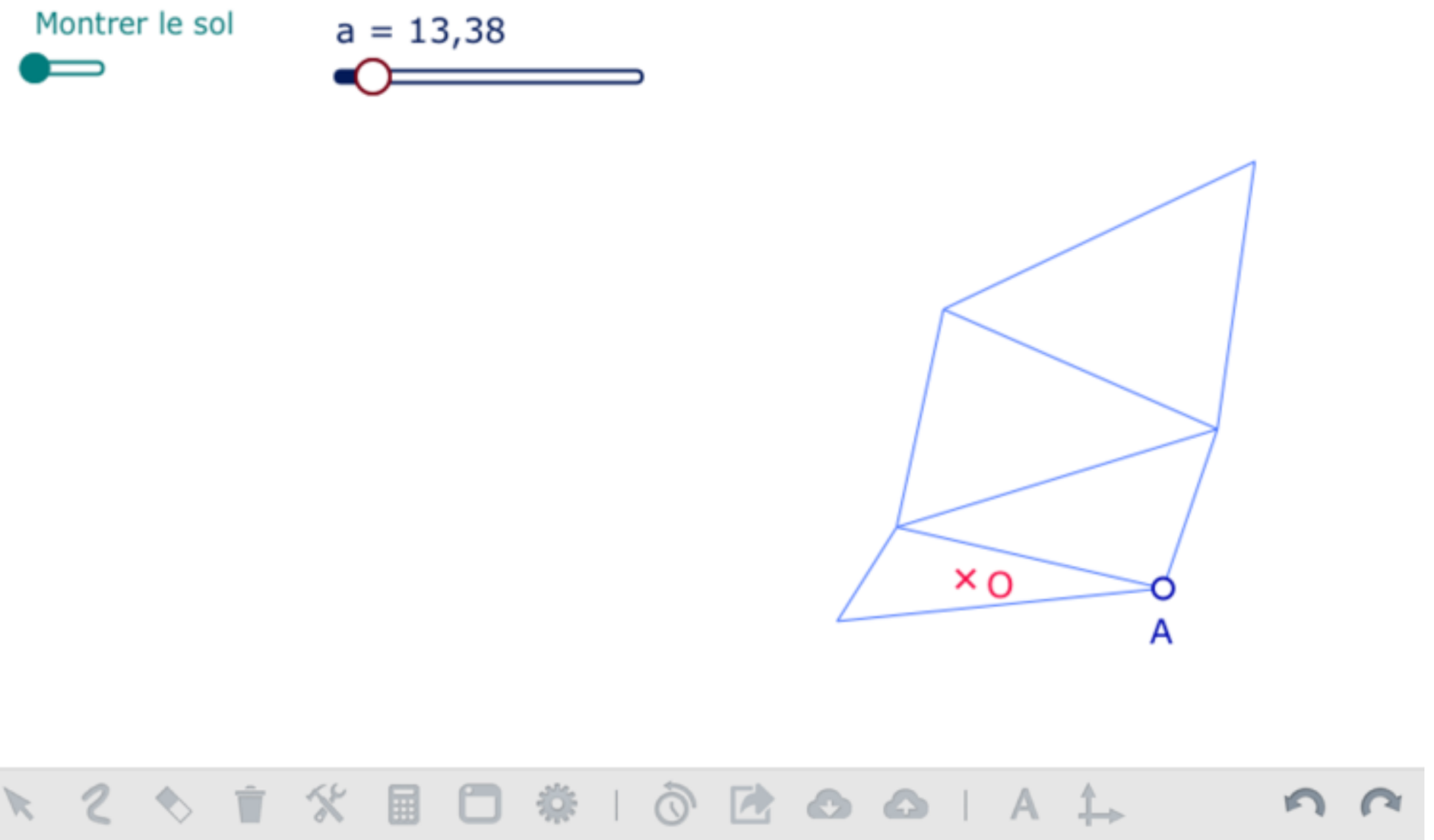
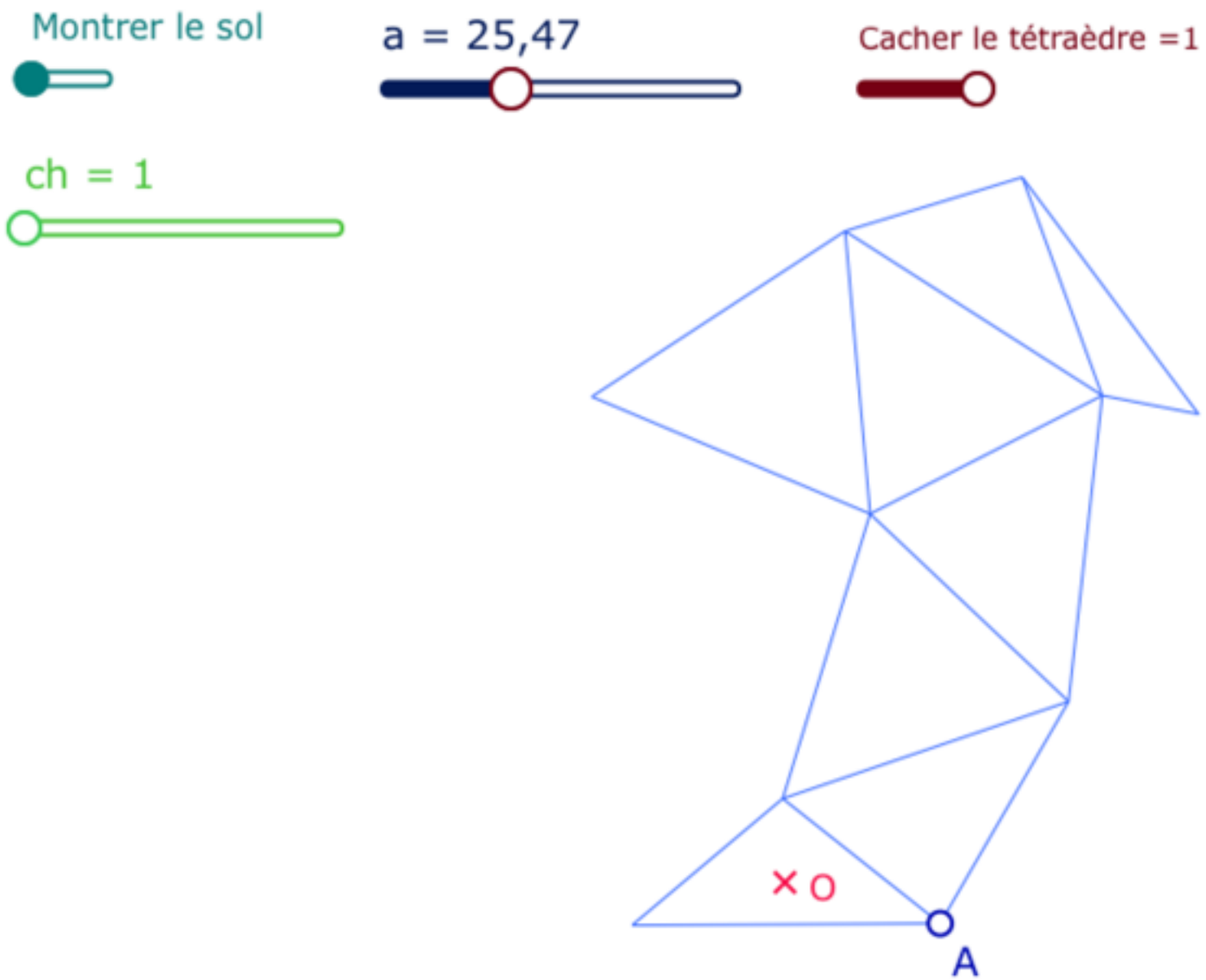


Figure dynamique 7.12 – Tétraèdre – Patron filaire en peau d'orange



*Toujours le code en A. La figure est en mode **consultation** à l'ouverture*

Figure dynamique 7.13 – Six patrons filaires de l'octaèdre régulier



*La figure est seulement en **consultation** - Le code est détaillé dans l'article de MathémaTICE sur la tortue de DGPad.*

Galerie 3D

Dans ce chapitre nous ne détaillerons pas le code de chaque figure, on trouvera plus de détail dans l'article de **MathémaTICE** consacré à la tortue : http://revue.sesamath.net/spip.php?article875

La galerie commence par quelques patrons de polyèdres semi-réguliers et se termine par quelques parcours hamiltoniens ou eulériens sur quelques polyèdres :

1. Trois patrons du dodécaèdre

2. Cinq patron du ballon de football

3. Un patron de l'icosidodécaèdre (car eulérien)

4. Un patron du rhombicosidodécaèdre (car eulérien)

5. Un «bi-patron» du snob-cube et de son chiral

6. Un patron du snob-dodécaèdre (juste parce que 92 faces)

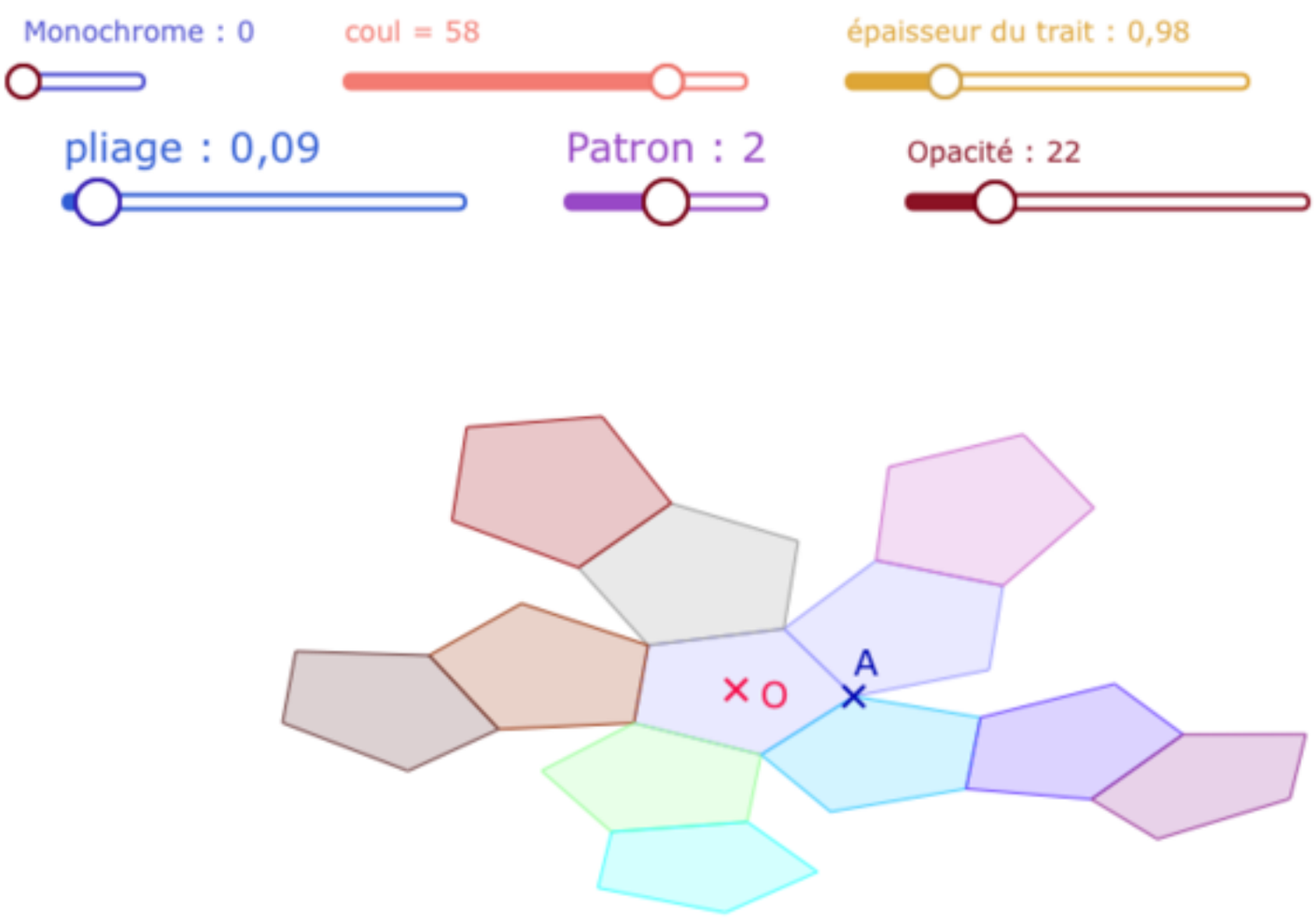
7. Des chemins hamiltoniens sur le rhombidodécaèdre

8. Chemins hamiltoniens et cycles eulériens sur l'icosidodécaèdre

9. 357 cycles hamiltoniens sur le rhombicosidodécaèdre

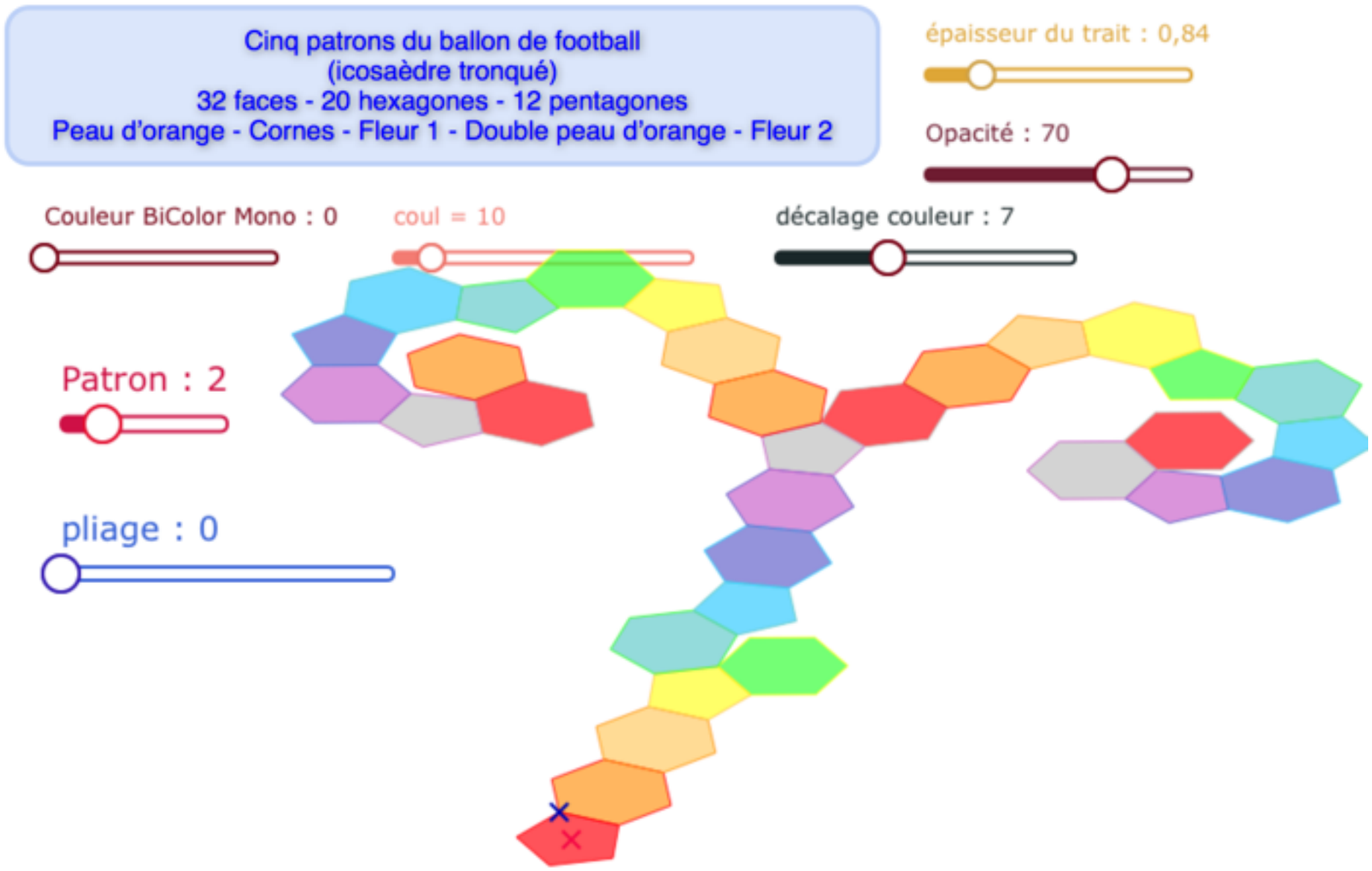
Dodécaèdre régulier

Figure dynamique 8.1 – Trois patrons du dodécaèdre



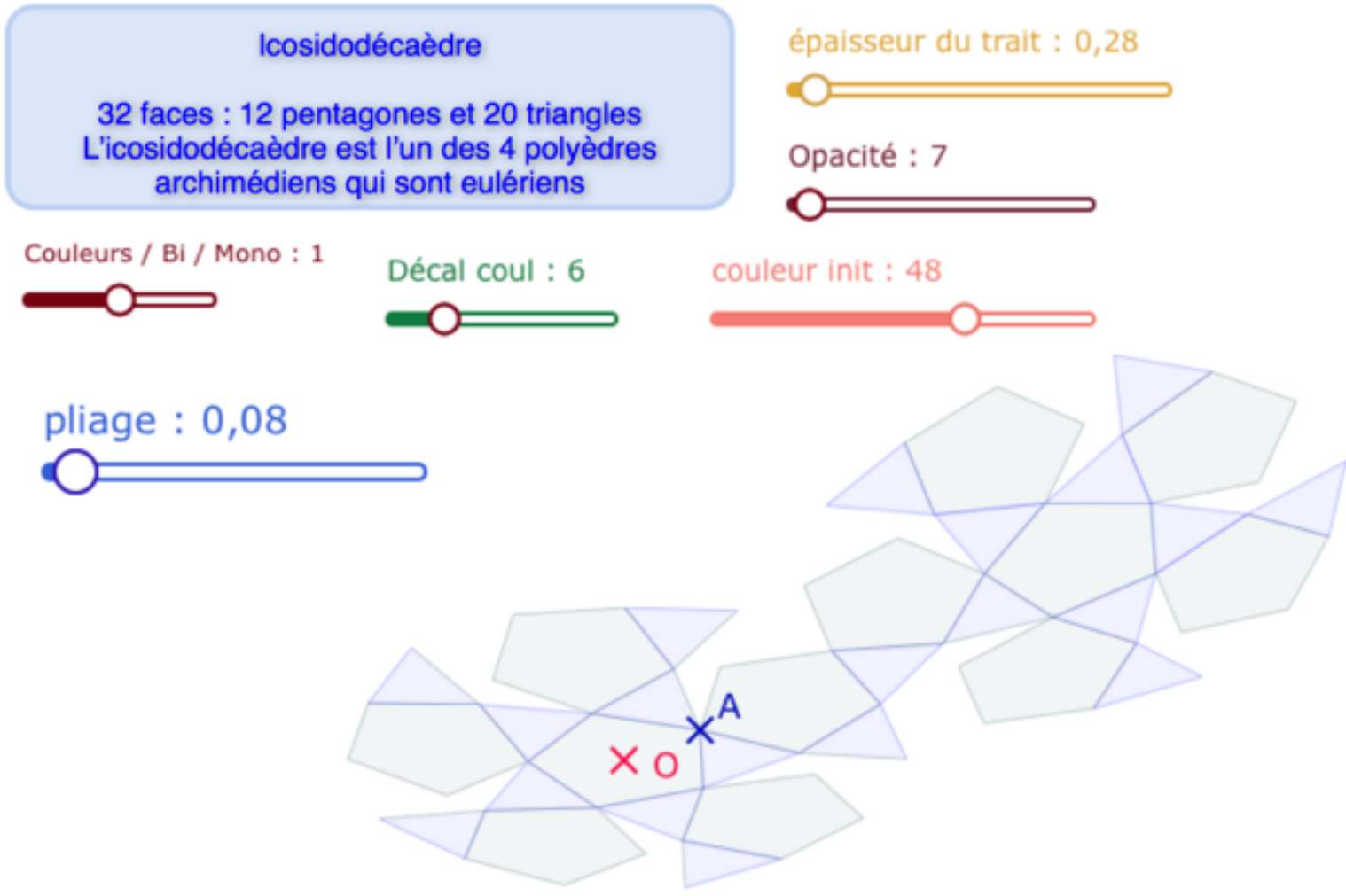
Icosaèdre tronqué - ballon de football

Figure dynamique 8.2 – Cinq patrons du ballon de football



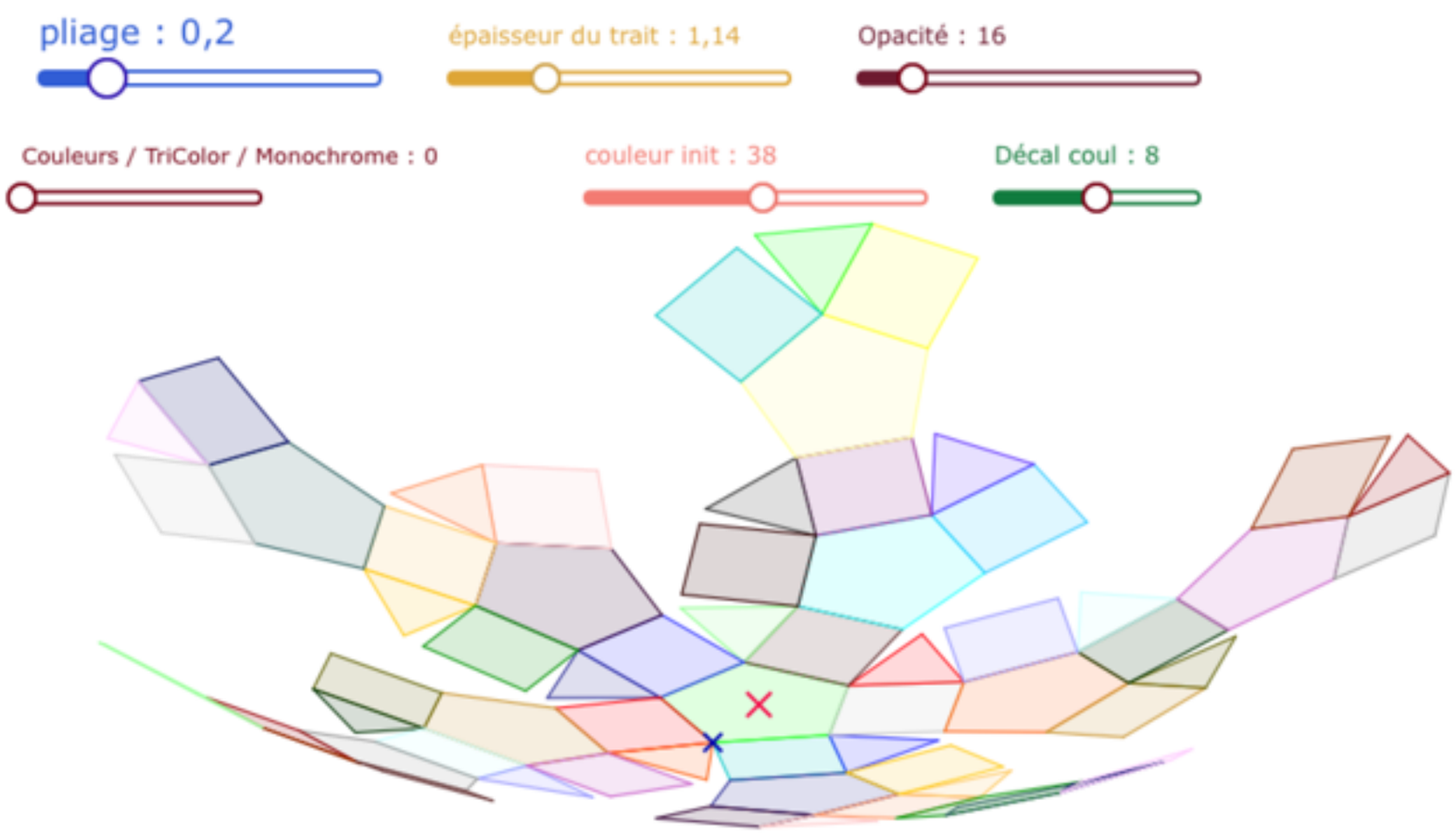
Icosidodécaèdre régulier

Figure dynamique 8.3 – Un patron de l'icosidodécaèdre



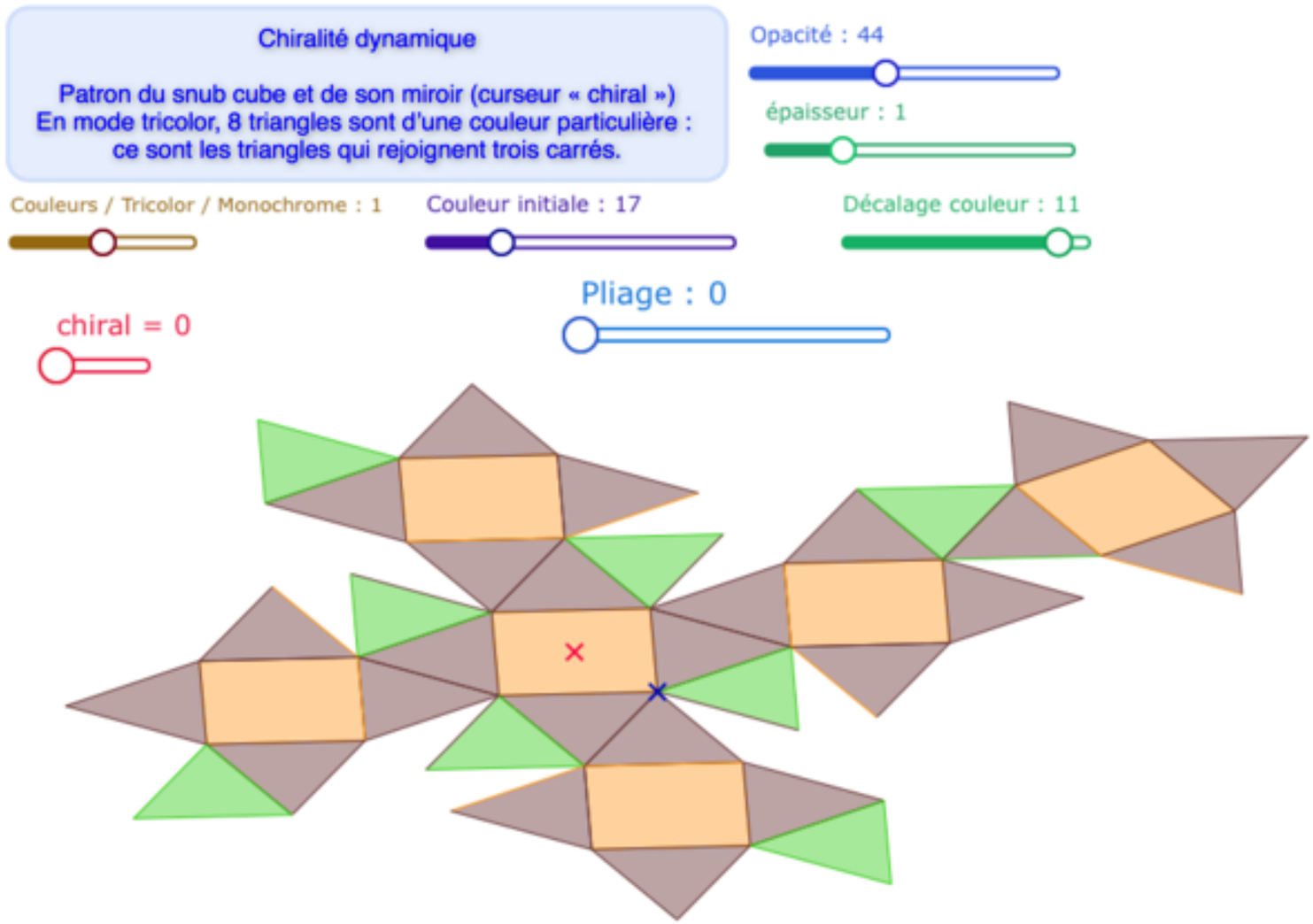
Rhombicosidodécaèdre

Figure dynamique 8.4 – Polyèdre semi-régulier eulérien (voir plus loin) - 62 faces (5x12+2)



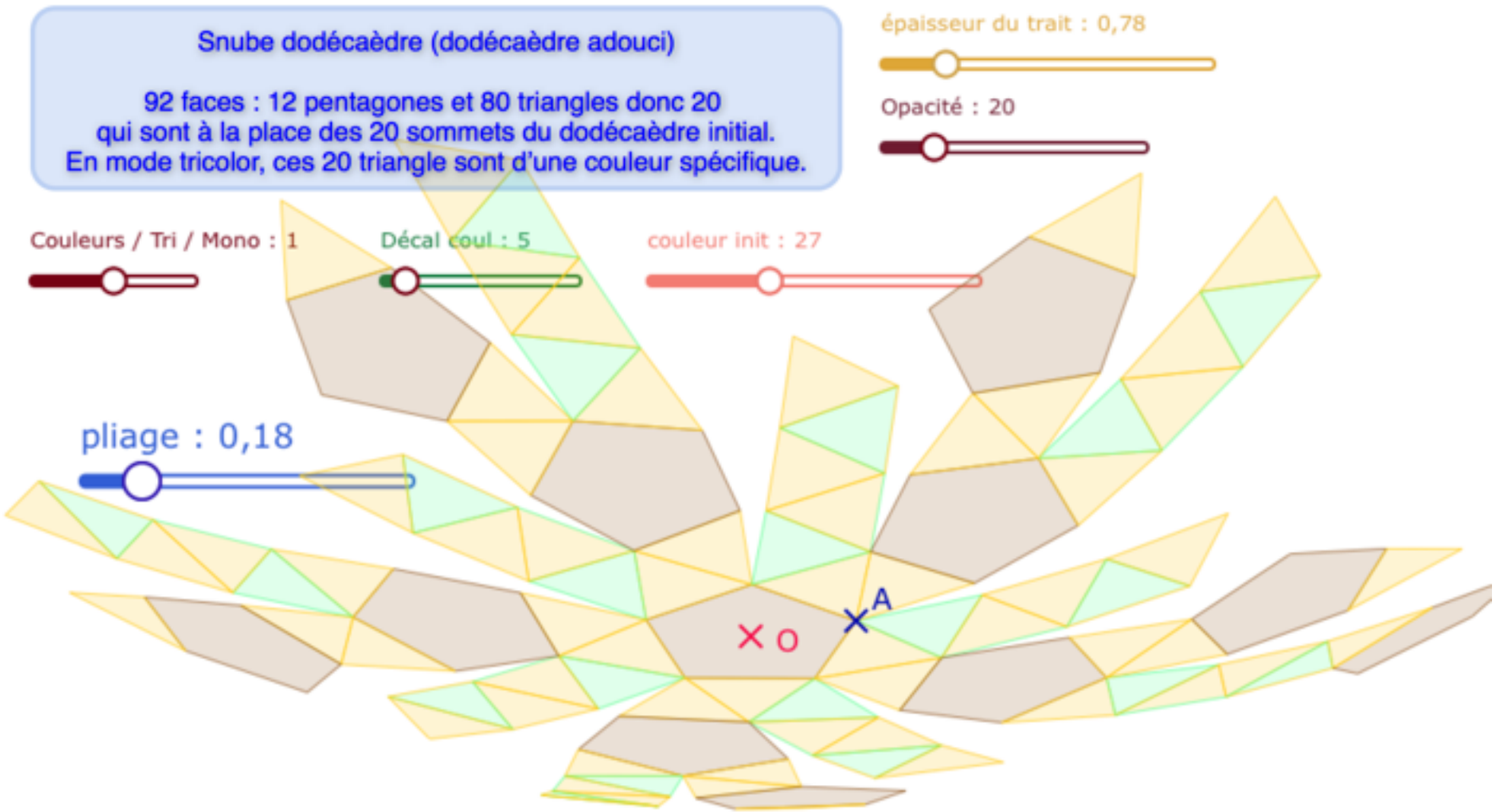
Snub cube et sa version chirale

Figure dynamique 8.5 – Chiralité dynamique sur le snub cube



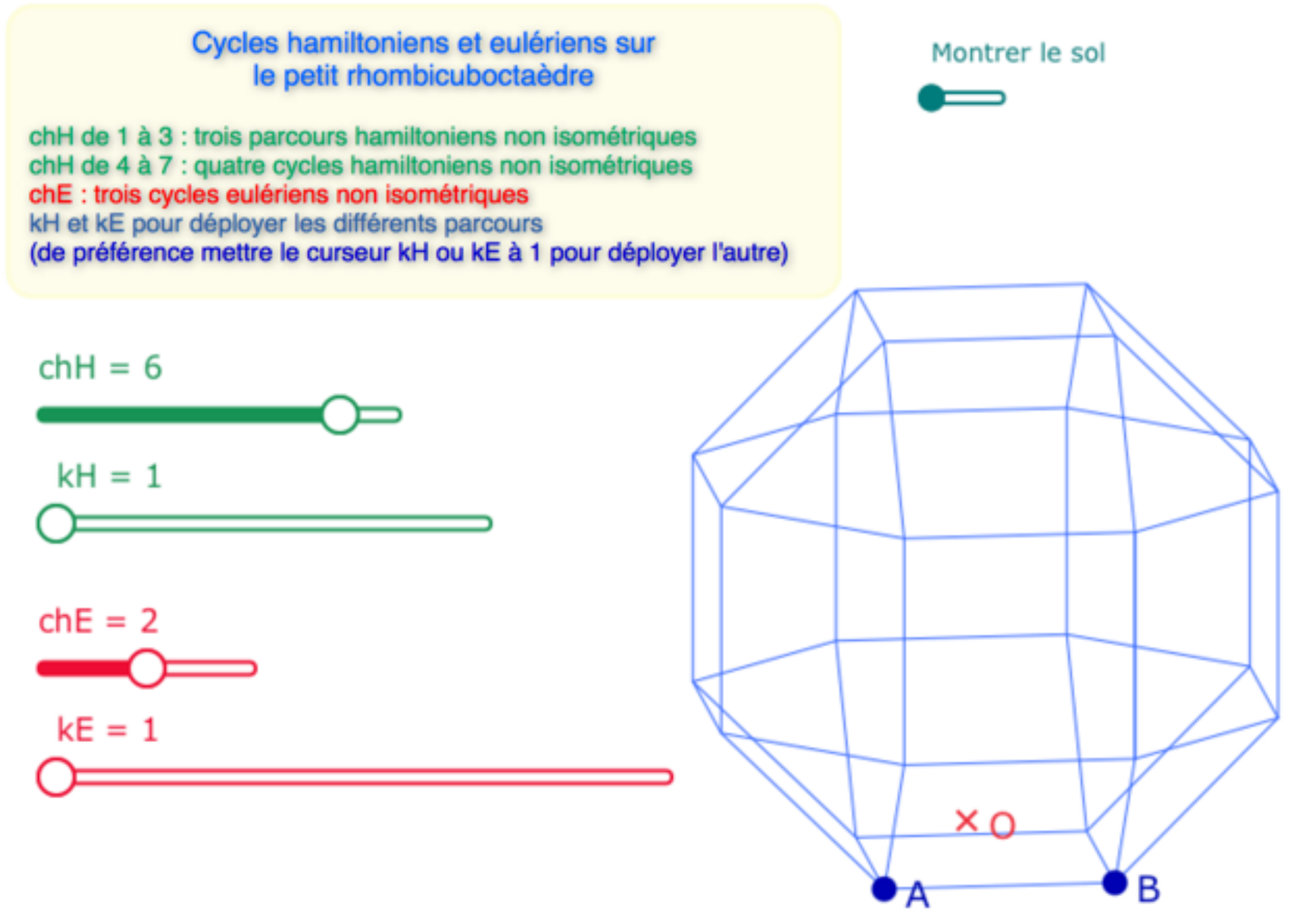
Snob-dodécaèdre

Figure dynamique 8.6 – Patron du snub dodécaèdre (92 = 5x18 + 2)



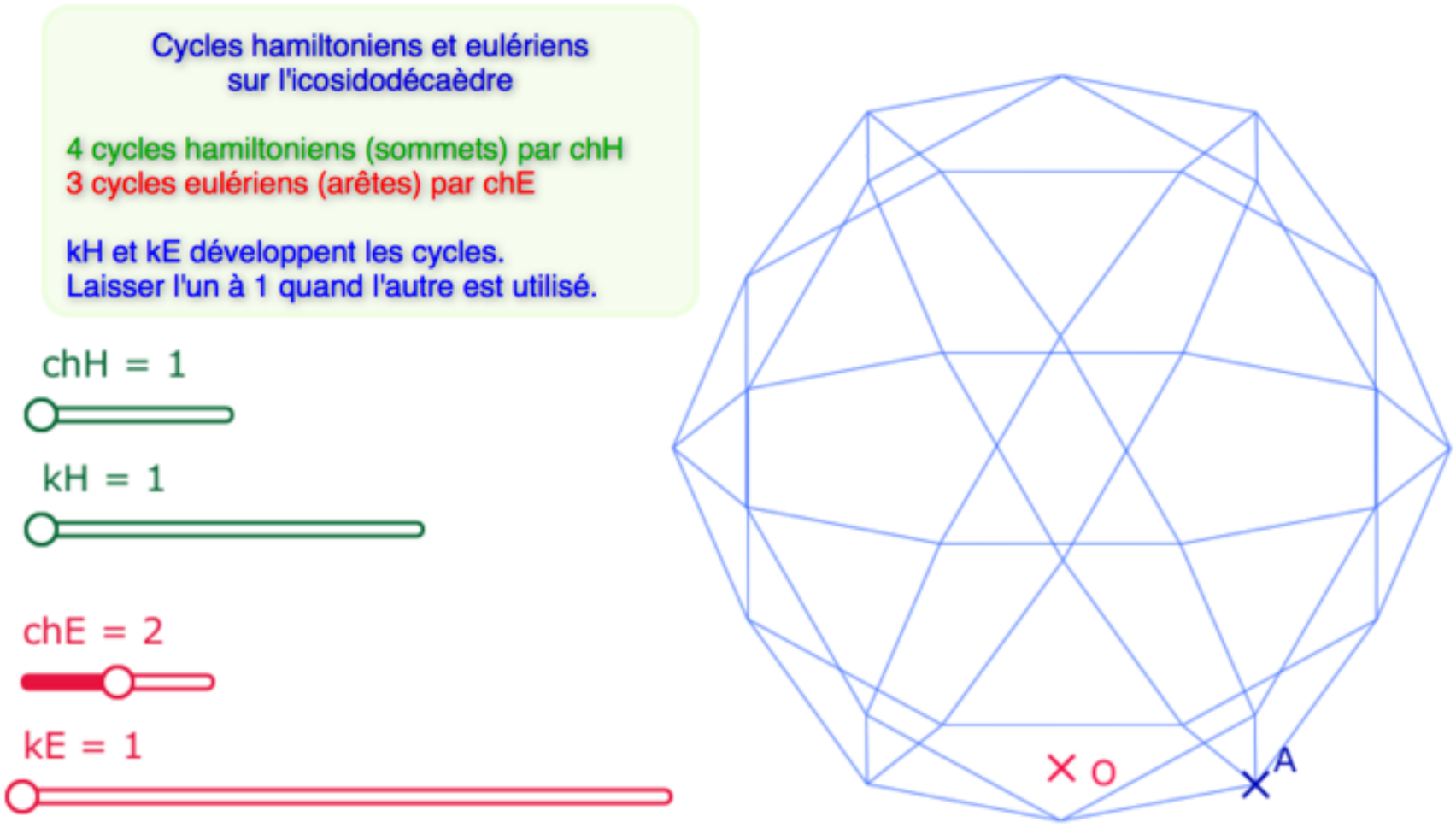
Chemins et cycles sur le rhombicuboctaèdre

Figure dynamique 8.7 – Chemins et cycles sur polyèdres - hamiltoniens pour les sommets, eulériens pour les arêtes



L’icosidodécaèdre est un polyèdre eulérien

Figure dynamique 8.8 – Chemins et cycles sur polyèdres - autre exemple



357 cycles hamiltoniens sur le rhombicosidodécaèdre

La figures suivante propose **357 cycles hamiltoniens** sur le rhombicosidodécaèdre.
Son organisation est la suivante :

Un premier cycle hamiltonien est créé (**chDeb=1**et **chFin=1**) avec les 30 premiers sommets plus ou moins sur un seul hémisphère et les 30 autres sur l'autre.

Sur cette base, deux partitions sont créées : celles des 30 premiers sommets utilisés, et celle des 30 derniers (31 avec le retour en A - croix bleue).

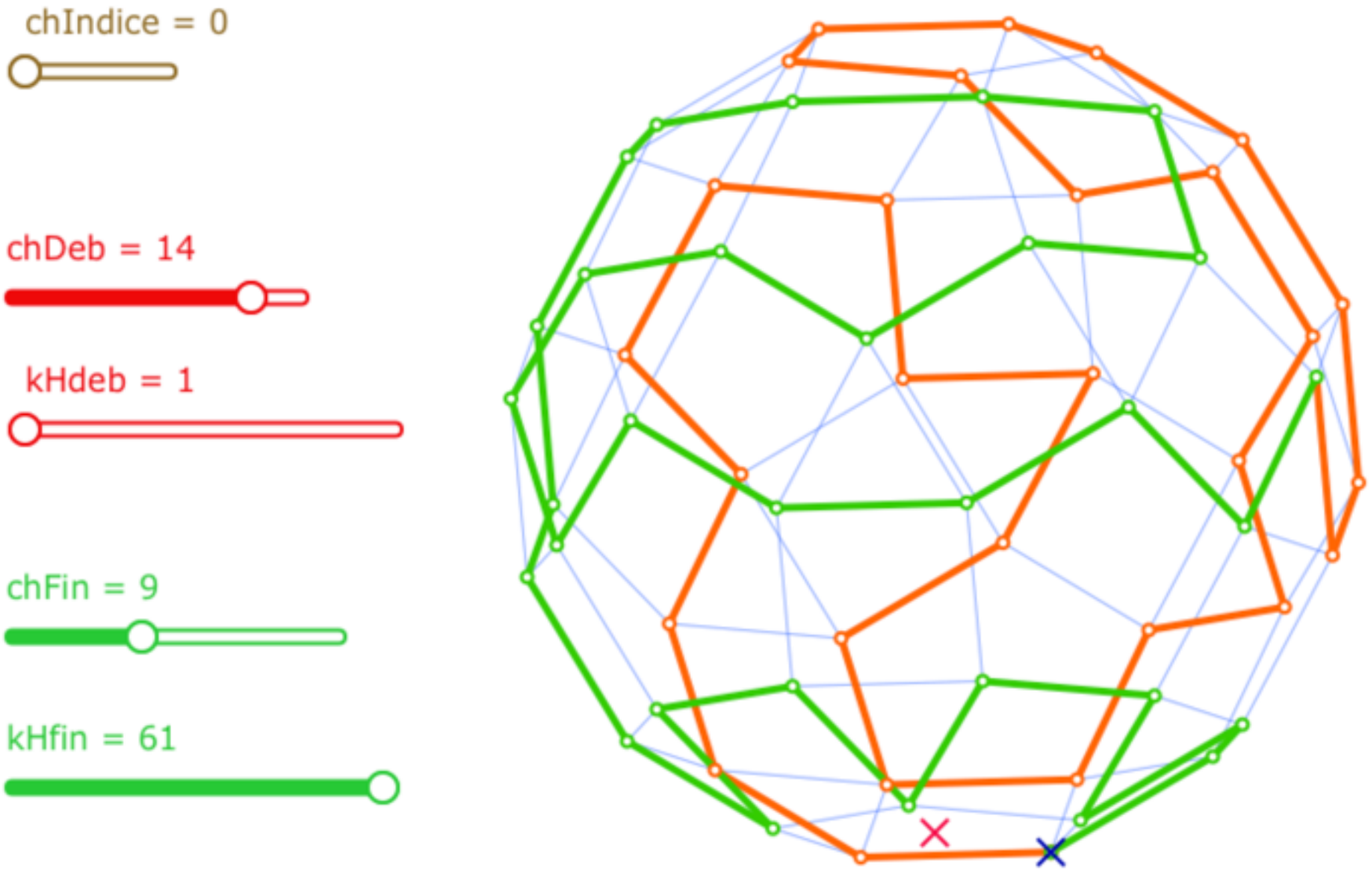
Pour chaque partition, on crée plusieurs variantes du parcours de ces 30 sommets, que l'on obtient par les deux curseurs de choix **chDeb** pour les 30 premiers (17 possibilités) et **chFin** pour les 30 derniers (21 possibilités). Il y a donc 357 cycles hamiltoniens dans cette figure, construits sur deux partitions d'un cycle.

Pour sélectionner la première partition mettre khdeb à 1 et kHfin à 30.
Pour la dernière, kHdeb à 30 et kHfin à 61.

On peut afficher ou non les **indices des points** dans la trace de A.

Le code de ces traitements est dans le point O (croix rouge, origine du repère 3D).

Figure dynamique 8.9 – 347 cycles hamiltoniens sur le rhombicosidodécaèdre



On tourne le trièdre simplement au doigt... et tout ceci, y compris le polyèdre... à la tortue...

Récursivité et tortue

Dans ce chapitre, on reprend quelques thèmes très connus, et on regarde comment les illustrer avec la tortue.

Généralement, les algorithmes sont les mêmes que dans un autre langage. Ici sans utiliser de grammaire, par exemple comme on pourrait le faire en python ou autres. Mais parfois non, en particulier si on veut colorier des parties de la figure, comme la fractale de Pythagore où il faut fermer les polygones.

Ce chapitre va être l'occasion de présenter la possibilité de faire parcourir un point d'une trace de la tortue (Hilbert 3D) voire même d'afficher une partie seulement du parcours récursif (Césaro).

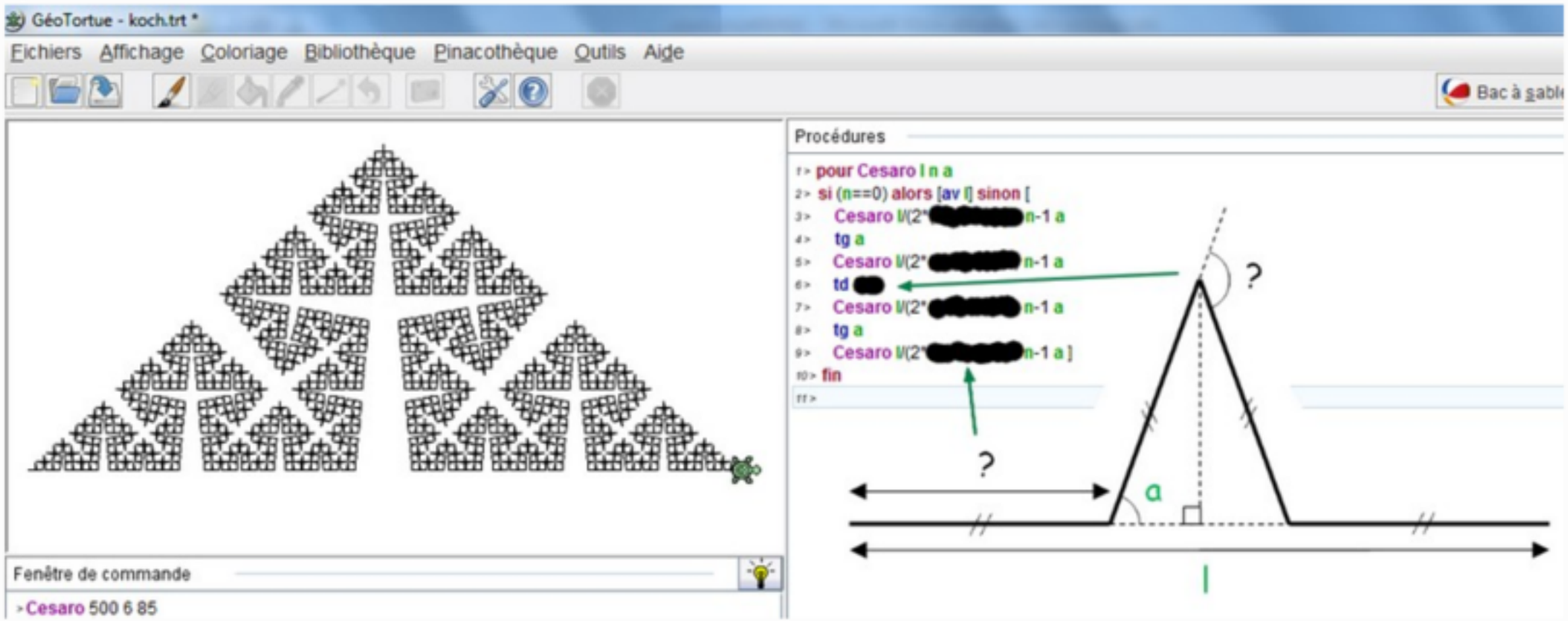
Et, à l'occasion du réglage d'un paramètre, la réaction en temps réel de la tortue va encore nous offrir quelques belles surprises qui méritent à elles seules ce chapitre.

Certaines copies d'écran montrent des réalisations avec un grand nombre d'objets, elles ont toutes été réalisées sur ordinateur. Sur une tablette, on évitera de se frotter à certains extrêmes.

Courbe de Césaro

Commençons par ce grand classique avec lequel notre collègue Julien Pavageau fait travailler ses élèves sur les angles en collège, avec GéoTortue, comme on le voit sur la copie d'écran de cette page (copie d'écran qui nous sert en même temps de définition si nécessaire).

La procédure est pratiquement identique à Koch avec cependant deux difficultés : déterminer le 2ème angle que doit faire la tortue ainsi que la longueur des nouveaux segments.



Avec des connaissances de fin cinquième vous devriez pouvoir déterminer le 2ème angle que j'ai caché et avec des connaissances de fin de quatrième vous devriez également pouvoir déterminer le nombre par lequel il faut diviser la longueur...

Cela paraît plus long à écrire, mais c'est exactement le même code qu'avec GéoTortue.
C'est juste écrit dans un environnement dynamique car on peut déplacer A et B.

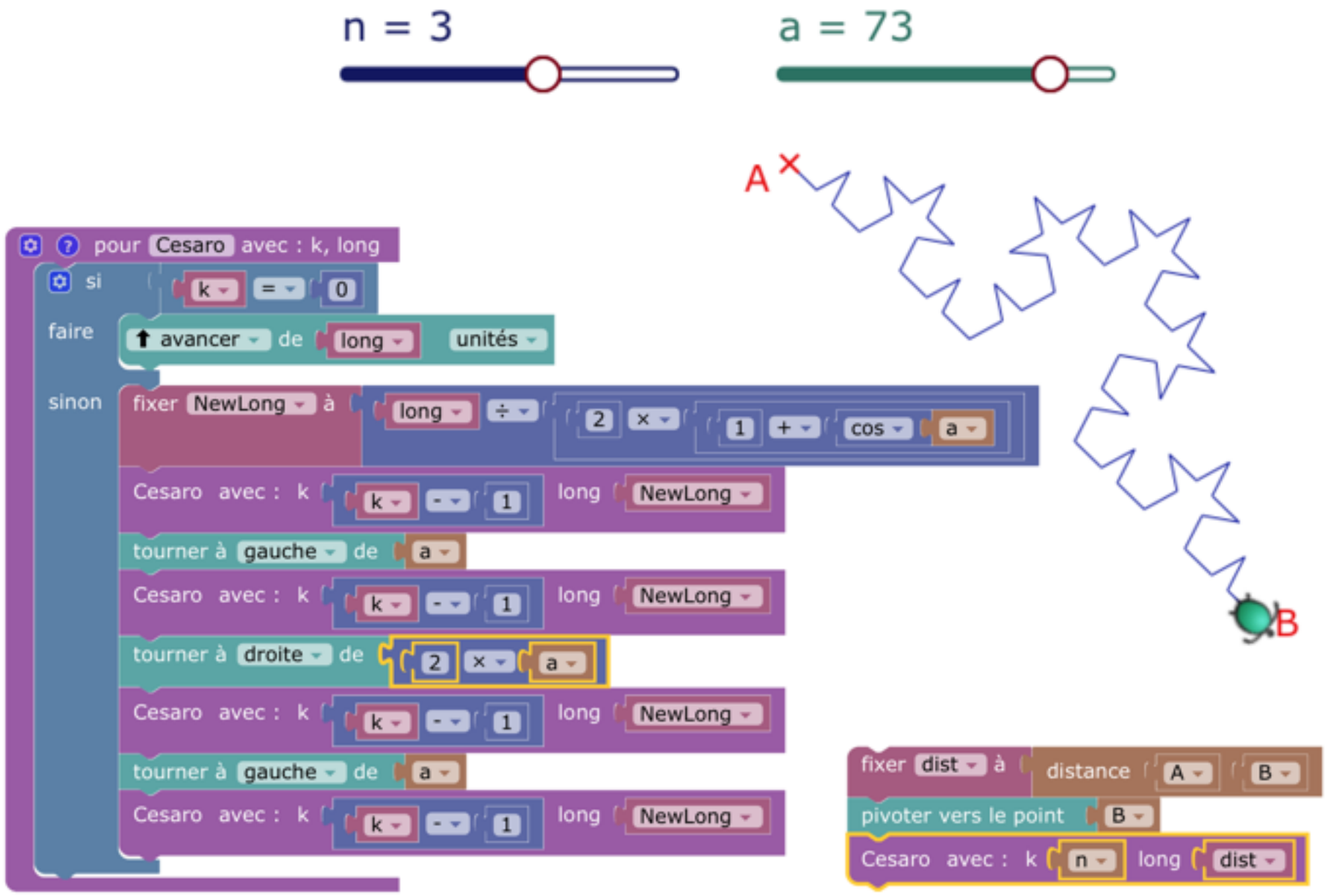
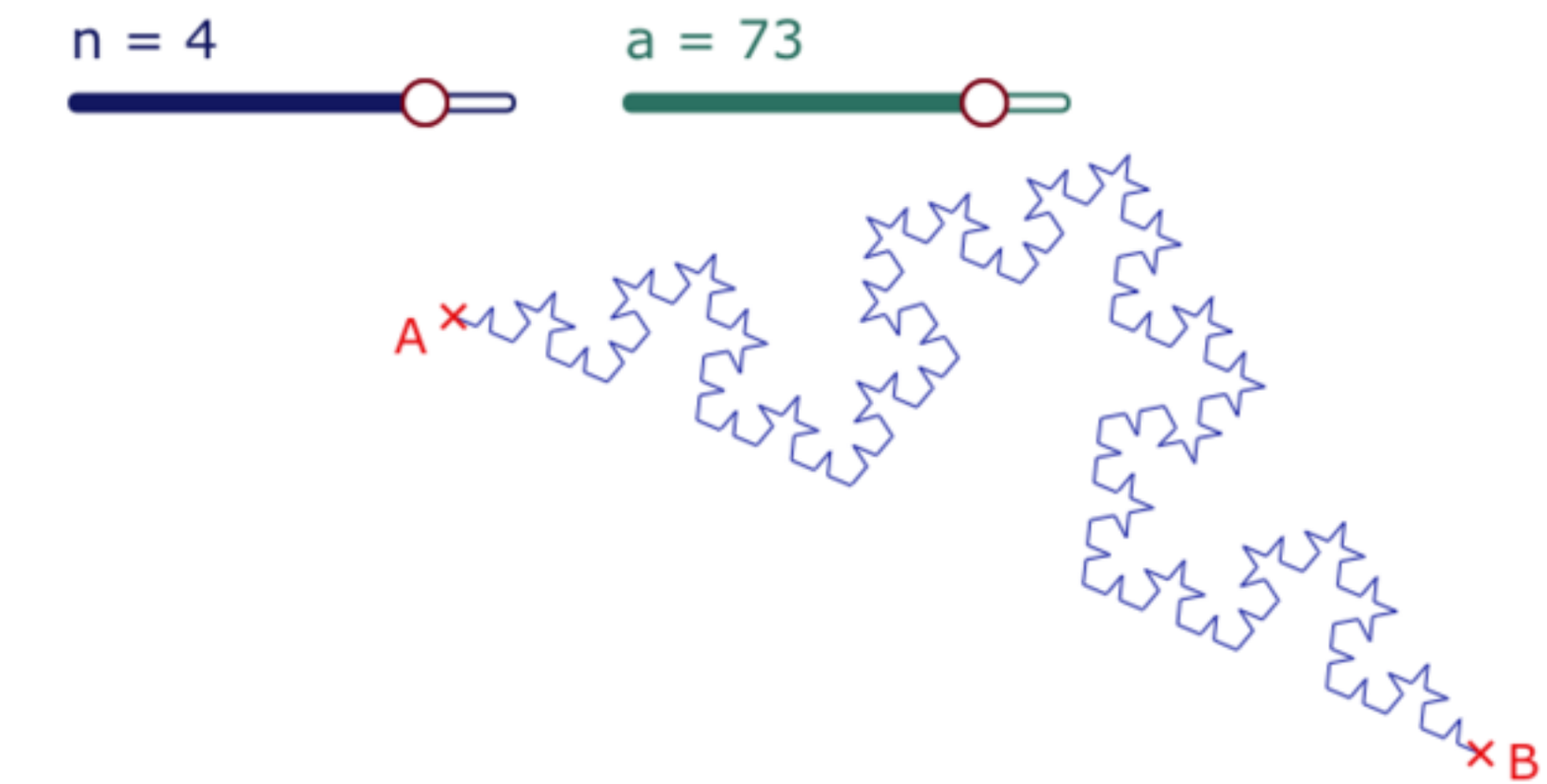


Figure dynamique 9.1 – Courbe dynamique de Césaro



*Le code est en A. Figure en mode **consultation** à l'ouverture.*

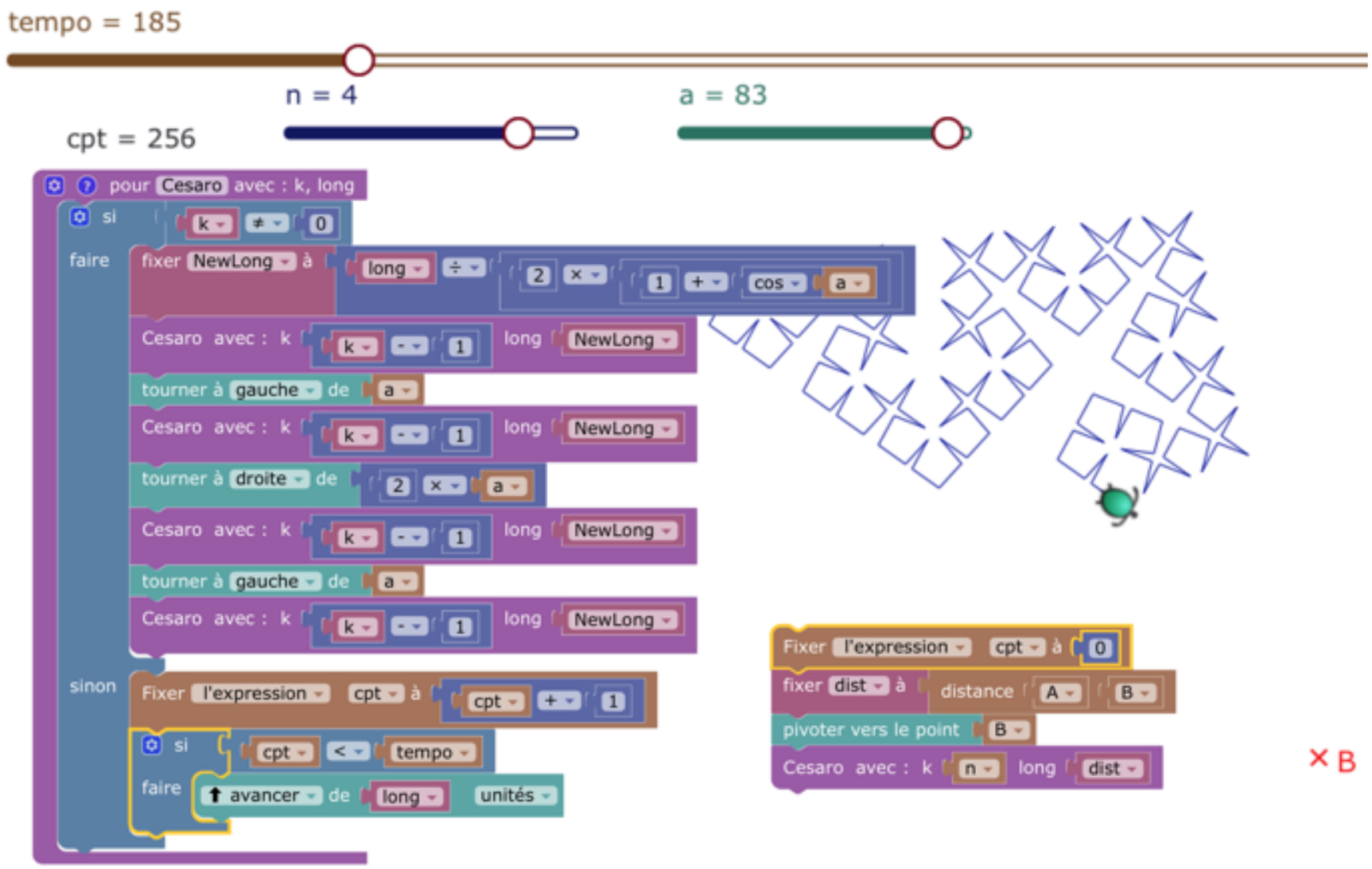
Césaro - 2. Afficher une partie de la courbe - Mode « pas à pas »

L'aspect dynamique de la tortue est assez particulier car il permet même d'afficher une partie de la courbe, pourtant définie récursivement. Voyons comment modifier le code pour cela.

On compte, avec le compteur **cpt**, initialisé à 0, le nombre de segments de la courbe.
La courbe complète contient 4^n segments. Et c'est ce qui est affiché dans la mise en œuvre de la procédure : tout est calculé.

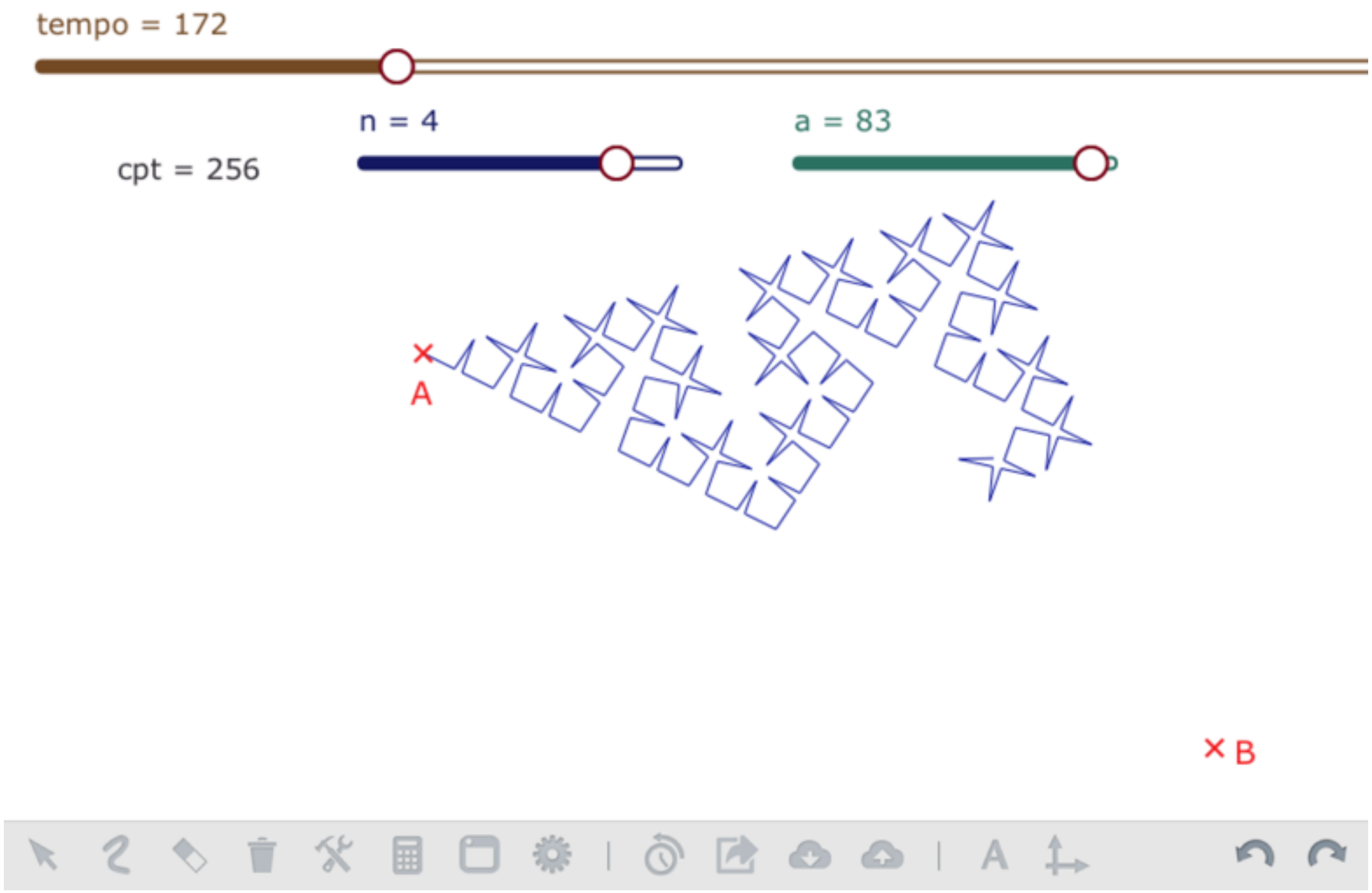
Seul l'affichage **avancer de long** n'est réalisé que si le compteur, au moment où il est calculé, est plus petit que le paramètre **tempo**, modifiable au curseur.

En pratique, la courbe se dessine au fur et à mesure que l'on tire le curseur **tempo**, ce qui est assez extraordinaire en terme de temps réel récursif.



Dans la figure suivante, le curseur est limité à 800 pour visualiser **à l'unité** l'avancée de la tortue pour n=4. Donc pour n=5 on ne va pas jusqu'au maximum de la trace (1024 segments).

Figure dynamique 9.2 – Césaro – Mode pas à pas

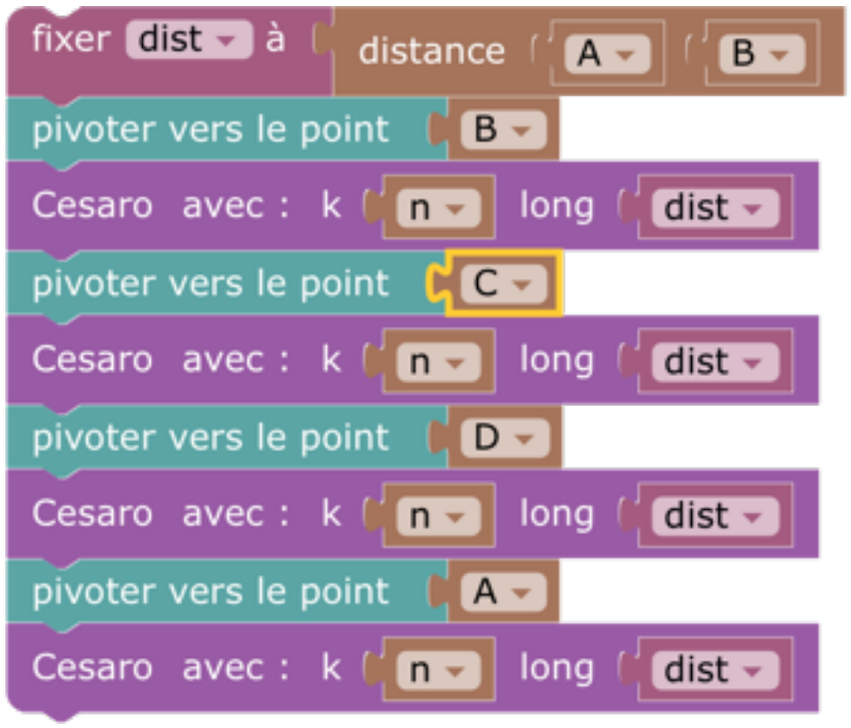


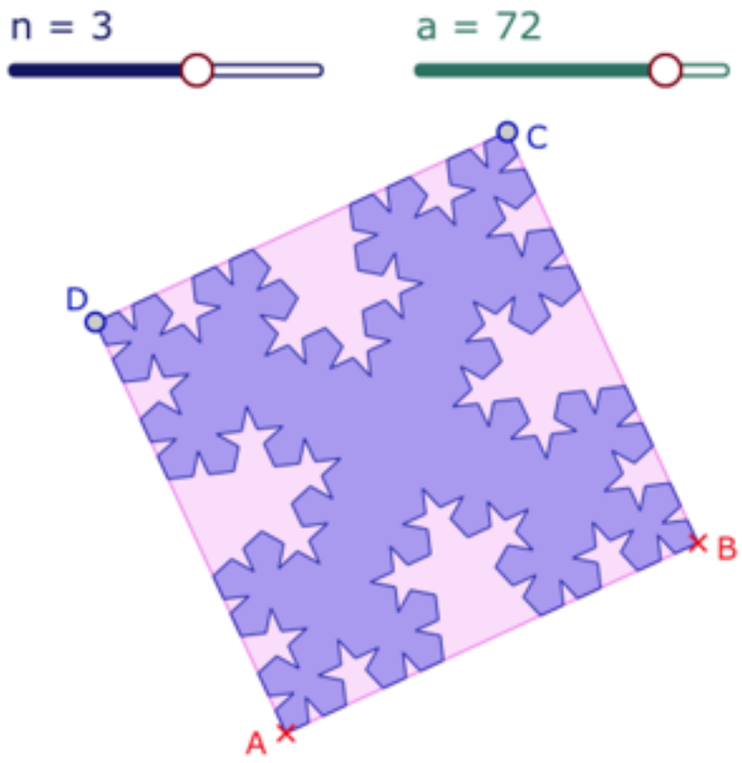
*En mode **consultation** à l'ouverture. Mettre tempo à 50 environ et passer à n = 3. On peut tester aussi n=5.*

Césaro - 3. Dans un carré

On peut appliquer Césaro sur les côtés d'un polygone. Voici le cas du carré.
C'est l'occasion de rappeler que, quand les objets sont cachés – comme le sont les points C et D dans la figure suivante – ils n'apparaissent pas dans les pop-up menu de Blockly (c'est un choix ergonomique pour les figures lourdes).

Si on devait remettre ces points dans la partie initialisation, il faudrait les rendre à nouveau visibles préalablement.





Pour éviter cette question des points cachés, on aurait pu construire une liste **sommets** = [A, B, C, D] et placer la procédure dans un compteur.
Dans la page suivante, figure avec un carré, et dans la suivante, avec un pentagone.

Question : pour quel angle l'aire coloriée en bleu est-elle minimale ? Est-ce lié à n ? Si oui quelle limite quand n tend vers l'infini ?

Figure dynamique 9.3 – Césaro à l'intérieur d'un carré

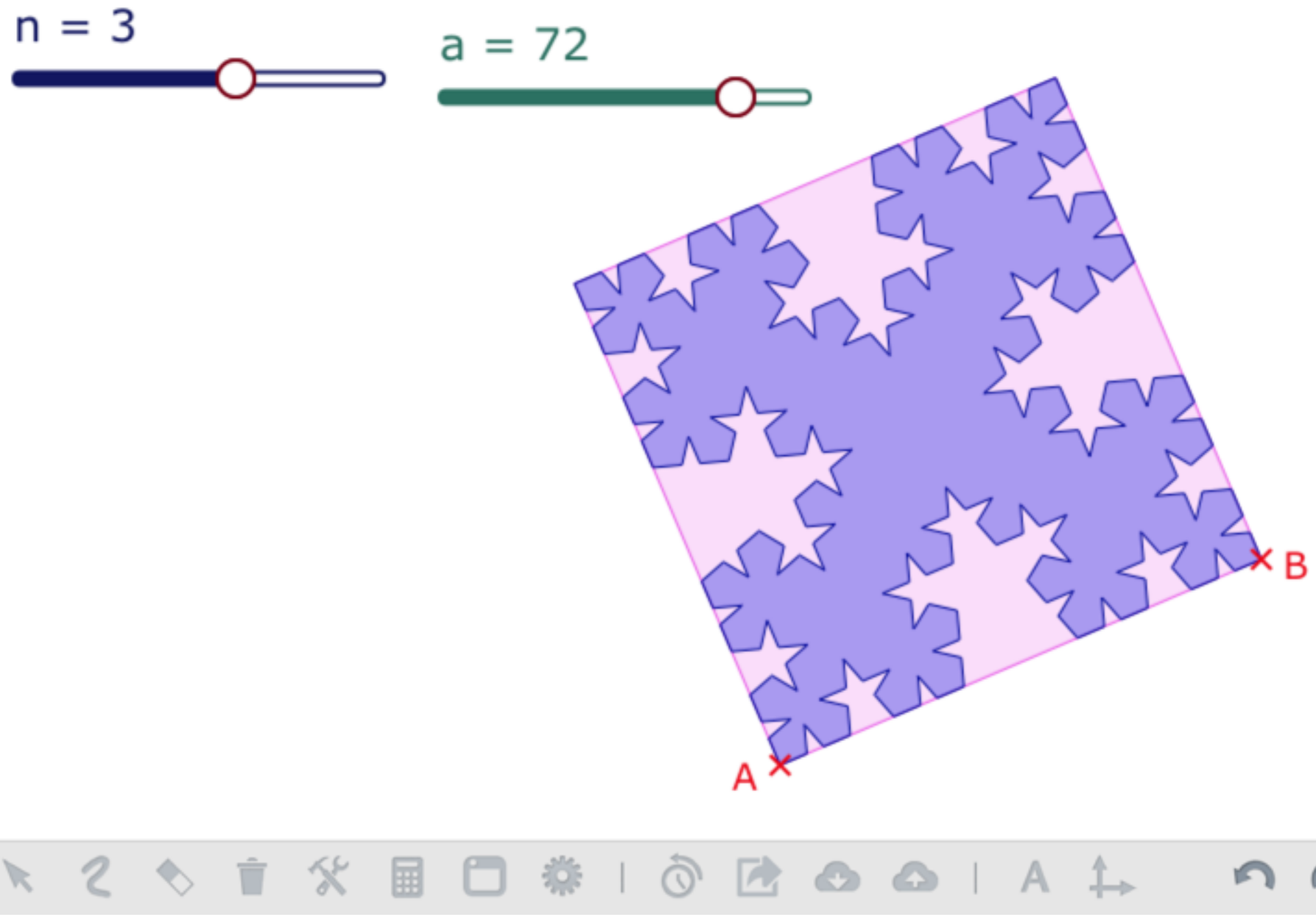
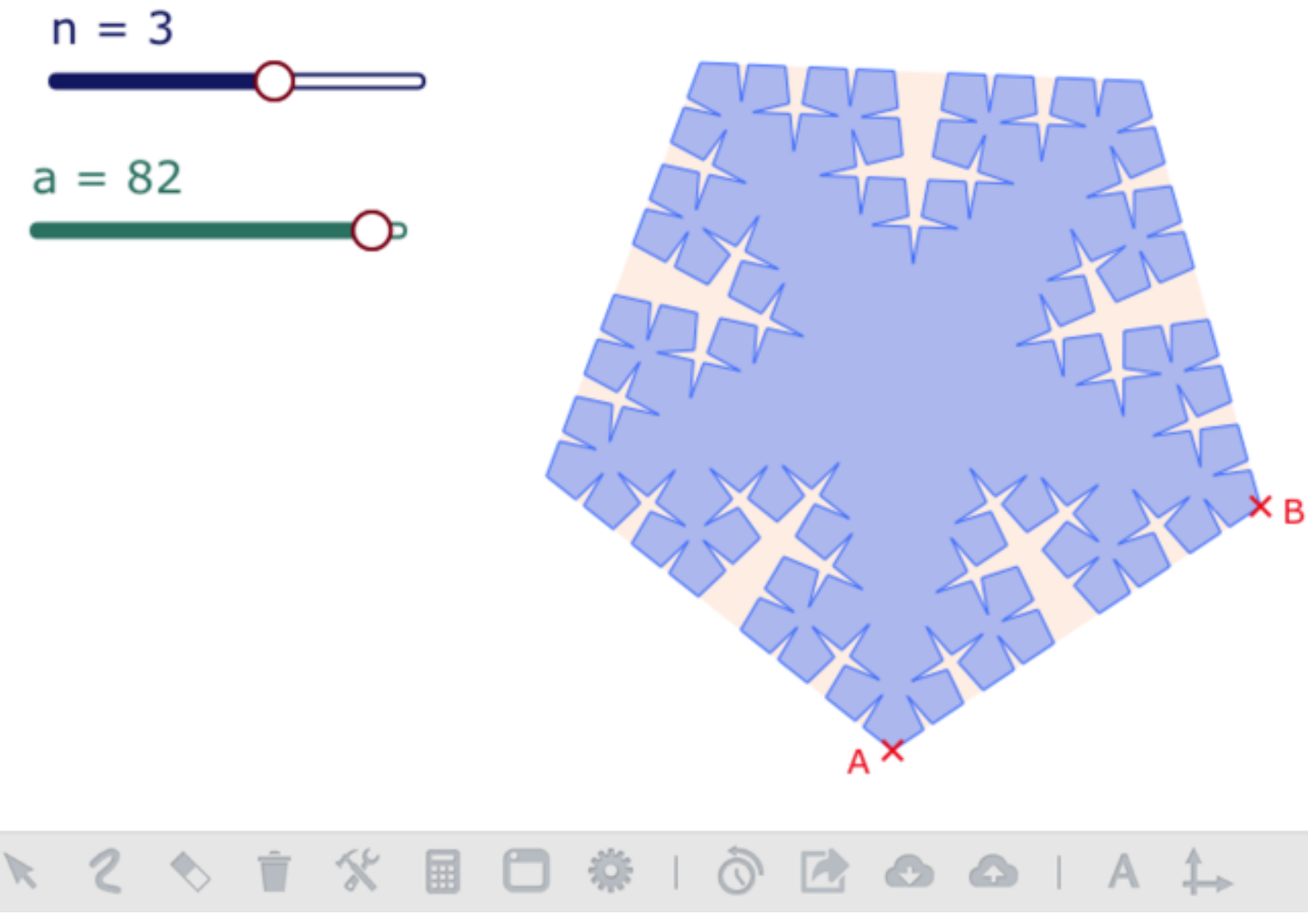
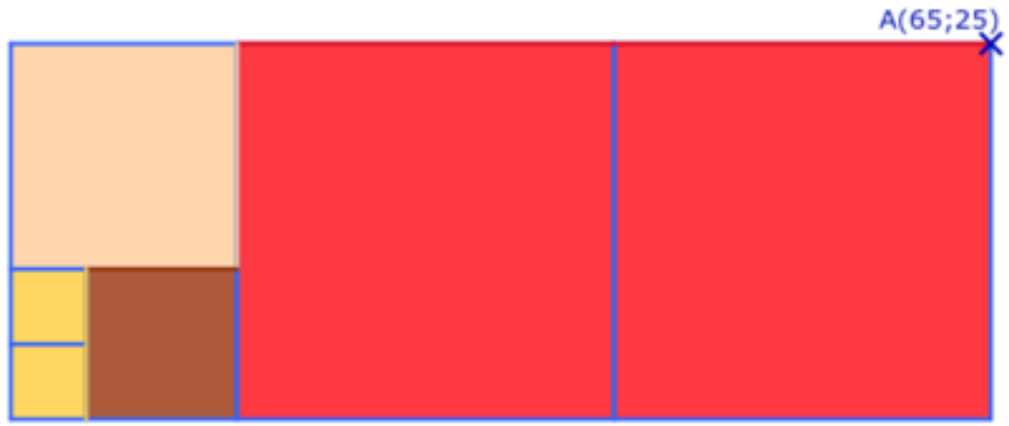


Figure dynamique 9.4 – Césaro - Avec un pentagone



PGCD Graphique - Anthyphérèse du rectangle

Historiquement, l'algorithme qui porte son nom est présenté par Euclide comme un algorithme géométrique de découpage d'un rectangle en carrés de côtés décroissants, successivement d'un côté et de l'autre du rectangle, pour trouver le carré de plus grand côté qui permet de paver le rectangle : c'est le pgcd des mesures (entières) des côtés du rectangle.
Le procédé est appelé **l'anthyphérèse** (soustractions alternatives).
Ci-contre on voit deux carrés de coté 25, puis un de 15, puis de 10 et deux carrés de côté 5. Le pgcd de 65 et 25 est donc 5.



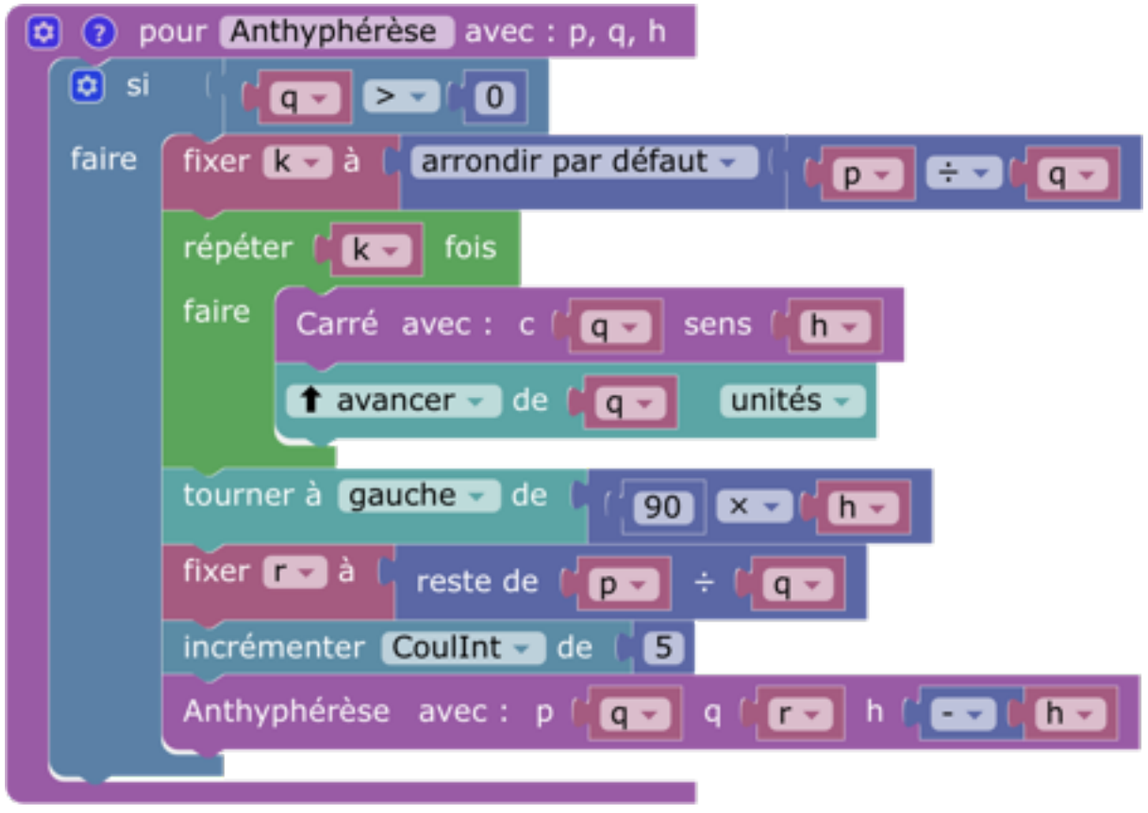

```
def anthyphérèse(a, b):
        """ a et b sont les dimensions du rectangle """
        if a == b:
                carré(a)
        else:
                while a > b:
                        carré(b)
                        forward(b)
                        a -= b # signifie: a = a-b
                forward(a)
                left(90)
                anthyphérèse(b, a)

longueur, largeur = 416 , 184 # affectations multiples
rectangle(longueur, largeur)
anthyphérèse(longueur, largeur)
```

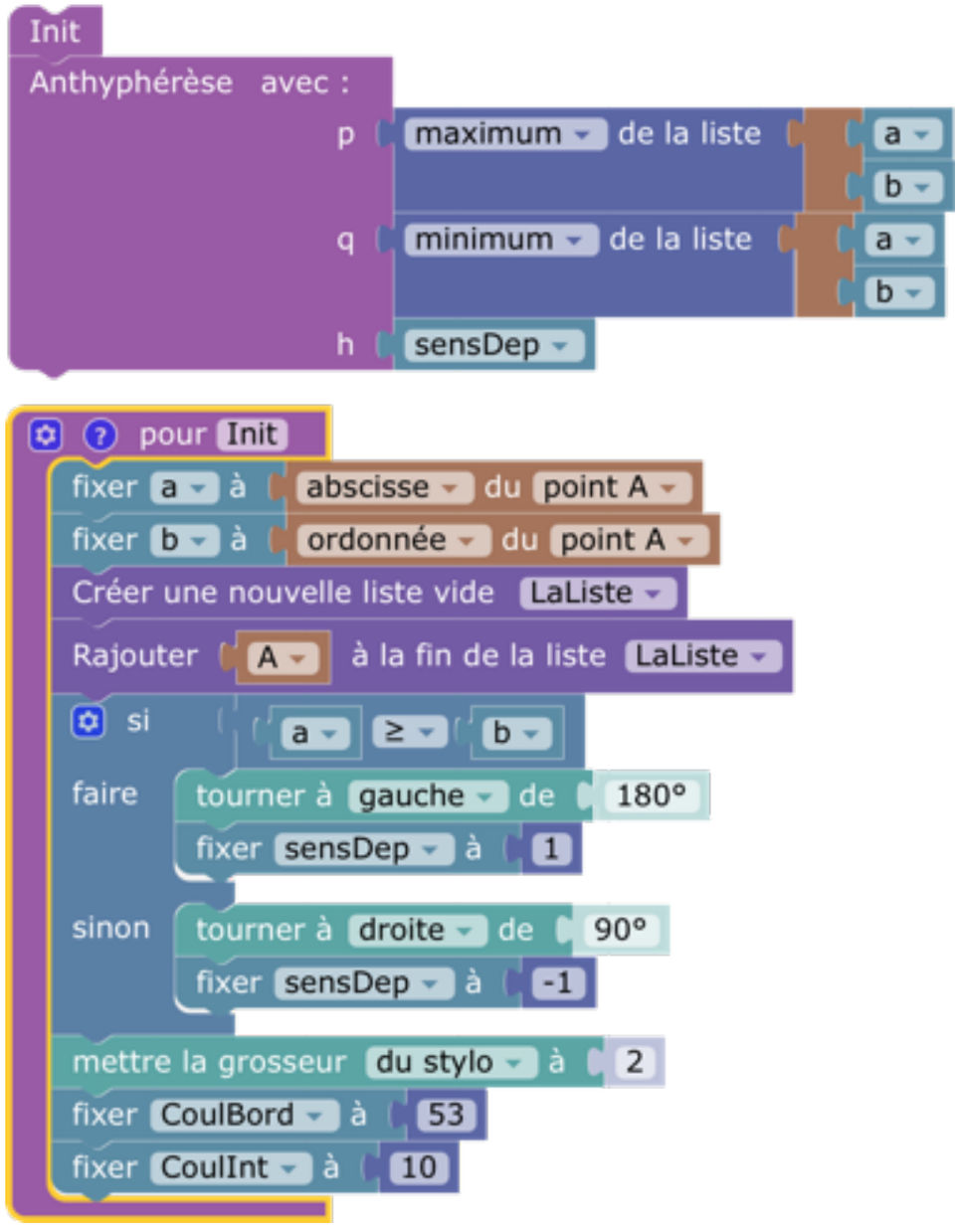
Bien avant que Python soit devenu le langage retenu pour le lycée, parce que l'on faisait de l'arithmétique, Guillaume Conan proposait déjà le code ci-contre, en mars 2012, avec la tortue de Python, dans cet article de MathémaTICE. On remarquera que **le critère de sortie** de la récursivité est a=b car **le test de la boucle tant que** est une inégalité stricte. De même, la tortue n'a pas à s'orienter au départ selon la forme du rectangle, car dans cet algorithme l'abscisse est toujours la longueur du rectangle. Comme ici on veut faire quelque chose de dynamique, il va falloir paramétrer un peu plus cette « soustraction alternative ».

En essence, l'algorithme est construit comme une succession de soustractions, d'où son nom grec. Cela se traduit par la présence de cette boucle « **tant que** » au coeur de l'algorithme. Or dans un contexte de construction en temps réel de la trace de la tortue par rapport au déplacement des blocs, il est délicat de construire un «tant que» avec les blocs de la tortue de DGPad. Ce thème est l'occasion de préciser comment contourner cette difficulté, qui plus est, dans une situation subtile d'appel récursif. Aussi, dans un premier temps, nous allons voir l'algorithme avec la division classique, procédé efficace, mais plus éloigné de l'esprit initial soustractif de cet algorithme.

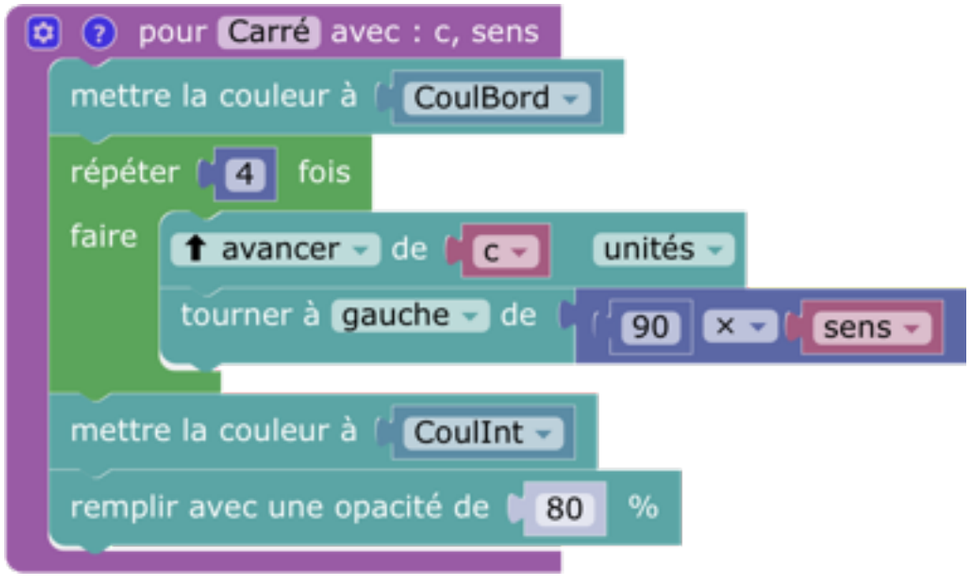
Initialisation et orientation de la tortue -
Procédure avec un « répéter »

La tortue part de A, et se dirige vers l'origine, d'abord dans le sens de l'abscisse si la longueur est en abscisse. La tortue doit aller le long des ordonnées si la longueur est en ordonnée. Donc la procédure générale doit comporter un paramètre qui lui permet de tenir compte et transmettre le sens de la tortue.

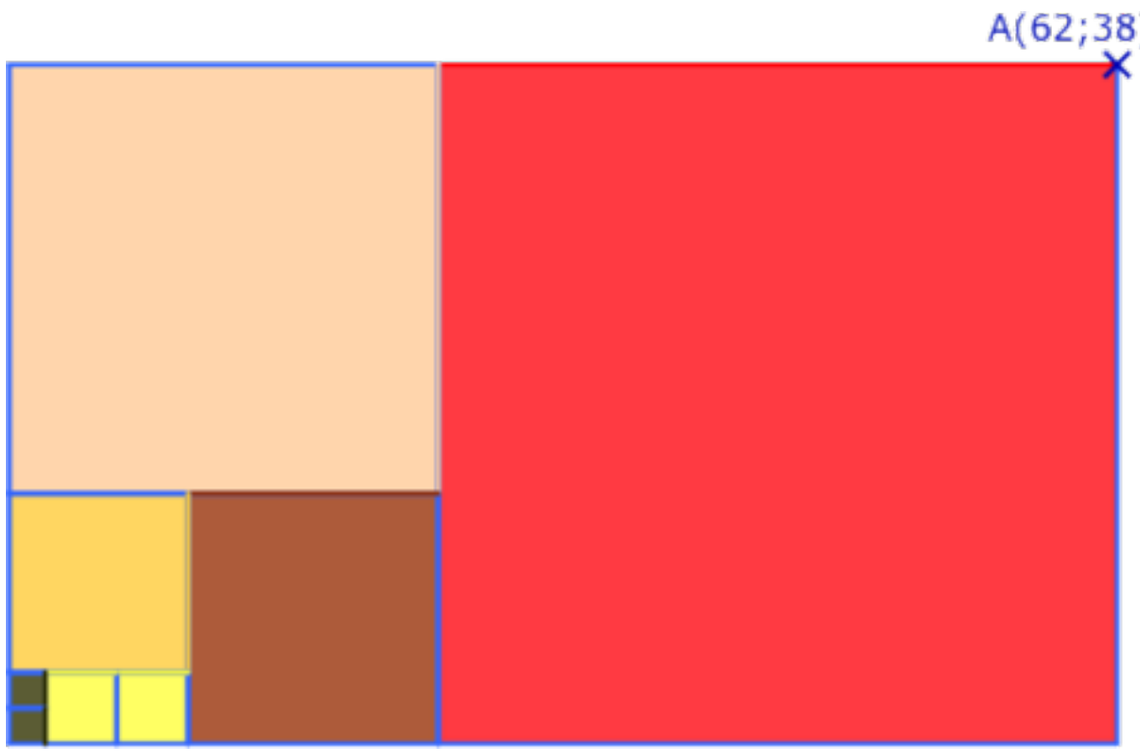




Contrairement à la version Python, ici, on teste (critère d'arrêt) si le second terme est non nul. Dans ce cas, il peut être un diviseur. Le quotient **k** de p par q donne le nombre de carrés à construire dans le sens précisé par **h**.
Ainsi le « tant que » de la version précédente devient un « répéter ». Le changement de signe du troisième paramètre dans l'appel de la procédure rend bien compte du nom du procédé.



Le dynamisme de la figure – déplacement du point A – amène la procédure **Carré** à devoir tenir compte du sens de tracé et de l'orientation de la tortue. D'où le paramètre **sens**.
On notera la gestion de la couleur du contour et de l'intérieur.
Dans la figure de la page suivante, ne pas hésiter à placer le point A comme ci-contre, tel que l'ordonnée soit supérieure à l'abscisse.



Une version avec le calcul des coefficients de Bézout est développée au chapitre suivant sur les listes Blockly.

Rectangle de droite :

64 = 2x28+8
28 = 3x8+4
8 = 2x4 donc
pgcd(28,64)=4

Rectangle de gauche :

62 = 38+24
38 = 24+14
24 = 14+10
14 = 10 + 4
10 = 2x4+2
4 = 2x2
pgcd(62,38)=2

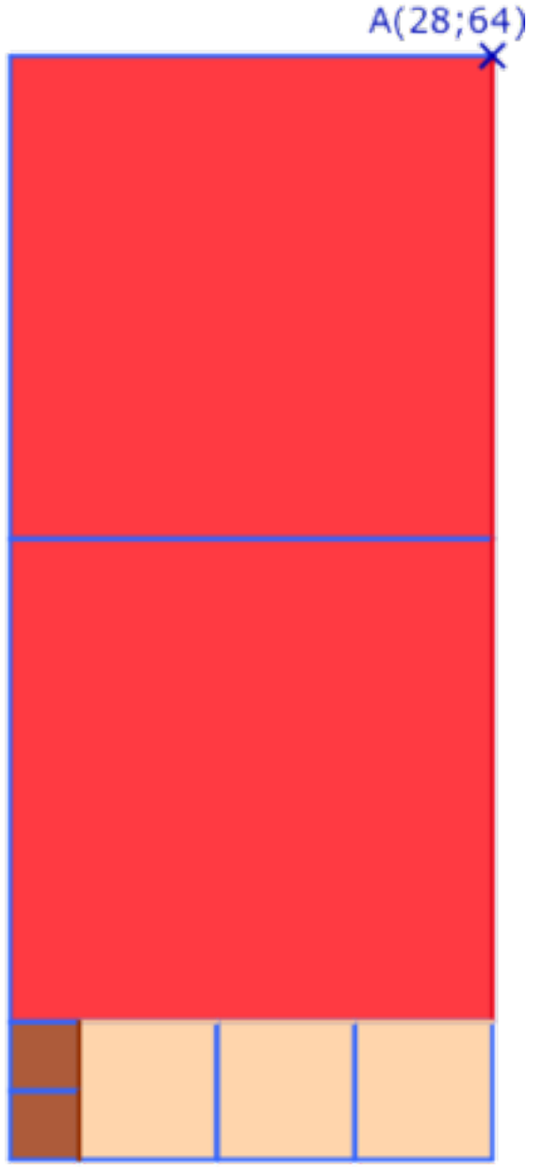
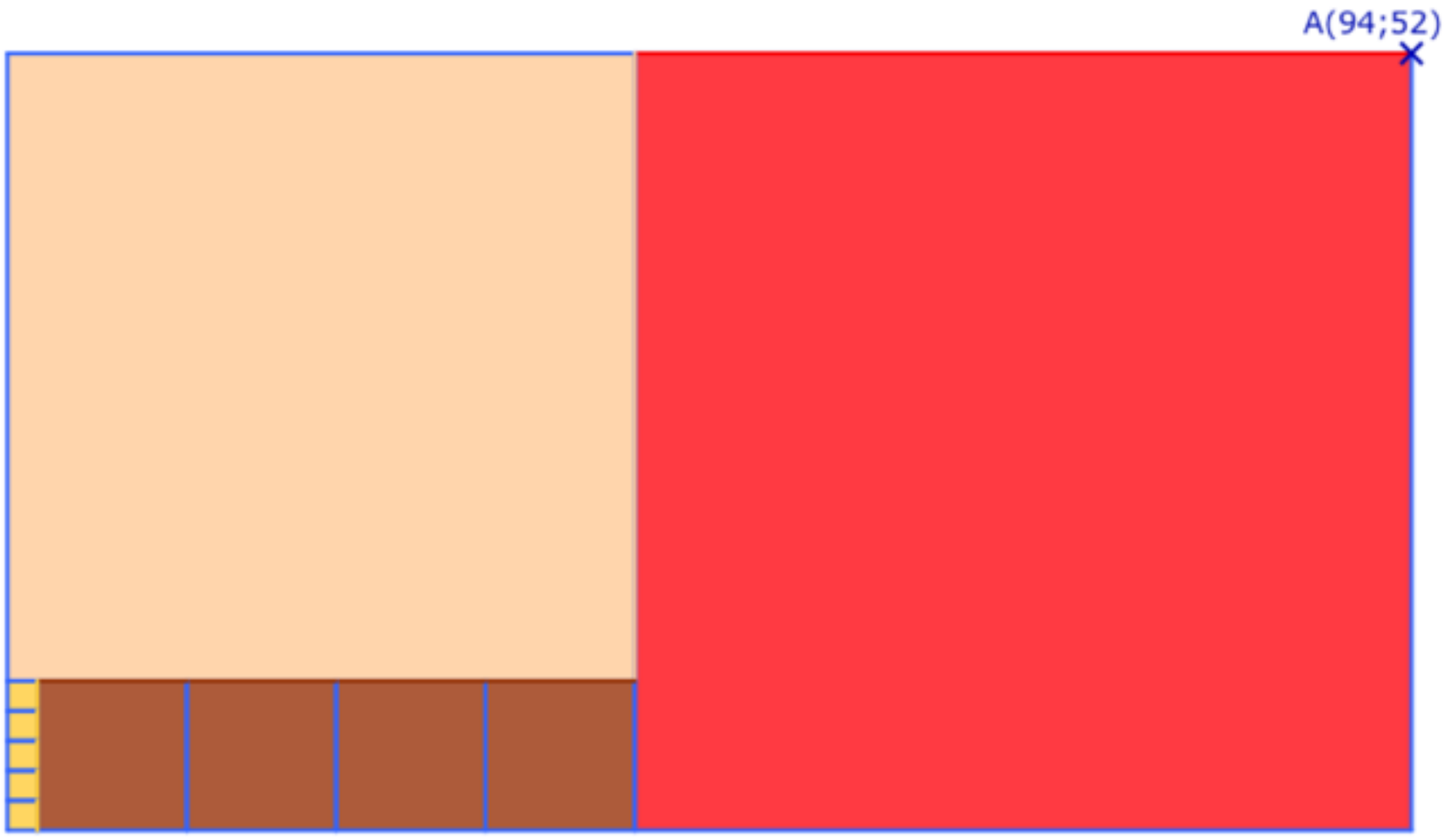
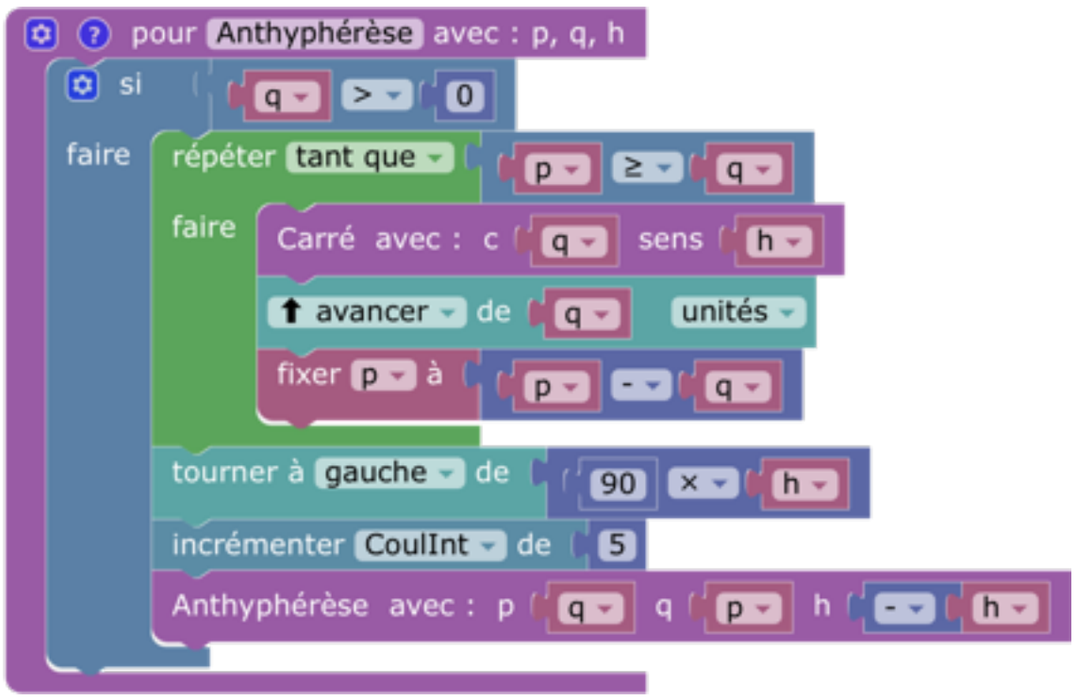


Figure dynamique 9.5 – PGCD – Approche graphique

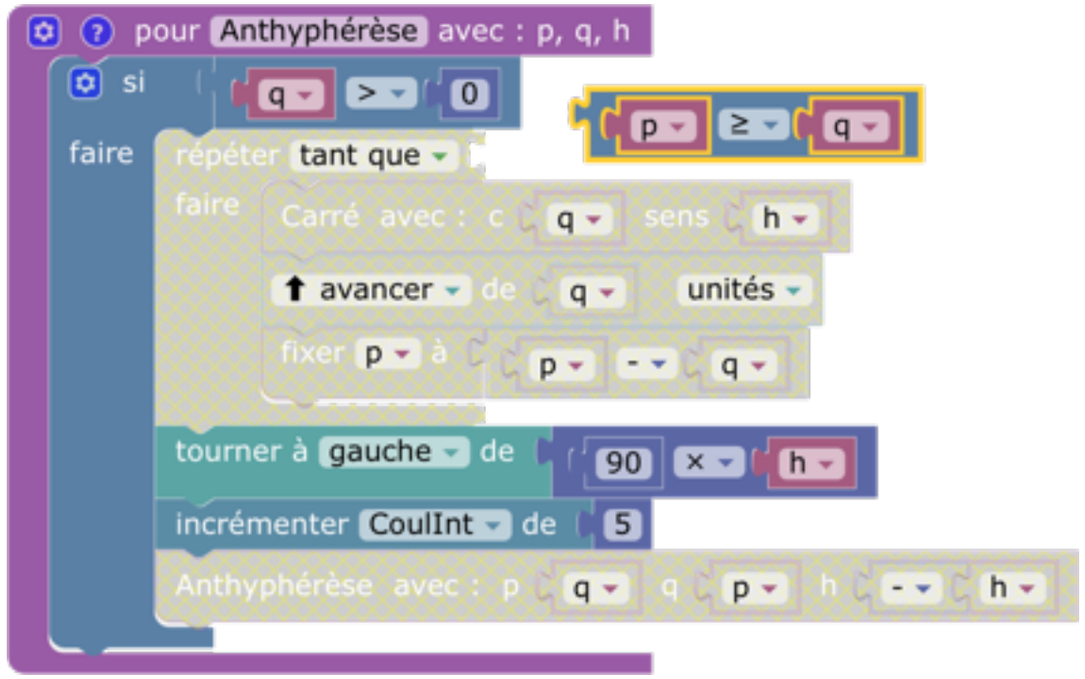


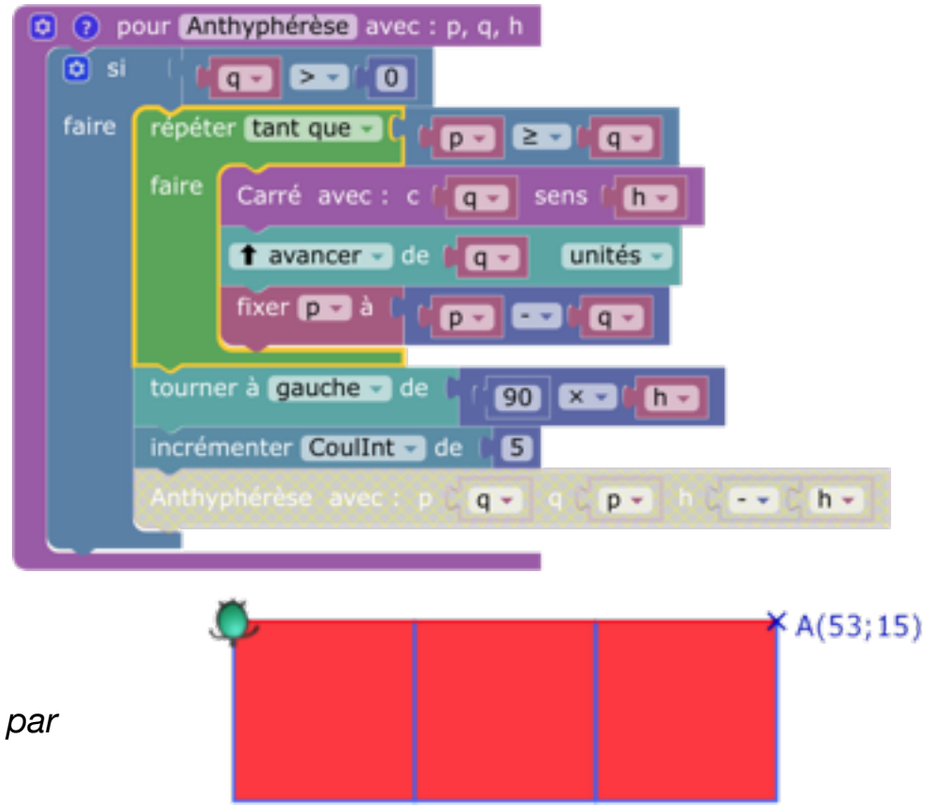
Retour sur l'anthyphérèse, avec une boucle « tant que »

On conserve le même test d'arrêt (**q>0**). Cela signifie que le test interne doit être large (**p≥q**) car il faut produire 0 pour arrêter la récursivité.



Mise en oeuvre pratique du « tant que » – Il faut *désactiver* la boucle pour la remplir ... **mais aussi** l'appel récursif.

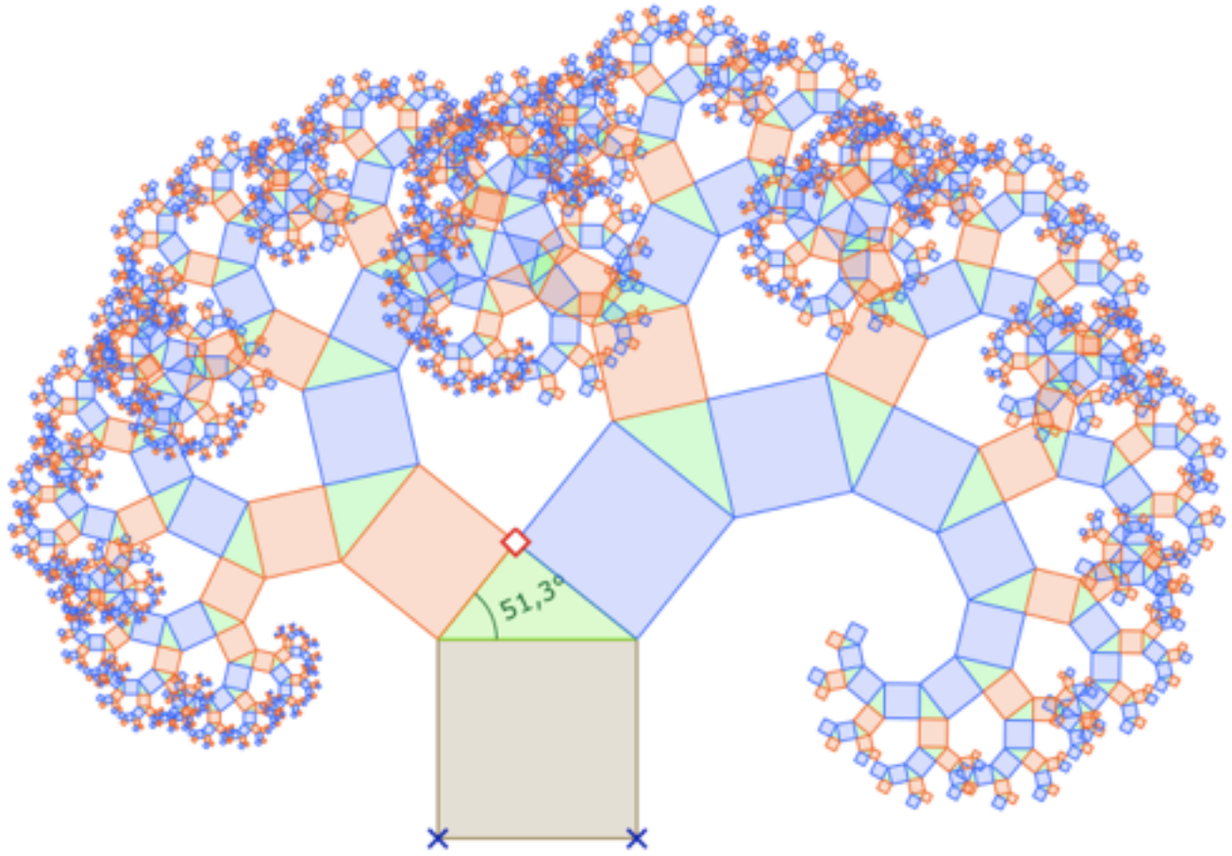




Ci-contre exemple d'une seule application de la procédure, et position de la tortue, par désactivation de l'appel récursif.

Fractale de Pythagore

La classique **fractale de Pythagore** va être l'occasion d'une nouvelle particularité surprenante de la réalisation en temps réel. L'objectif est de faire la figure suivante à la tortue.



A la différence de à ce que l'on a fait au chapitre 1, géométriquement avec une macro construction où l'angle était donné avec un curseur, cette fois l'angle se manipule sur la figure par le point rouge du triangle.

Dans un premier temps, on va commencer par construire les carrés, puis ensuite seulement on ajoutera les triangles. L'important, pour colorier, est de fermer les polygones. Toujours pour la colorisation, on ne pose le crayon que pour faire un polygone (carré ou triangle), d'où les instructions de *levée du crayon* pour ne pas perturber cette colorisation.

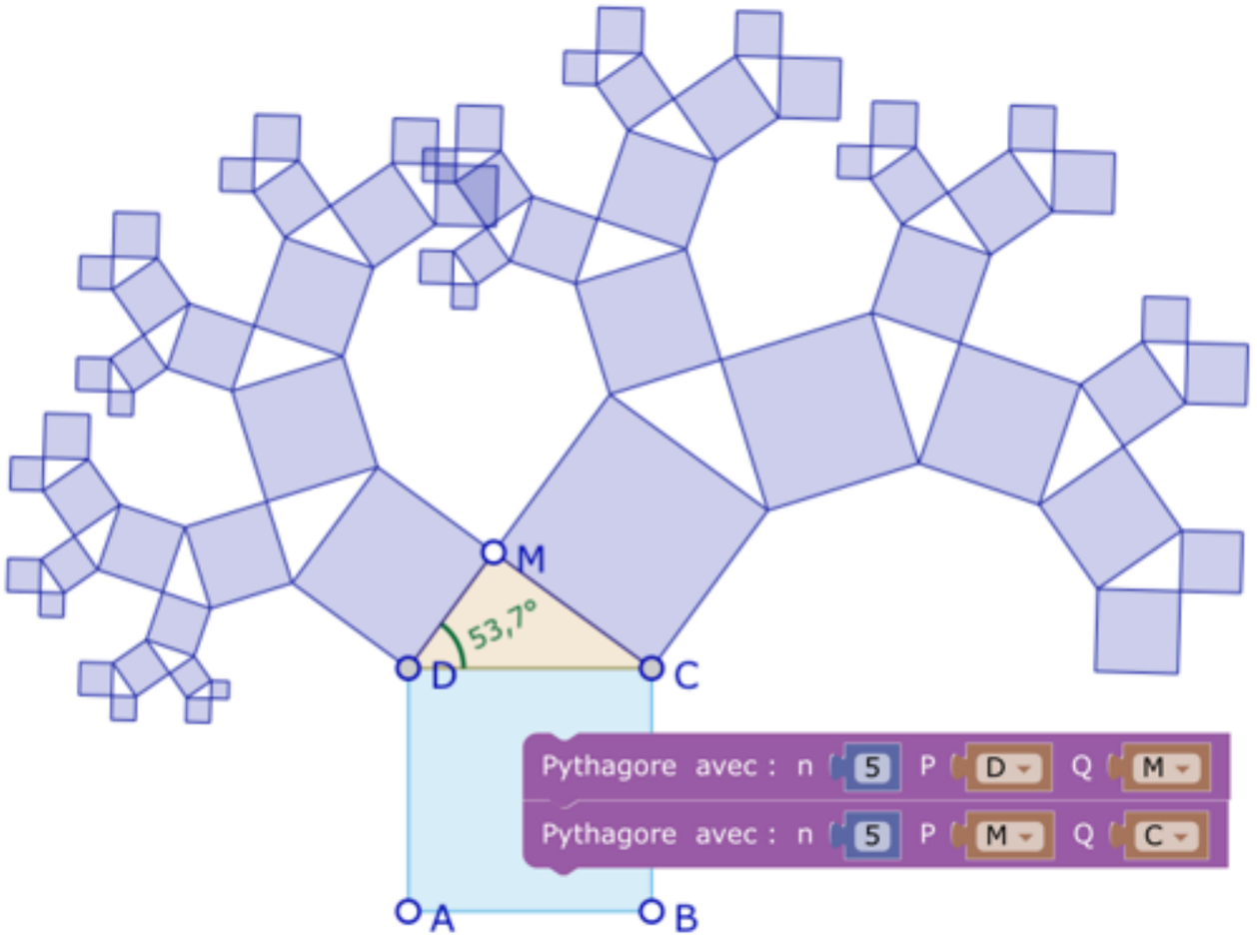
Cette approche comporte une certaine **algébrisation** de la démarche car on utilise systématiquement la **position de la tortue pour nommer localement** des sommets qui vont permettre l'appel récursif.

Le principe est le suivant : on construit – par exemple ci-dessus – sur le côté gauche orienté [PQ] de l'angle droit du triangle, d'une part le carré direct orange [PQRS] et ensuite le point U, sommet du triangle vert suivant. Puis on appelle la procédure sur le segment orienté [SU], le petit côté ci-dessus du triangle vert de gauche, puis sur l'autre segment orienté [UR].

Pythagore - Etape 1 - Version de base

A partir des points P et Q (au départ D et M), on construit **le carré direct PQRS**, algébriquement donc, que l'on rempli à 20 %. Puis on revient - en levant le crayon – et on tourne de l'angle qu'il convient en fonction de l'angle en D, pour noter - après avoir posé le crayon pour que ce soit dans la trace – **la position U** du sommet du triangle rectangle.

Il suffit alors d'appeler la procédure aux deux segments **orientés** du triangle rectangle dont [RS] est l'hypoténuse, les segments [SU] et [UR].




```
pour Pythagore avec : n, P, Q
  fixer cote à distance P Q
  lever le stylo
  rejoindre le point P
  poser le stylo
  rejoindre le point Q
  tourner à gauche de 90°
  avancer de cote unités
  fixer R à position de la tortue
  tourner à gauche de 90°
  avancer de cote unités
  fixer S à position de la tortue
  tourner à gauche de 90°
  avancer de cote unités
  remplir avec une opacité de 20 %
  lever le stylo
  reculer de cote unités
  tourner à gauche de 90 + aG
  avancer de cote × cos aG unités
  poser le stylo
  fixer U à position de la tortue
  si n > 1
  faire Pythagore avec : n n - 1 P S Q U
       Pythagore avec : n n - 1 P U Q R
```

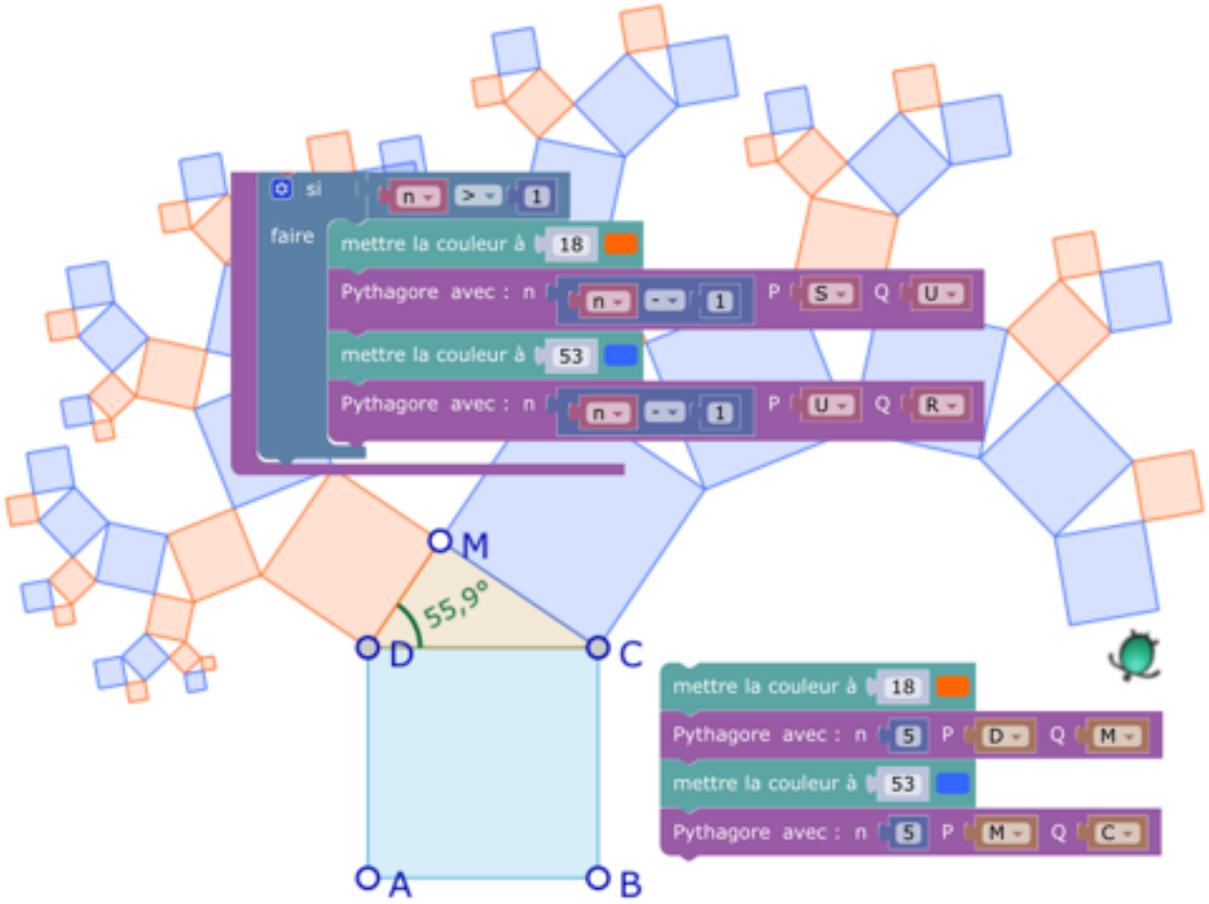
Le programme principal consiste à appeler la procédure sur le segment [DM] ce sera la construction de gauche, celle que l'on va mettre en orange, puis le segment [MC] avec les carrés bleus (branche droite).

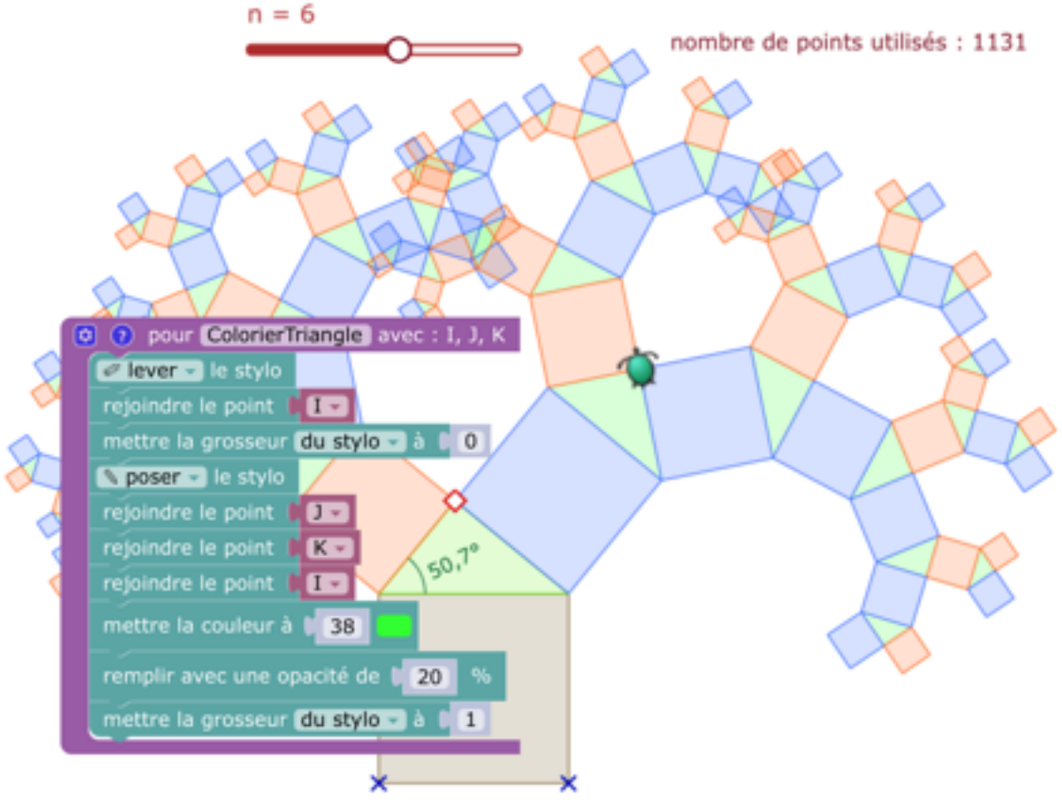
Pythagore - Etape 2 - Colorisation des carrés

Les carrés ayant été faits correctement en regard à la colorisation (lever le crayon) les carrés des branches gauche (orange) et droite (bleu) se colorient sans difficulté, il reste les triangles.

Pythagore - Etape 3 - Colorisation des triangles

Sur l'illustration ci-contre – mais aussi sur le code de la page précédente - on voit que la tortue est sur le sommet du triangle suivant.





Donc si on colorie le triangle RUS avant l'appel de la récursivité, les triangles suivants, au-dessus de chaque carré, seraient coloriés. Il ne faut donc colorier RUS qu'après l'appel récursif, soit après le second appel de Pythagore (non illustré ici). Si on affiche le code en modifiant la profondeur, on peut vérifier que, avec cette colorisation, en fin de procédure, la tortue est toujours au même endroit, sur le sommet du premier triangle de la branche bleue.

Pythagore - sur la colorisation des triangles

Quand on place le bloc **rejoindre le point**, par défaut il est en général initialisé sur un des premiers points de la figure, ici **le point A** (le point dans lequel est tout le code de la figure de la page suivante).

Si on a placé la procédure en cours de construction dans le corps de la procédure principale (par exemple pour la tester), alors, comme la figure est ajustée en temps réel, on a, à un moment donné, l'écran suivant surprenant, **une nouvelle illustration du temps réel de la tortue** de DGPad : les triangles ont A comme sommet !

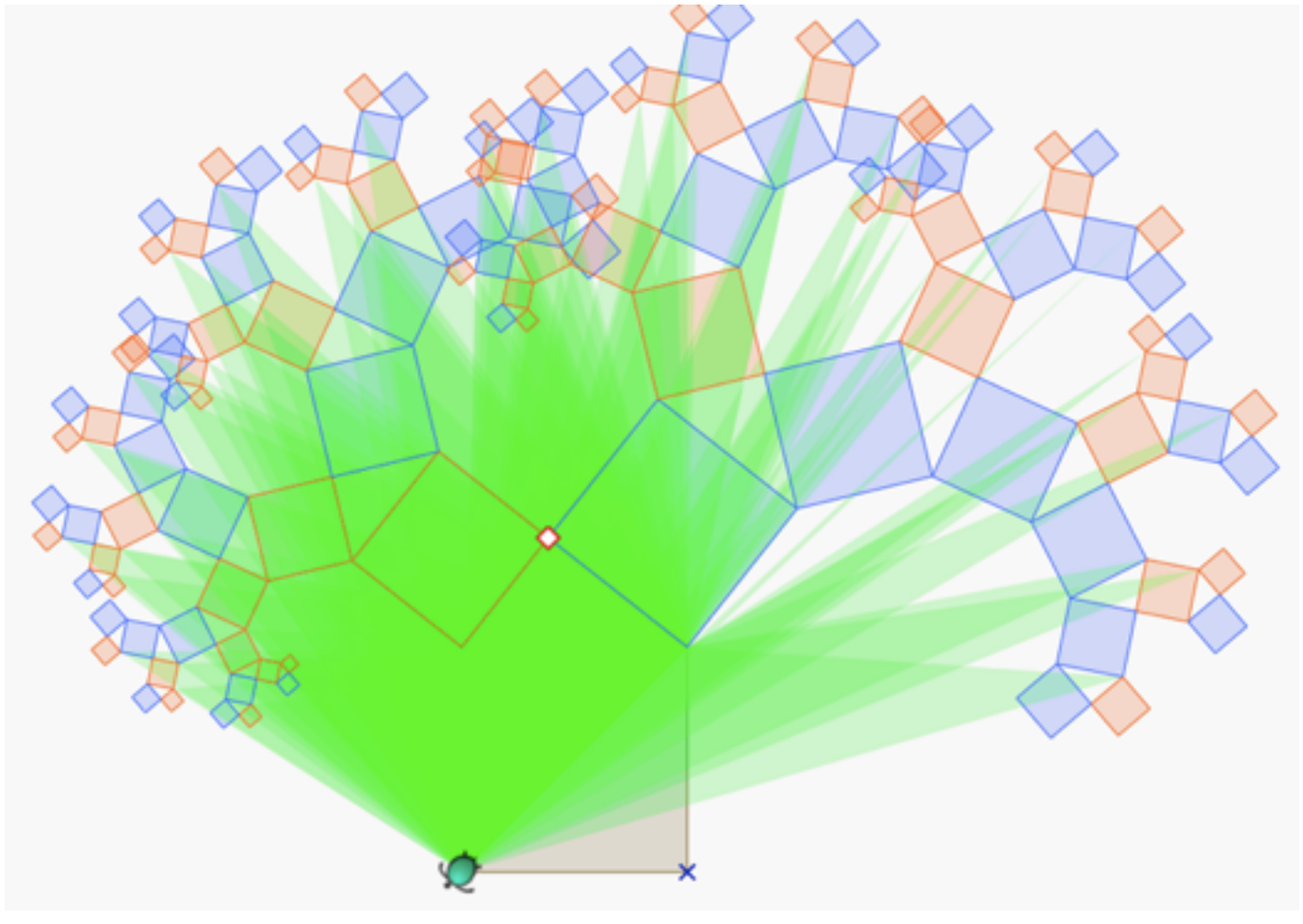


Figure dynamique 9.6 – Fractale de Pythagore

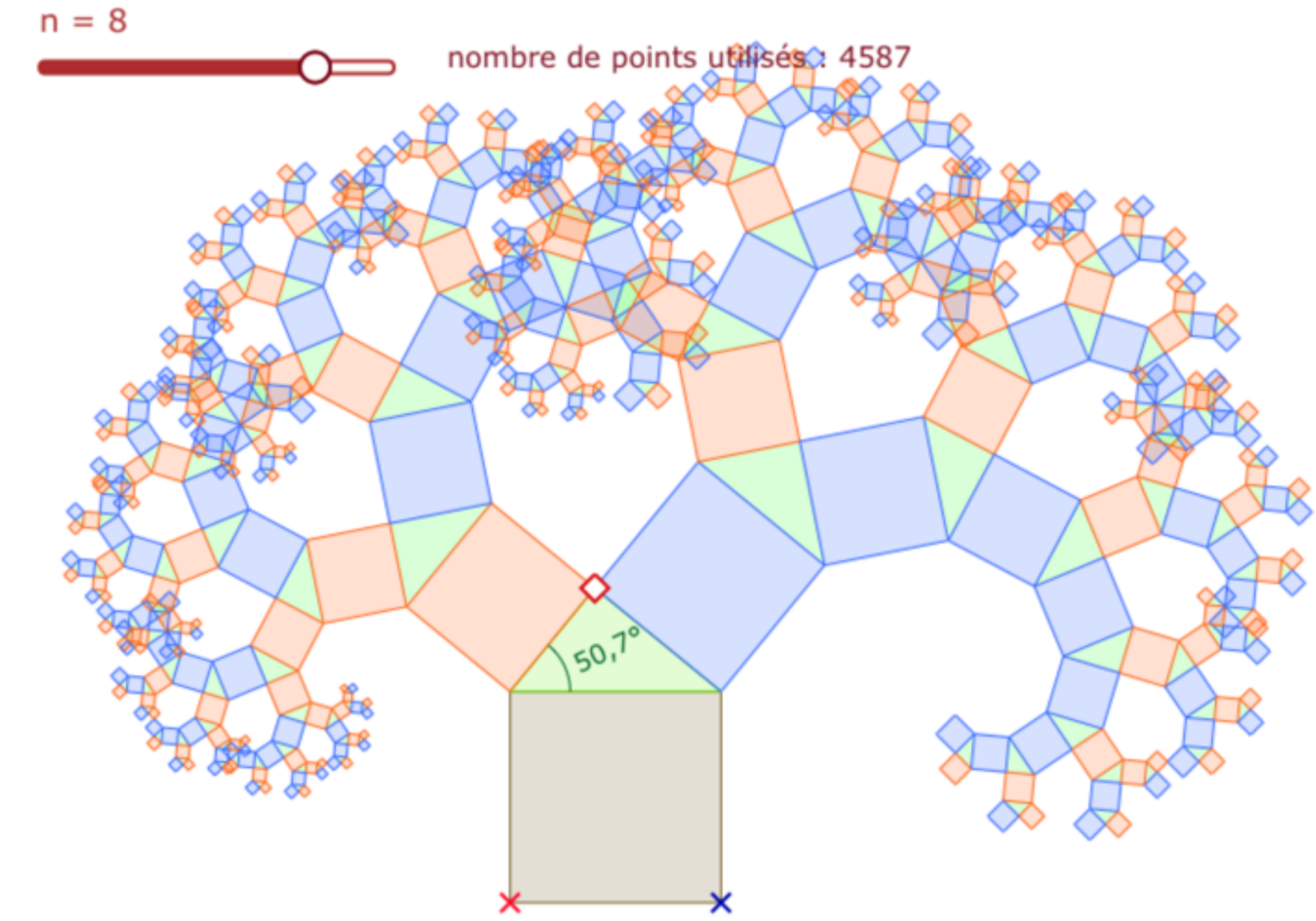


*Figure en mode **consultation** à l'ouverture. Le code est dans le point rouge du triangle.*

Courbe du dragon

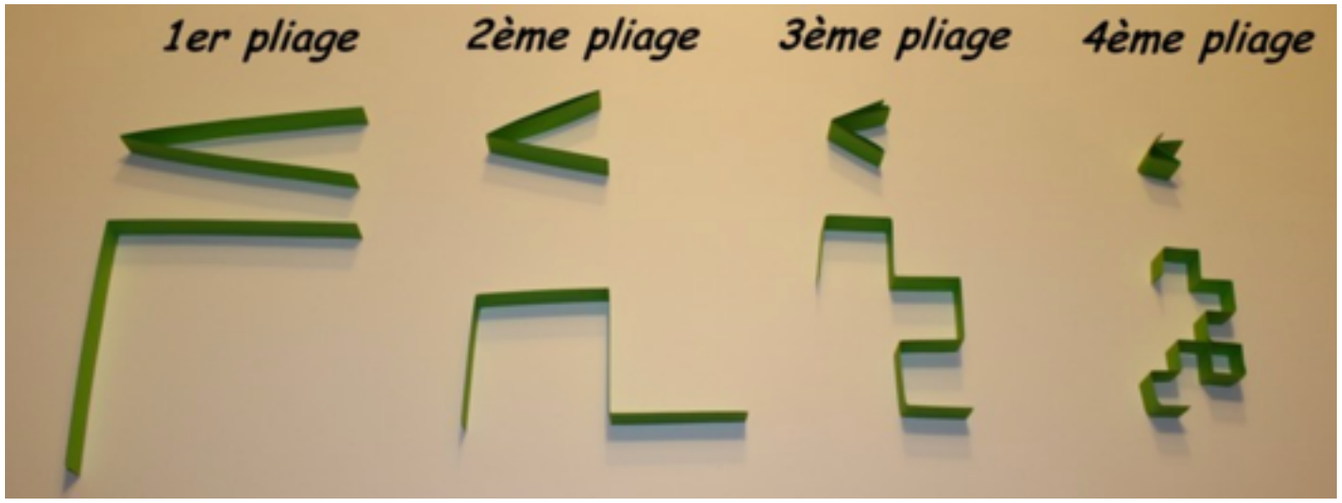
La courbe du dragon est un autre grand classique de programmation. Notre collègue déjà cité, Julien Pavageau, en a fait une page très riche, avec une présentation papier superbe (copie d'écran ci-contre), une figure DGPad, par les expressions, car faite avant l'intégration de Blockly, et une belle réalisation en GéoTortue.

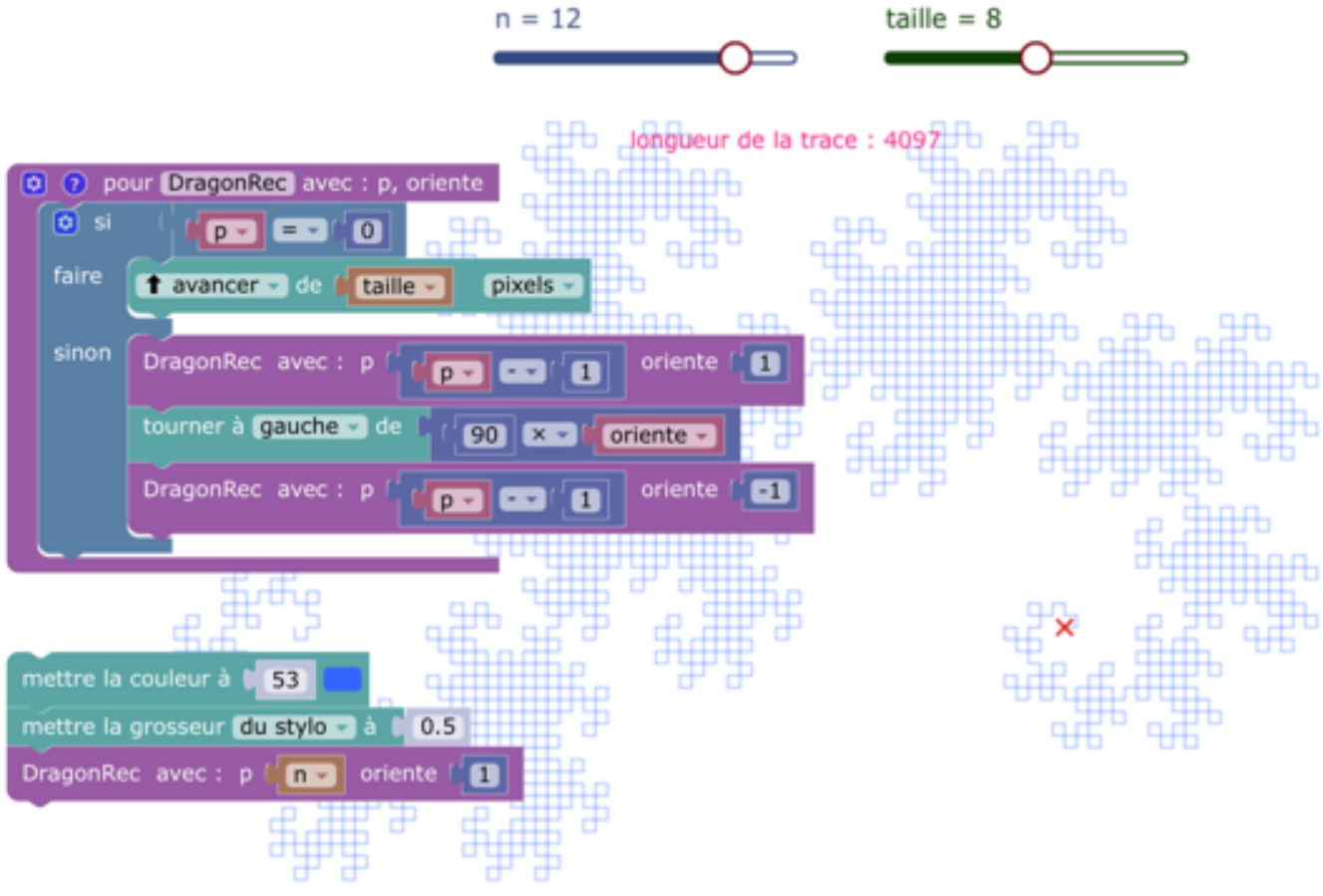
La programmation ci-contre reprend le code que l'on trouve dans tous les langages, adopté à la tortue.

L'intérêt de cette présentation est dans la suite. On sait que la courbe du dragon pave le plan en traçant 4 figures en rotation d'un quart de tour autour du point de départ.

Et c'est en voulant réaliser cette itération de rotation que l'on rencontre encore une belle surprise. En effet, on veut itérer le procédé avec une rotation. Mais il faut bien commencer par glisser un premier bloc. Alors la figure se met à jour, et donne un résultat surprenant...

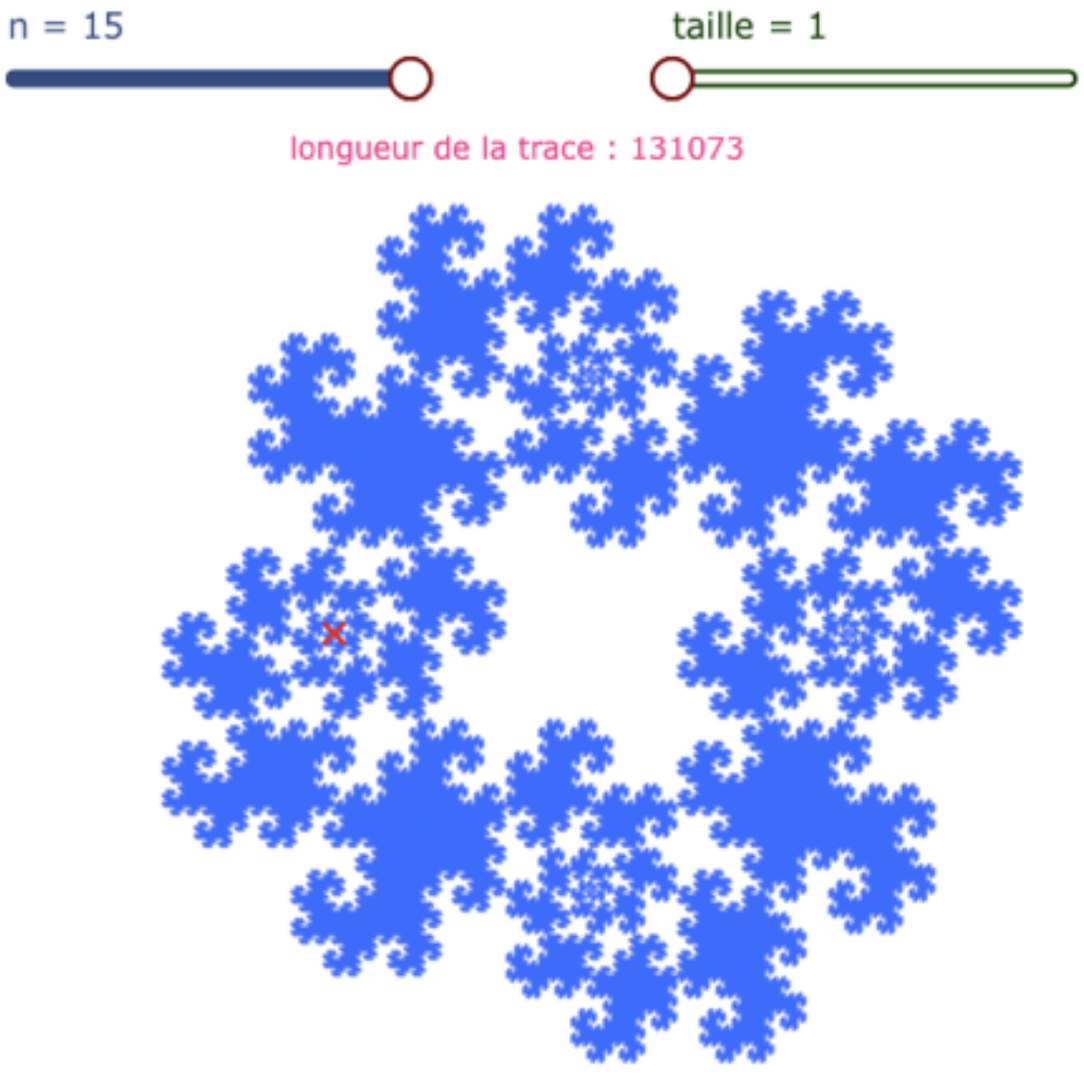
Voyons cela...

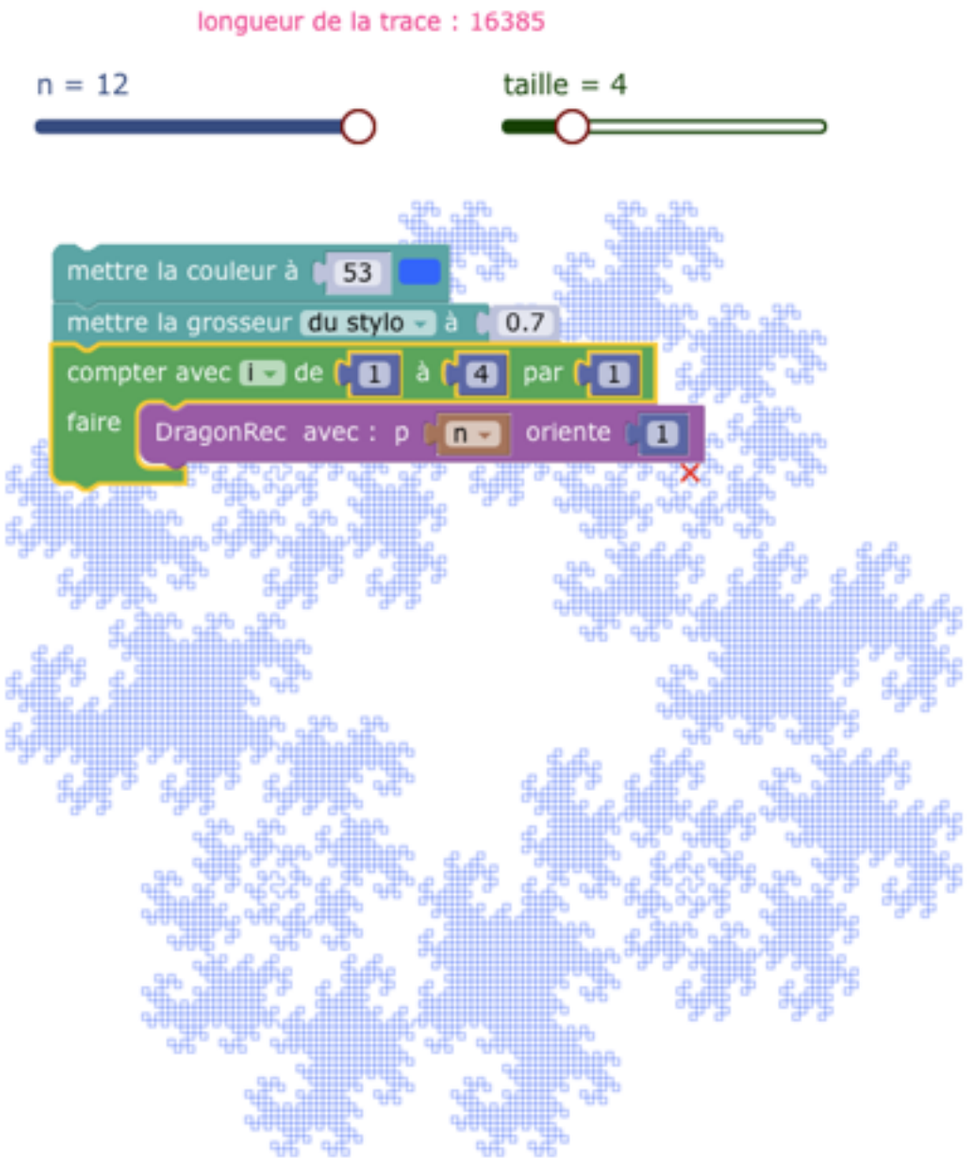




Dragon - vers l'itération de 4 rotations

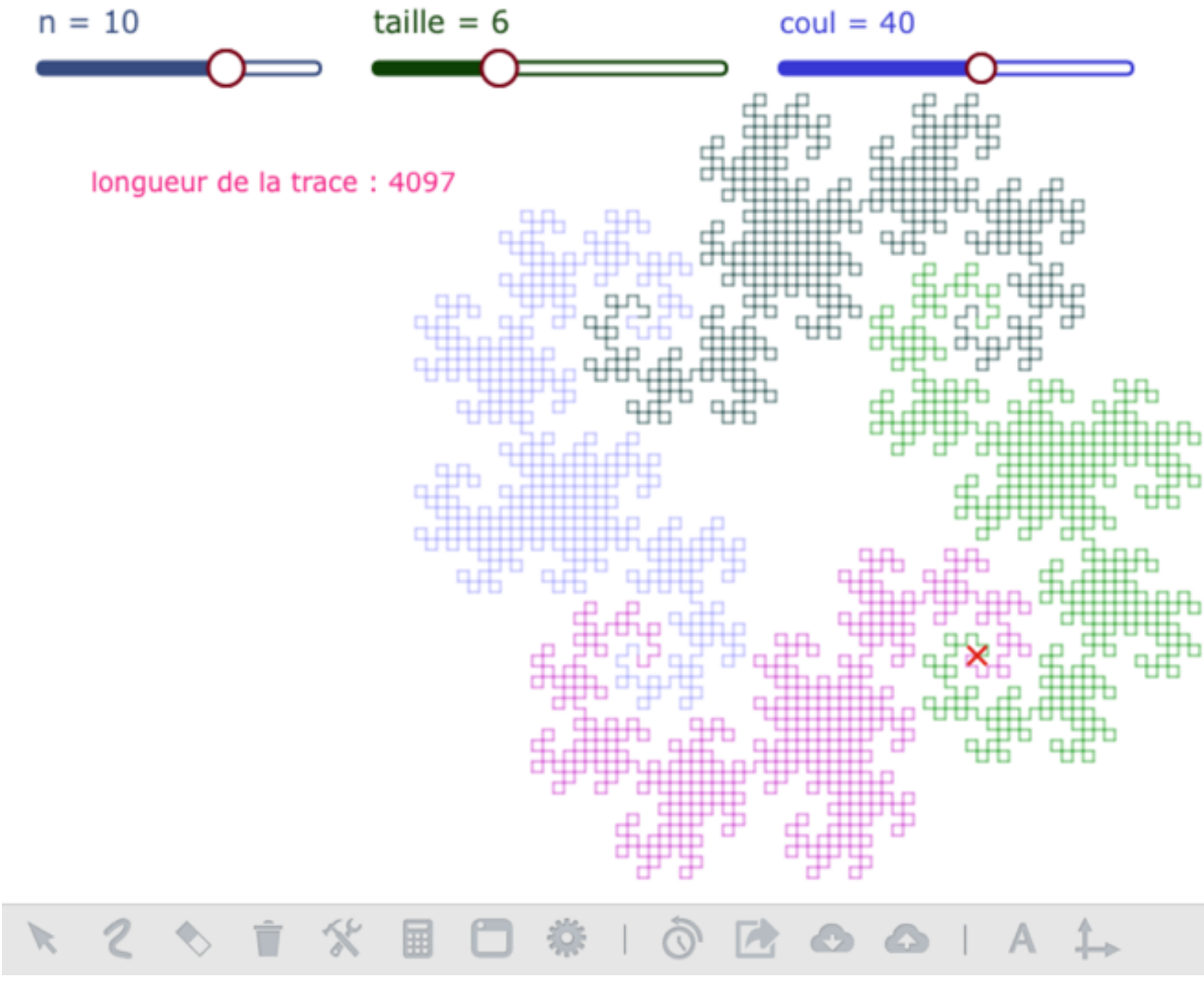
Simplement en glissant la procédure récursive dans le compteur, on obtient une autre figure obtenue par 4 dragons. On notera les 130 000 éléments ci-dessous.





Dans la page suivante, cette figure, coloriée.

Figure dynamique 9.7 – Quatre dragons – pas en rotation comme prévu initialement, mais bout à bout



*En **consultation** à l'ouverture. Le nombre de segments est* $1 + 4 \times 2^n$ *(+1 à cause du point de départ).*

Dragon - L'itération de 4 rotations

Simplement... on ajoute la rotation qui manquait au code précédent.

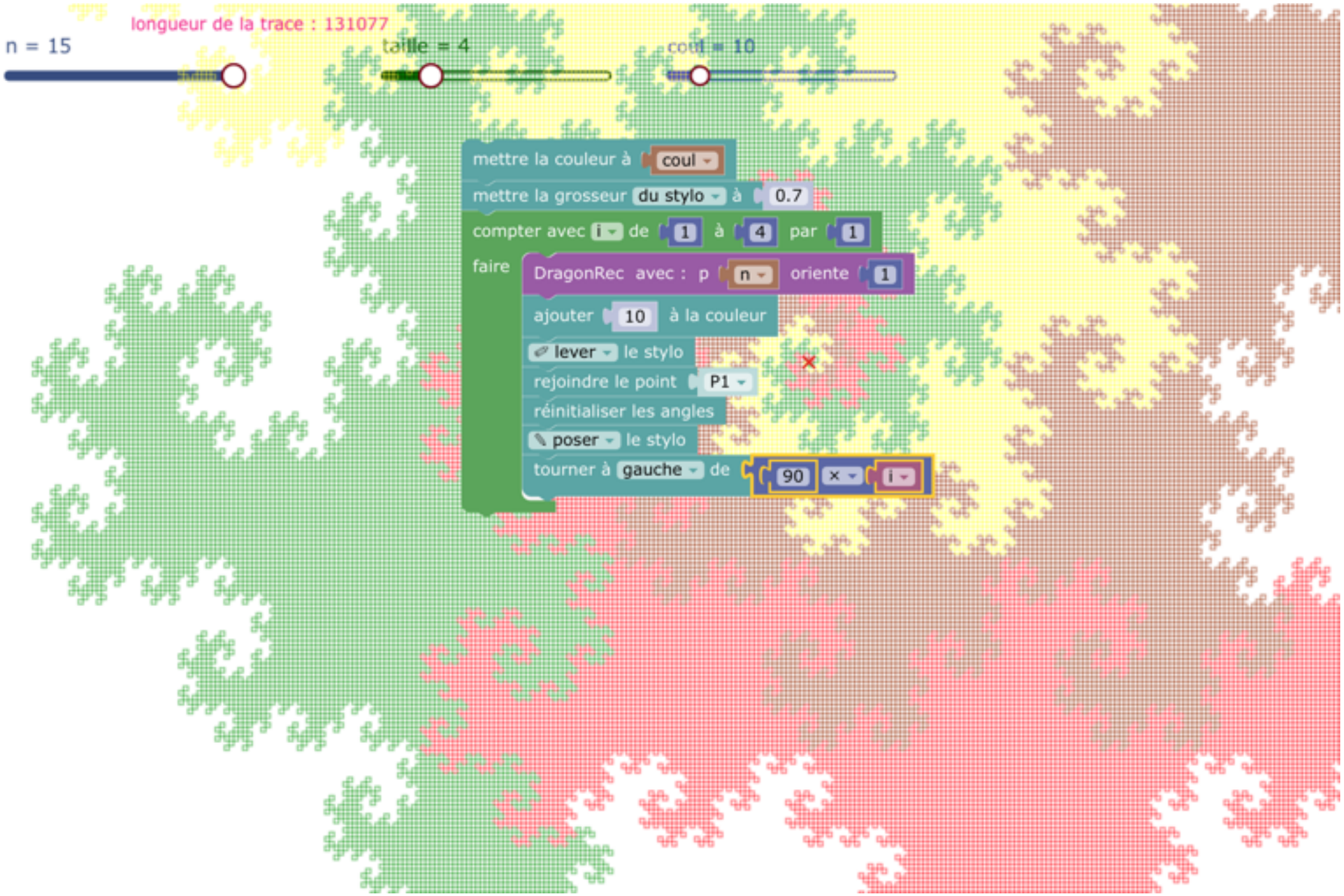
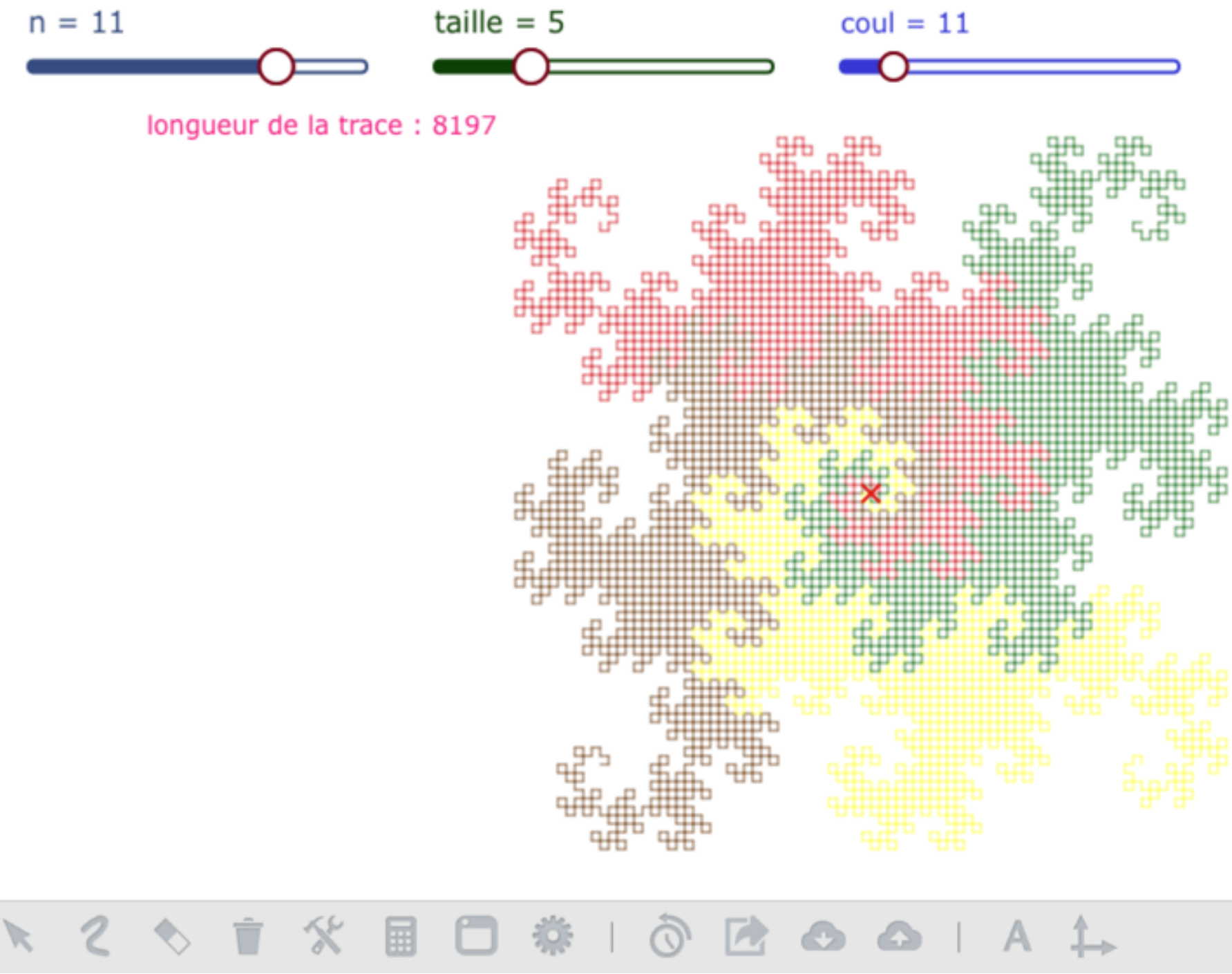


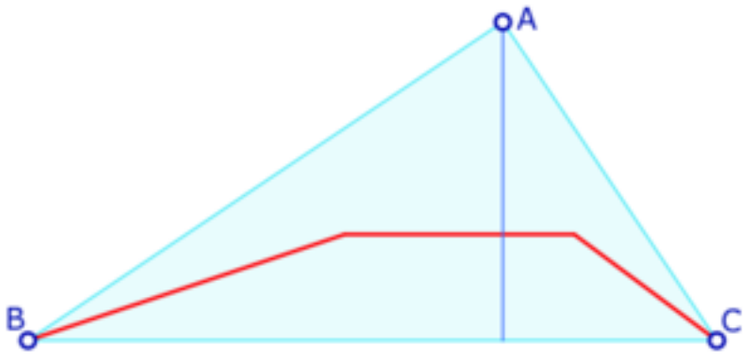
Figure dynamique 9.8 – Pavage du plan par quatre dragons en rotation autour d'un point d'origine

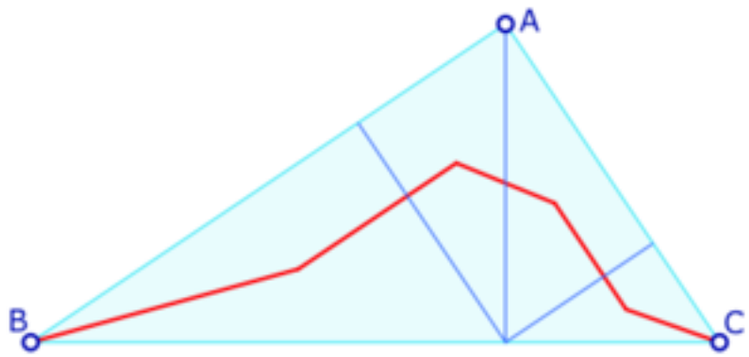


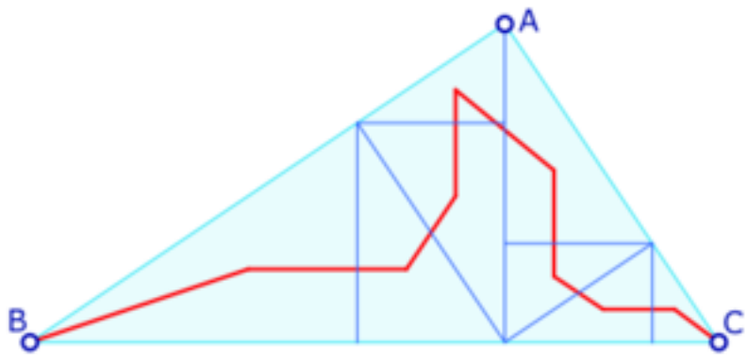
*En mode **consultation**. Activer le mode **standard** pour modifier et jouer avec le code.*

Courbe de Polya

La courbe de Polya (mathcurve) est un autre bel exemple de ce que l'on peut réaliser avec une tortue dynamique, et en particulier, par l'approche dynamique, observer des exemples de régularité intéressants. On part d'un triangle ABC rectangle en A. On le partage en deux triangles semblables à l'initial en prenant la hauteur. La première étape de la courbe va de B à C en passant par les deux centres de gravité des deux triangles. Puis on itère le procédé comme ci-dessous. La courbe de Polya est la courbe limite. Elle a été étudiée par Polya car c'est une courbe remplissante qui, dans le cas d'un triangle rectangle non isocèle, a des propriétés différentes en particulier de celle de Peano.



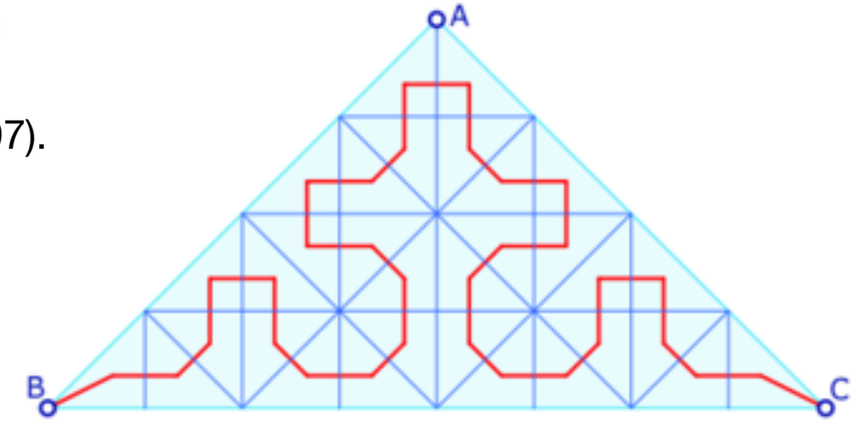


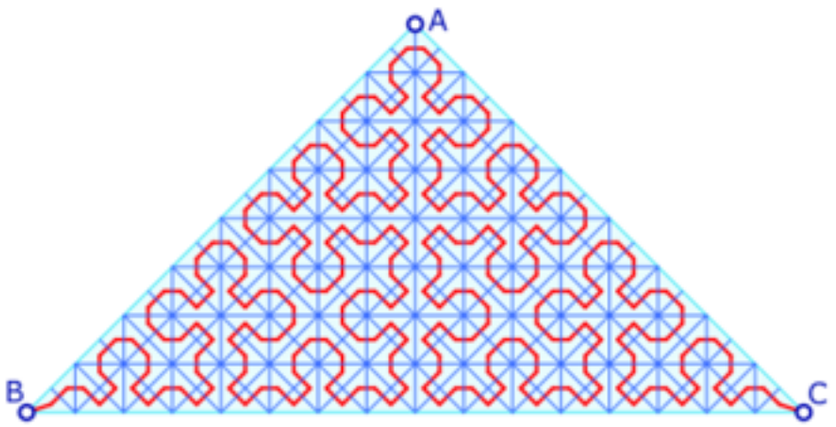


Dans le cas d'un triangle rectangle isocèle, la courbe est un cas particulier de la courbe de Césaro. En dehors de ce cas, contrairement à celle de Césaro où la courbe limite n'est dérivable en aucun point, celle de Polya est dérivable presque partout pour un angle en B compris entre π/12 et π/6, et même avec une dérivée presque partout nulle par un angle inférieur à π/12.

Voir ce passionnant article de Jean Pierre Kahane dans le Repère IREM 29 (octobre 1997).

Article plus récent (2016) de Serge Cantat dans Image des mathématiques.

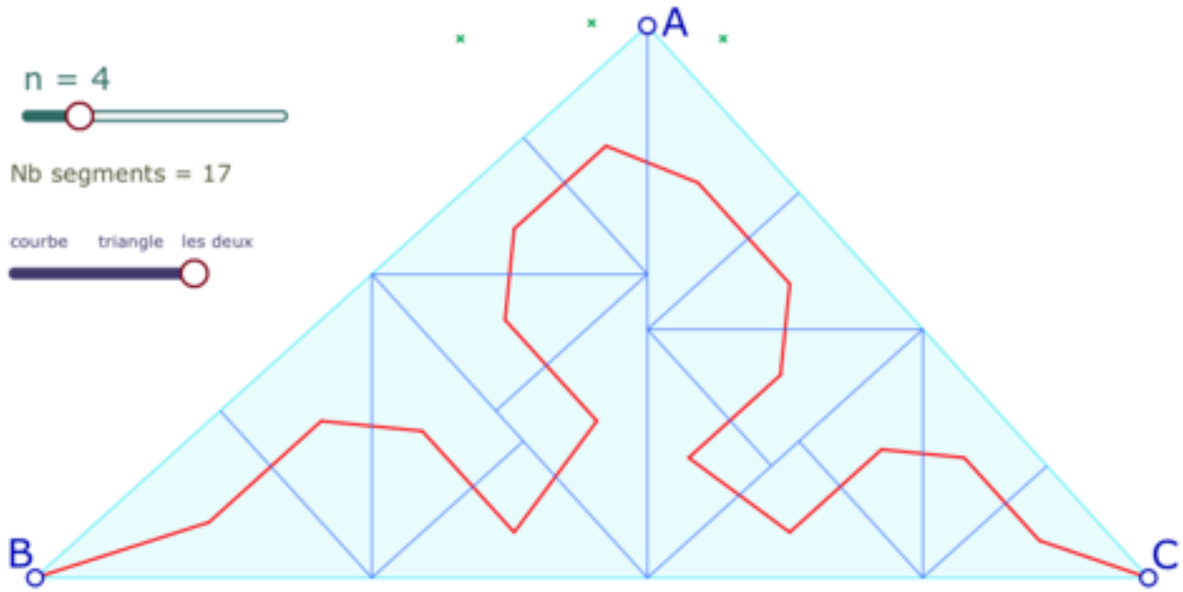




Construction de la courbe de Polya

Pour illustrer la construction on montre une version finale, avec les triangles. Dans cette figure, le code est en A, et donc la phase d'initialisation met la tortue en B.

On commence par faire une fonction qui construit la projection orthogonale de l'angle droit. Celle-ci étant disponible, le code de la courbe est élémentaire : la construction de la courbe revient à rejoindre le centre de gravité d'un triangle en cours.




```
pour Projection avec : P, Q, R
    fixer longH à  distance P Q  ×  cos  angle 180  P Q R
    fixer ptH à  Q  +  ( R  -  Q )  ×  longH  ÷  distance Q R
    retour ptH
```

```
pour Polya avec : I, J, K, prof
    si  prof  =  0
    faire  rejoindre le point  ( I  +  J  +  K )  ÷  3
    sinon  fixer H à  Projection avec : P I Q J R K
           Polya avec : I H J J K I prof  prof  -  1
           Polya avec : I H J I K K prof  prof  -  1
```

```
Init
    Polya avec : I A J B K C prof n
    rejoindre le point C
```

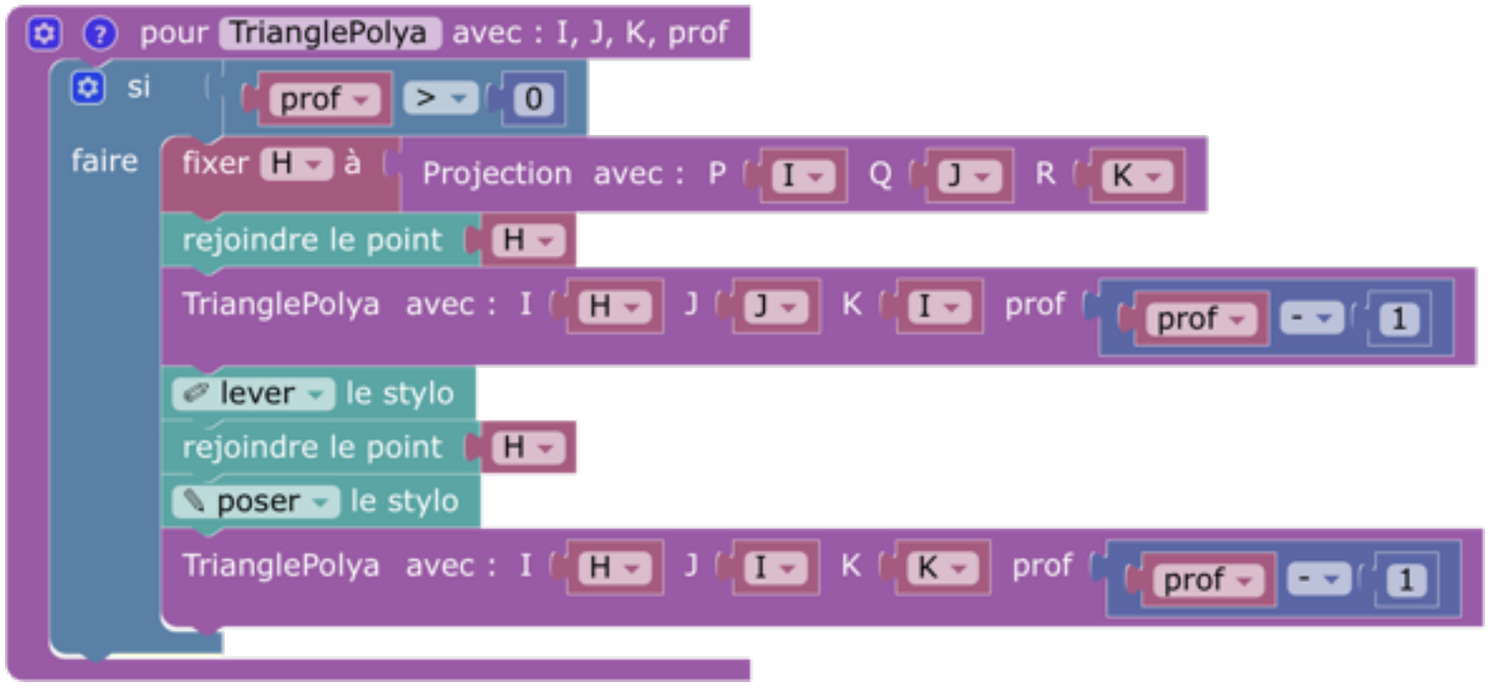
Procédures principale et d'initialisation

```
pour Init
    lever le stylo
    rejoindre le point B
    poser le stylo
    mettre la grosseur du stylo à 2
    mettre la couleur à 10
```

La construction des triangles

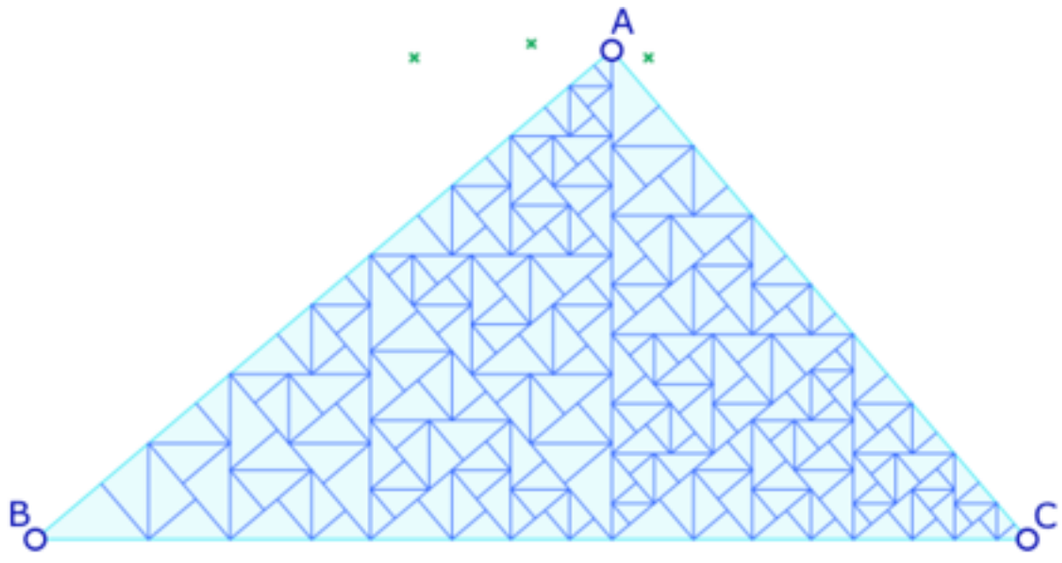
La tortue est initialement en A (l'angle droit). Si la profondeur n'est pas nulle, on calcule le pied de la hauteur. On rejoint ce point, puis on lance la récursivité sur la première branche (un des deux triangles), puis on revient en H et on lance à nouveau la récursivité sur l'autre branche (le second triangle).

Dans la figure finale, on a ajouté un curseur, ce qui ralenti un peu l'affichage.



Alignements particuliers - Le nombre d'or dans la courbe de Polya

Quand on déplace le point A, on note qu'il y a des cas particuliers d'alignements des sommets des triangles (à droite A est sur l'un des trois points verts). Ces alignements sont réalisés, par exemple, quand l'angle en B vérifie $\cos\theta = \tan\theta$ ce qui correspond à deux angles complémentaires dont l'un est $\sin^{-1}(\varphi - 1)$ où φ est le nombre d'or.



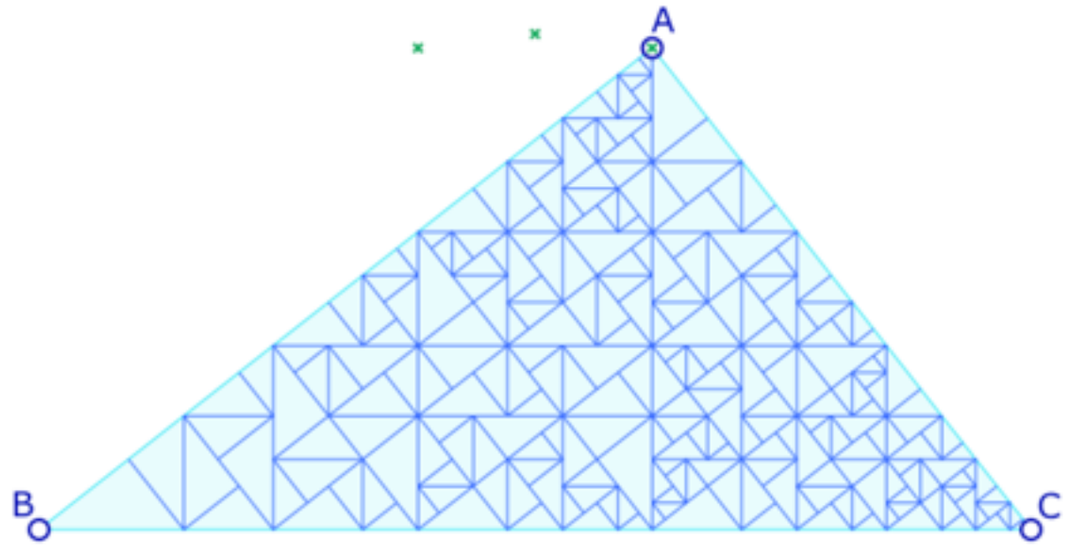
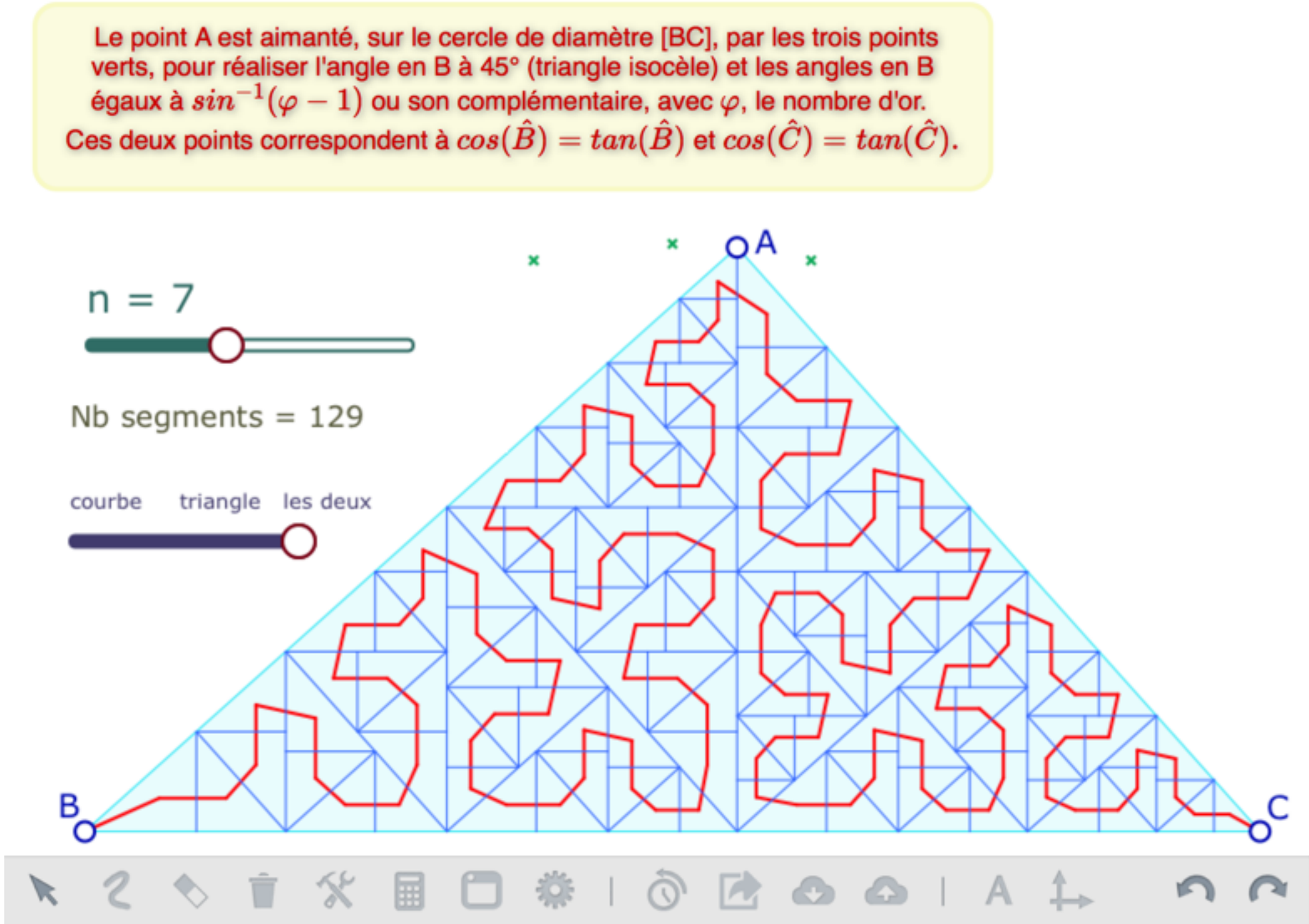


Figure dynamique 9.9 – Courbe de Polya et découpage d'un triangle rectangle en triangles semblables



On peut agir sur A, choisir la profondeur n, et choisir le type d'affichage.

Arbres 2D et 3D

Ce paragraphe sur les arbres est né de la pratique des arbres récursifs 2D proposé par le site de GéoTortue ou encore sur le site de l'IREM de Paris 13. Voici une version de base, de type scolaire. La seule règle à respecter est que **la somme des angles à gauche et à droite soit équilibrée**. Ceci étant acquis, tout est possible en termes de variantes.

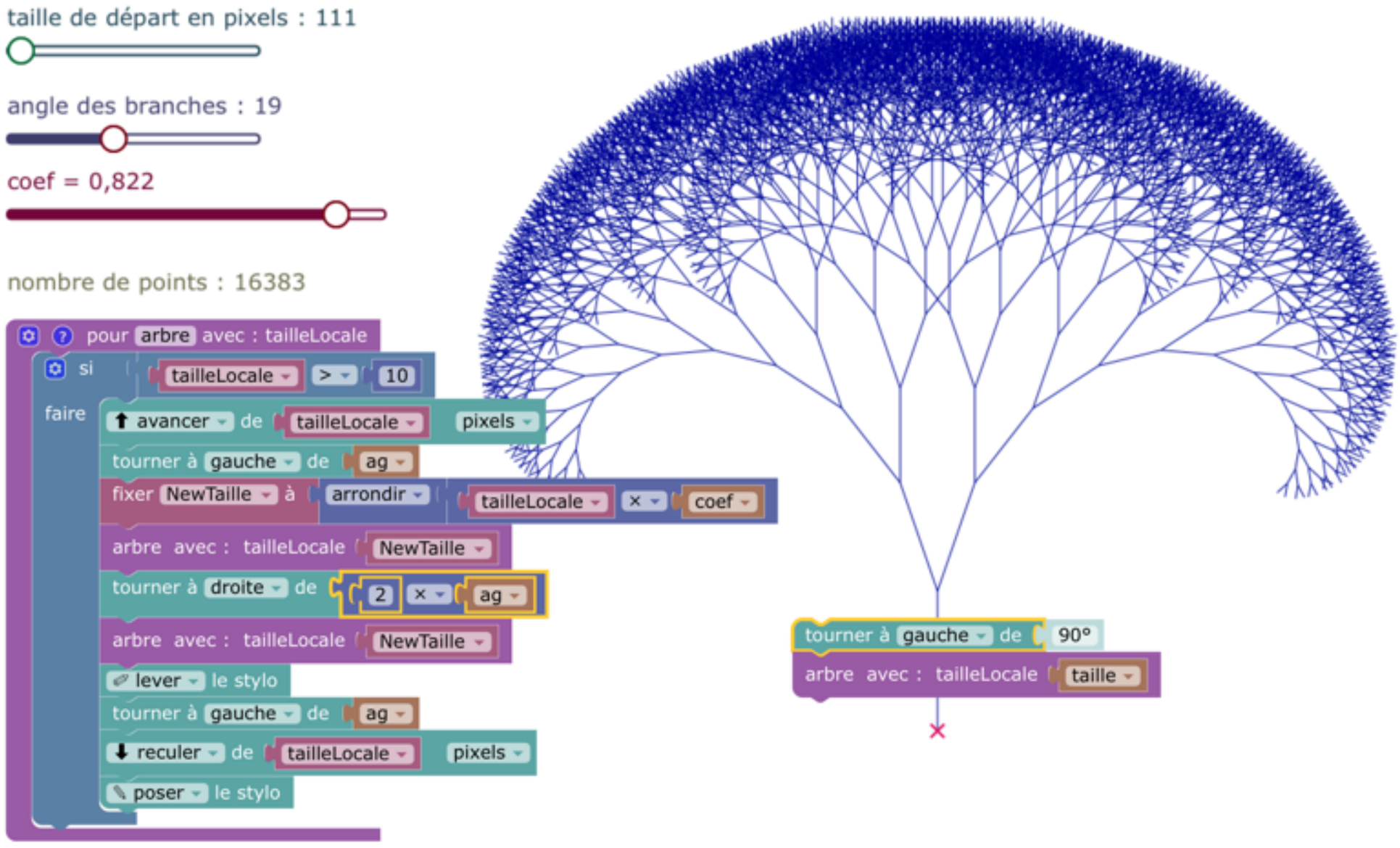
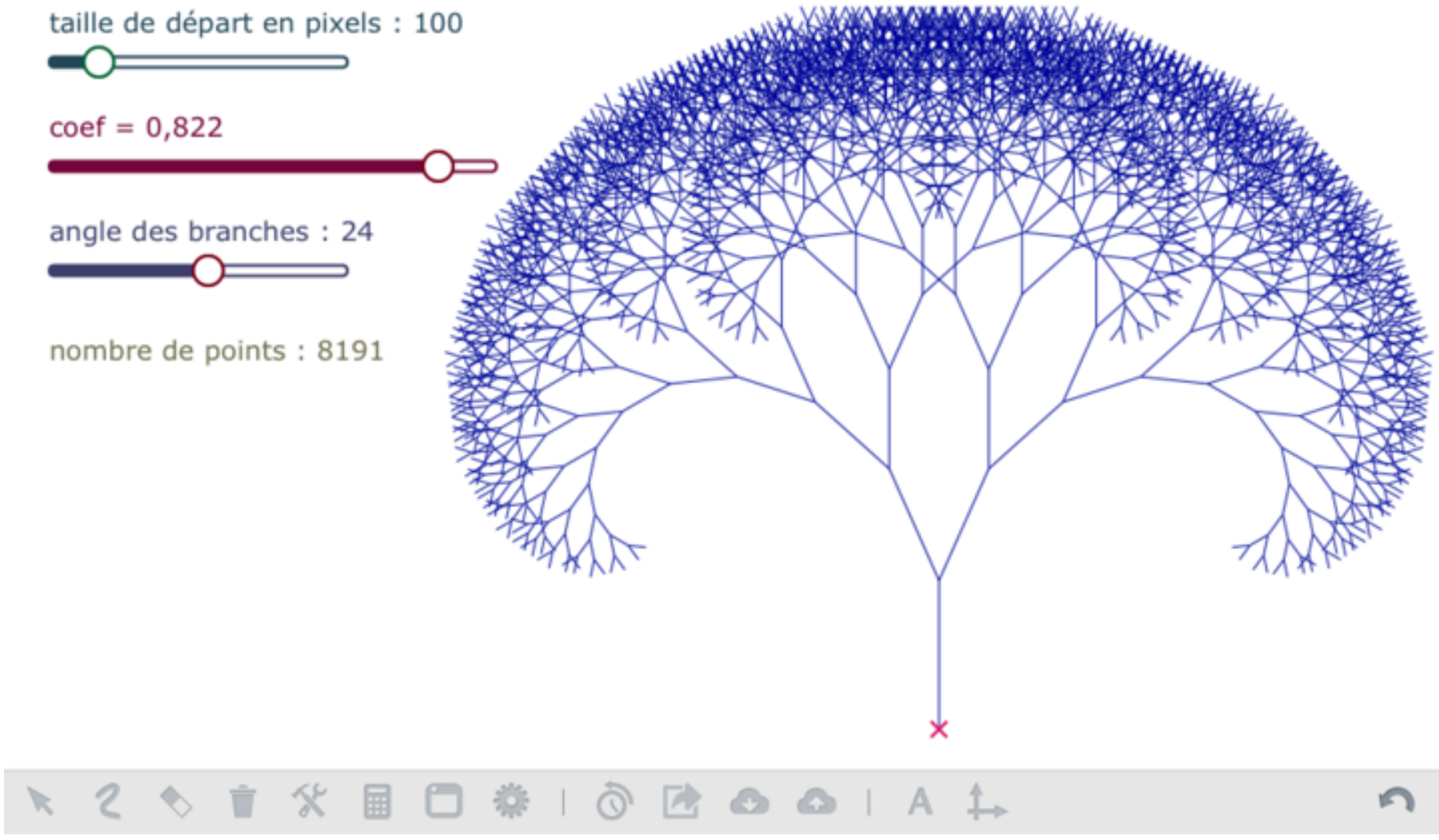
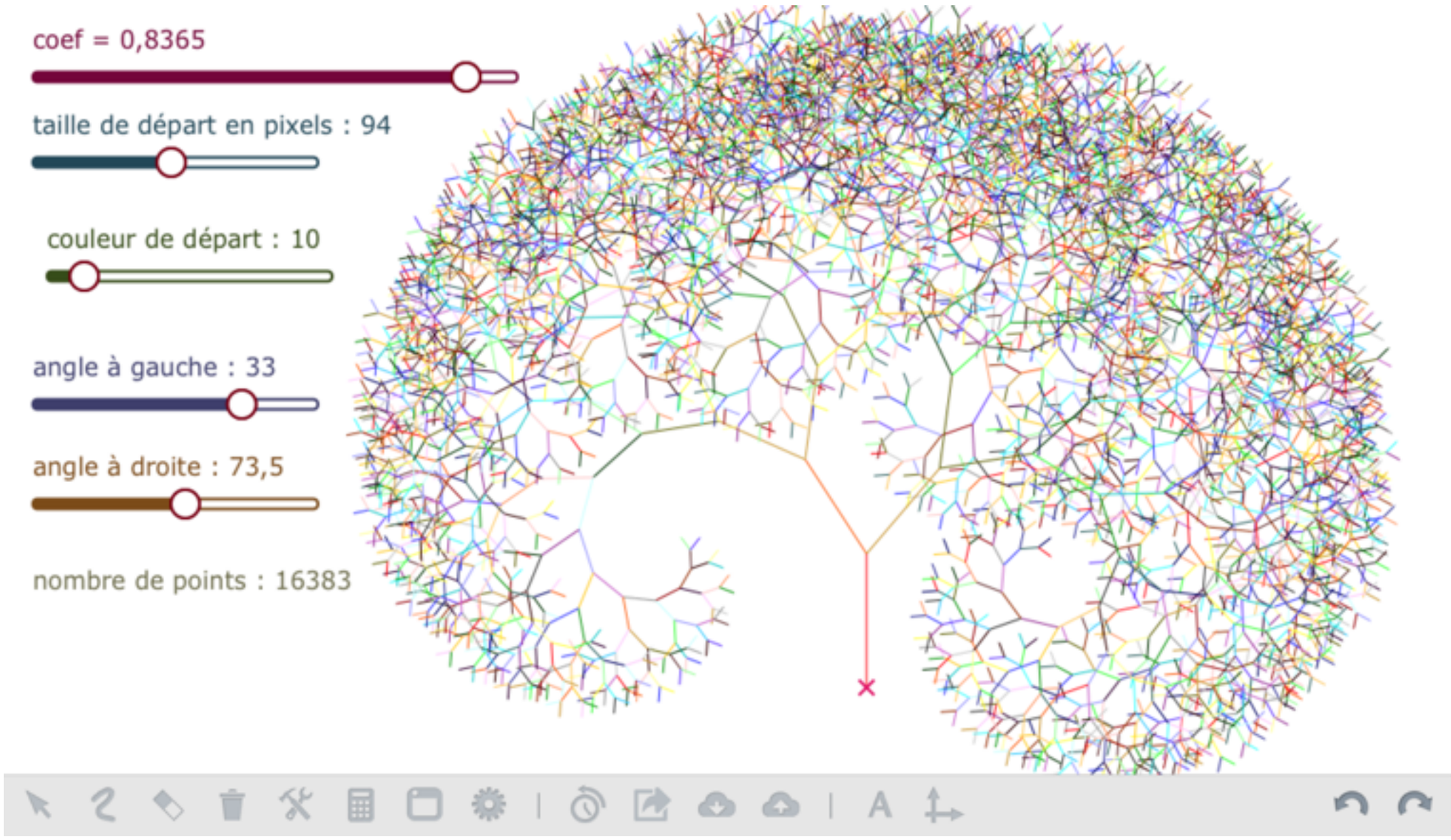


Figure dynamique 9.10 – Arbre 2D – un seul angle



*En **consultation** à l'ouverture.*

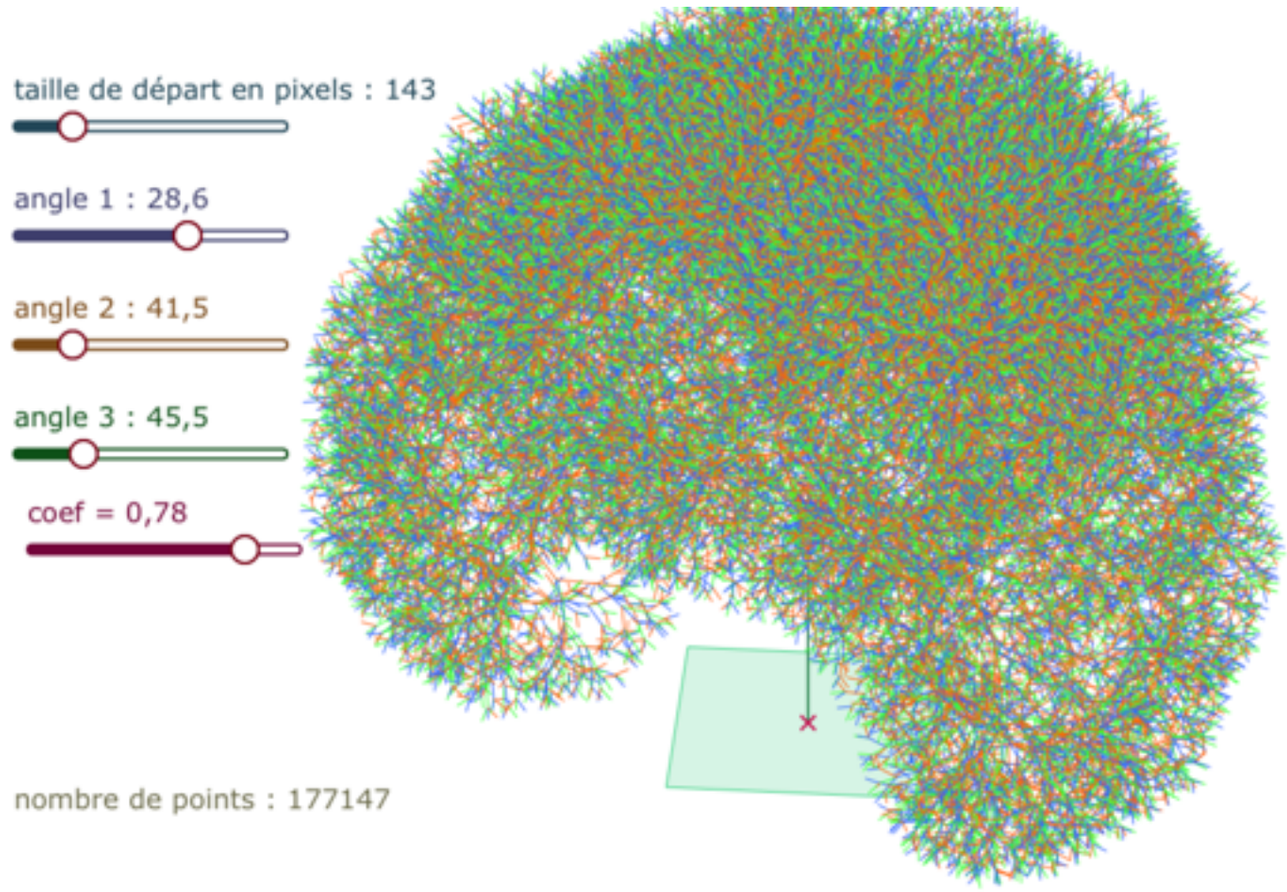
Figure dynamique 9.11 – Arbre 2D avec 2 angles et des couleurs



*Encore en mode **consultation** à l'ouverture. Le code est dans l'unique point de la figure.*

Arbre 3D avec 3 angles distincts

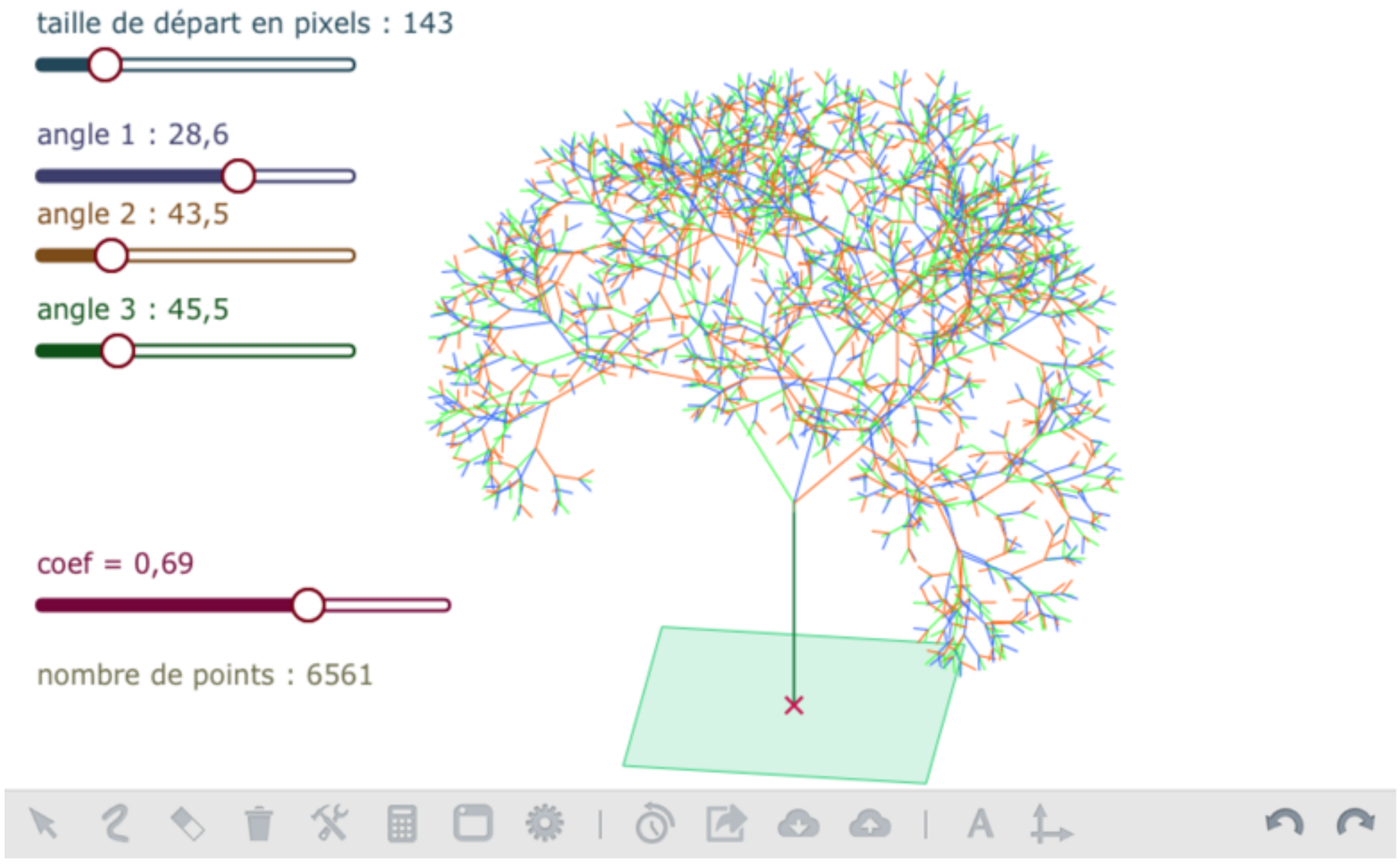
On se donne trois angles, à 120° autour de la branche racine.



Dans cette version, on voit qu'il y a une répartition équitable sur les couleurs, ce qui ne sera pas le cas dans la version suivante, qui va densifier les branches, en mettant les appels dans une autre procédure.

Dans la figure suivante, **ne pas dépasser 0,78** pour le coefficient.

Figure dynamique 9.12 – Arbre 3D avec trois angles, un par axe.

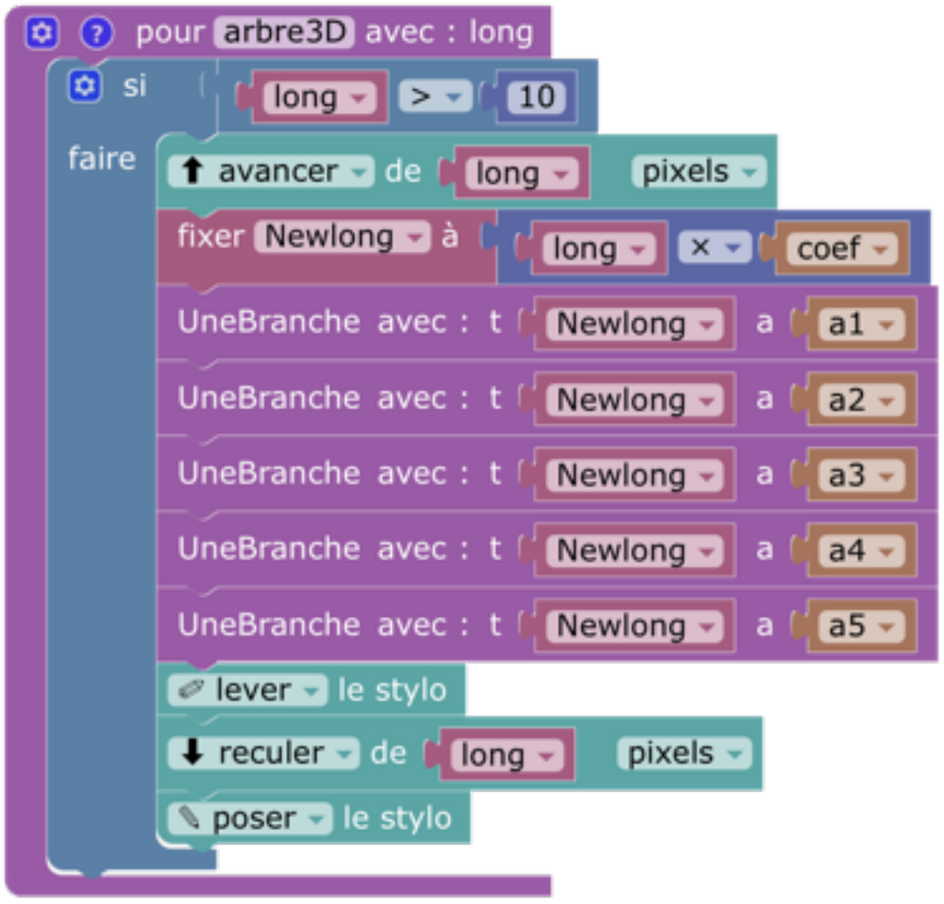


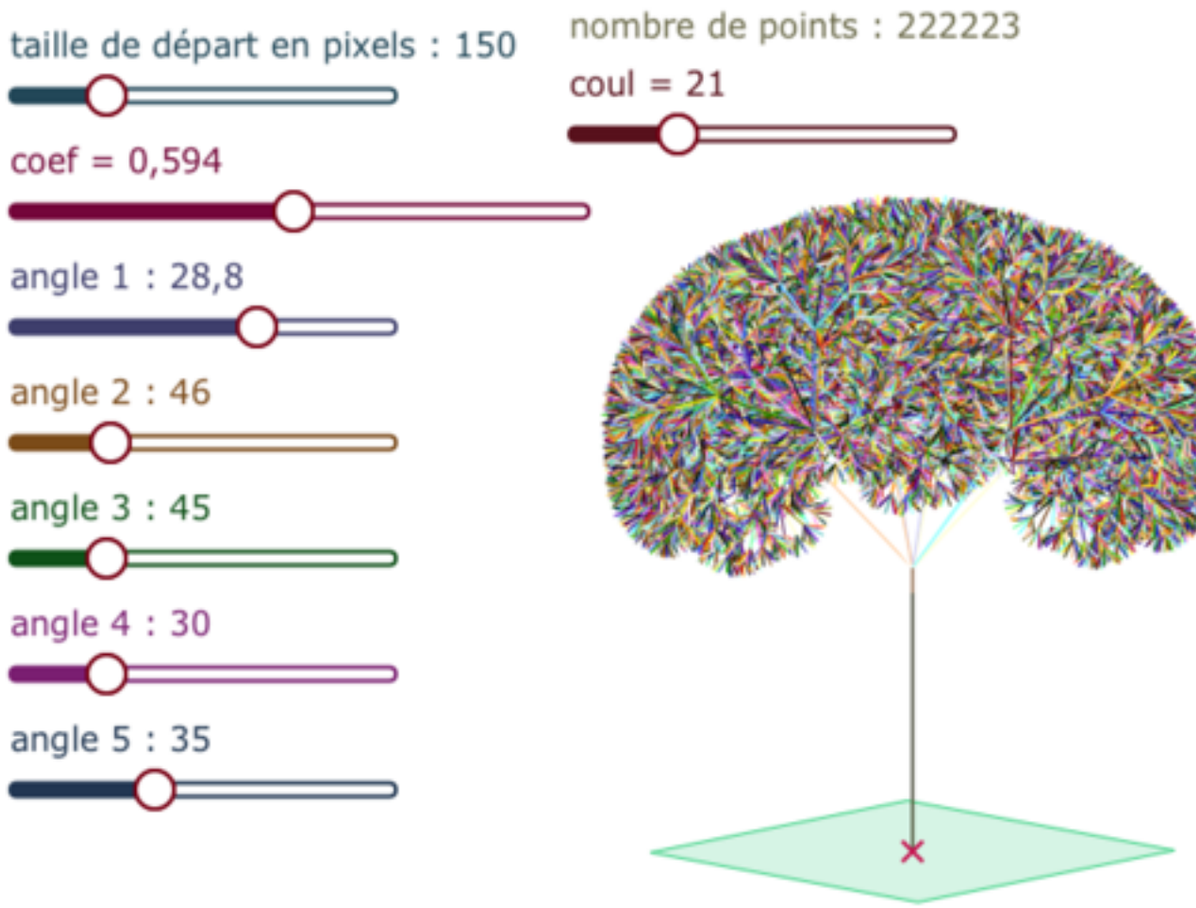
*En **consultation** à l'ouverture. Attention à la sensibilité au coefficient : maximum 0,78.*

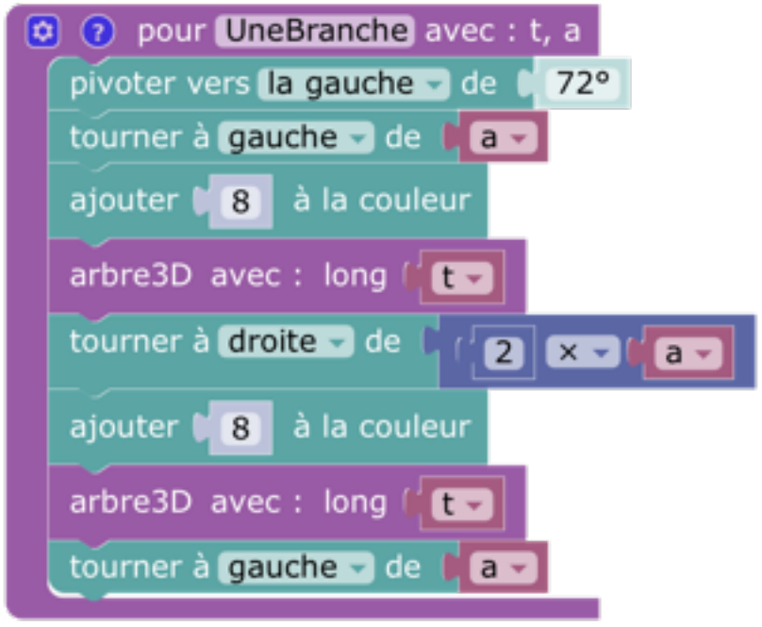
Arbre 3D - 5 branches par arbre et deux arbres par branche...

Dans cette version, on a 5 branches, et on appelle 2 fois l'arbre par branche, donc l'arbre est largement plus densifié.

Par contre, la gestion comme dans la figure précédente, placée dans la procédure **UneBranche** est inopérante : la couleur du dernier arbre est prédominante. D'où cette version d'ajout d'un indice de couleur.



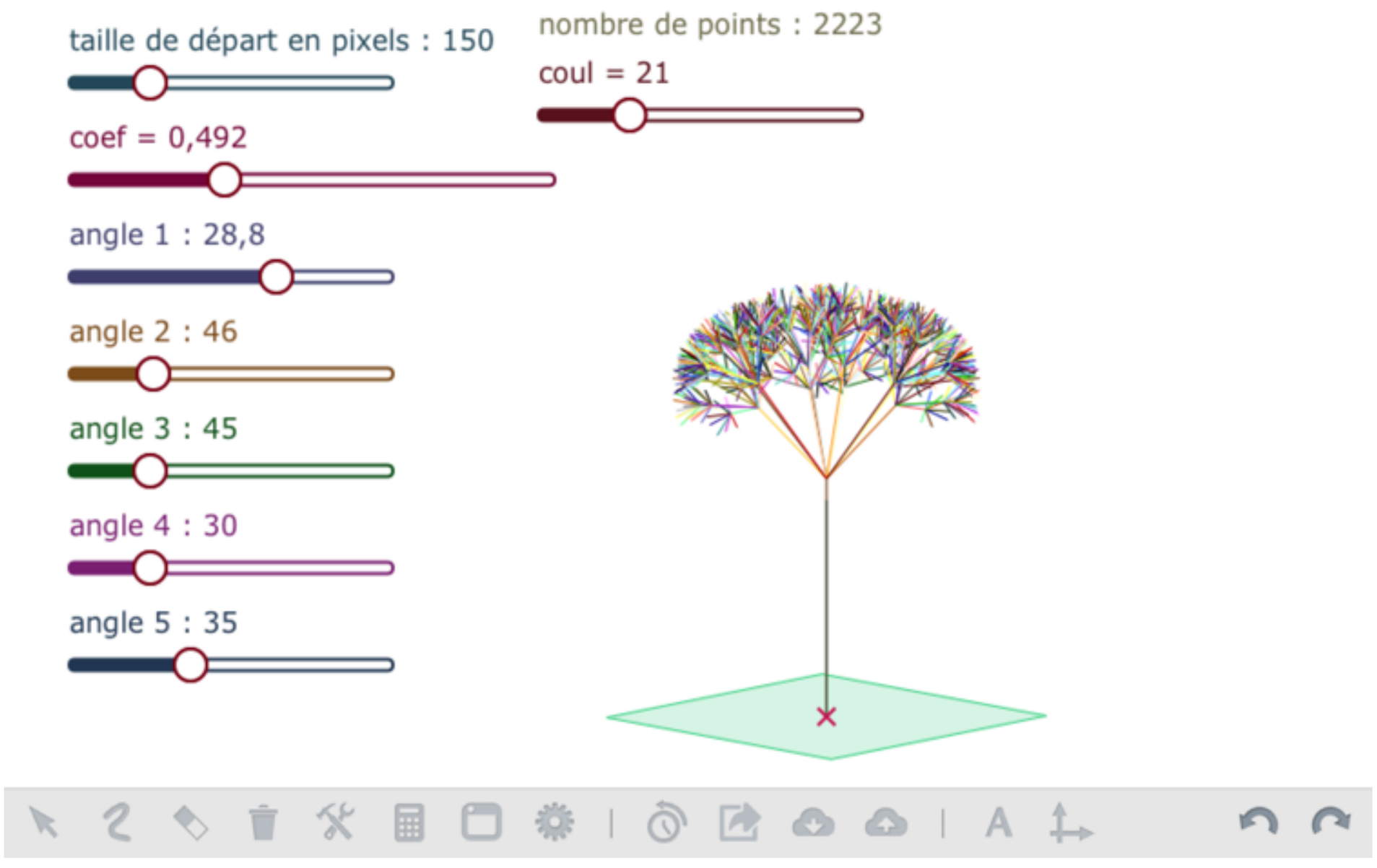




Attention

Bien remarquer qu'il y a déjà **222 000 objets en 3D** pour un coefficient à 0,59.
Ne pas essayer de reproduire sur tablette.

Figure dynamique 9.13 – Arbre 3D avec 5 branches par arbre et 2 arbres par branches

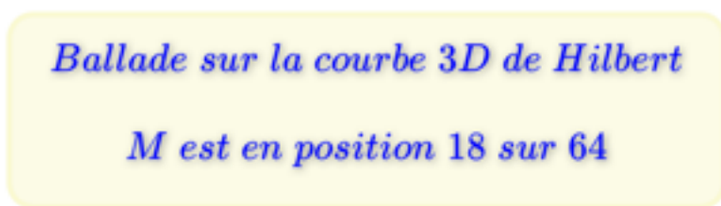


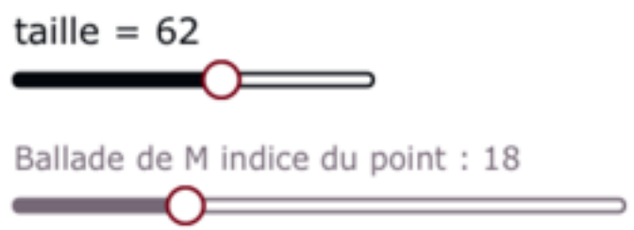
Attention au coefficient : nombres d'objets 223 - 2223 - 22223. Eviter sur tablette le suivant 222223 !

Hilbert 3D et Penrose à la tortue

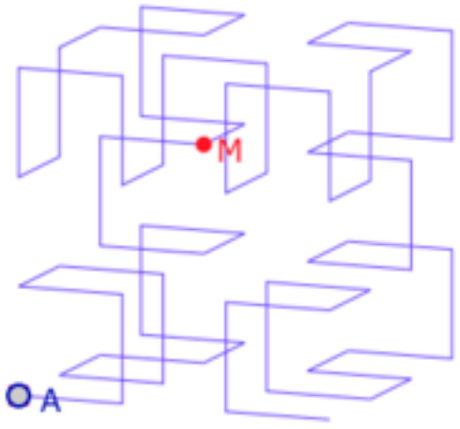
Dans cette partie, deux figures extraordinaires de **Patrice Debrabant**, l'auteur du site carmetal.org qui donne les informations récentes sur les logiciels CaRMetal et DGPad. Patrice a écrit plusieurs articles dans *MathémaTICE*, seul ou souvent avec Alain Busser. Pour l'esthétique, voir par exemple son superbe article sur les fleurs de tractoires, ou ses très belles animations des systèmes différentiels (en collaboration) ou encore les récents (septembre 2017) spirolatères différentiels et tortue réaliste tortue gaussienne avec Alain Busser.

Dans la figure suivante, la courbe 3D de Hilbert est réalisée à partir d'une seule procédure récursive (elle est souvent faite avec 2 ou 4 procédures). Et de plus, les segments sont construits à la suite les uns des autres.





Pour le vérifier un compteur permet (pour n=2 et 3) de faire parcourir la courbe par le point M à l'unité prés (il peut falloir agrandir un peu le curseur pour n=3 car il y a 512 segments). Pour n=4 on n'a pas l'avancée à l'unité car il y a trop de segment mais on peut faire une animation sur le curseur de la ballade.



Cette figure est aussi l'occasion de voir comment **faire parcourir à un point une trace** de tortue. On prend un point et on lui attribue un **comportement d'expression** de la façon ci-contre.

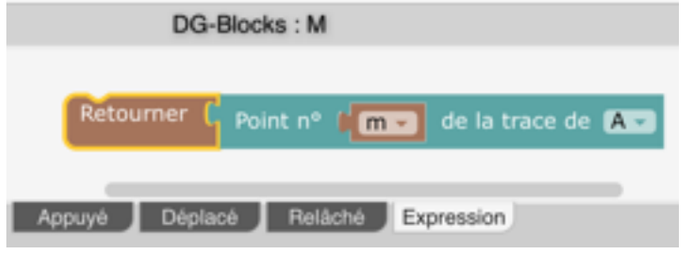
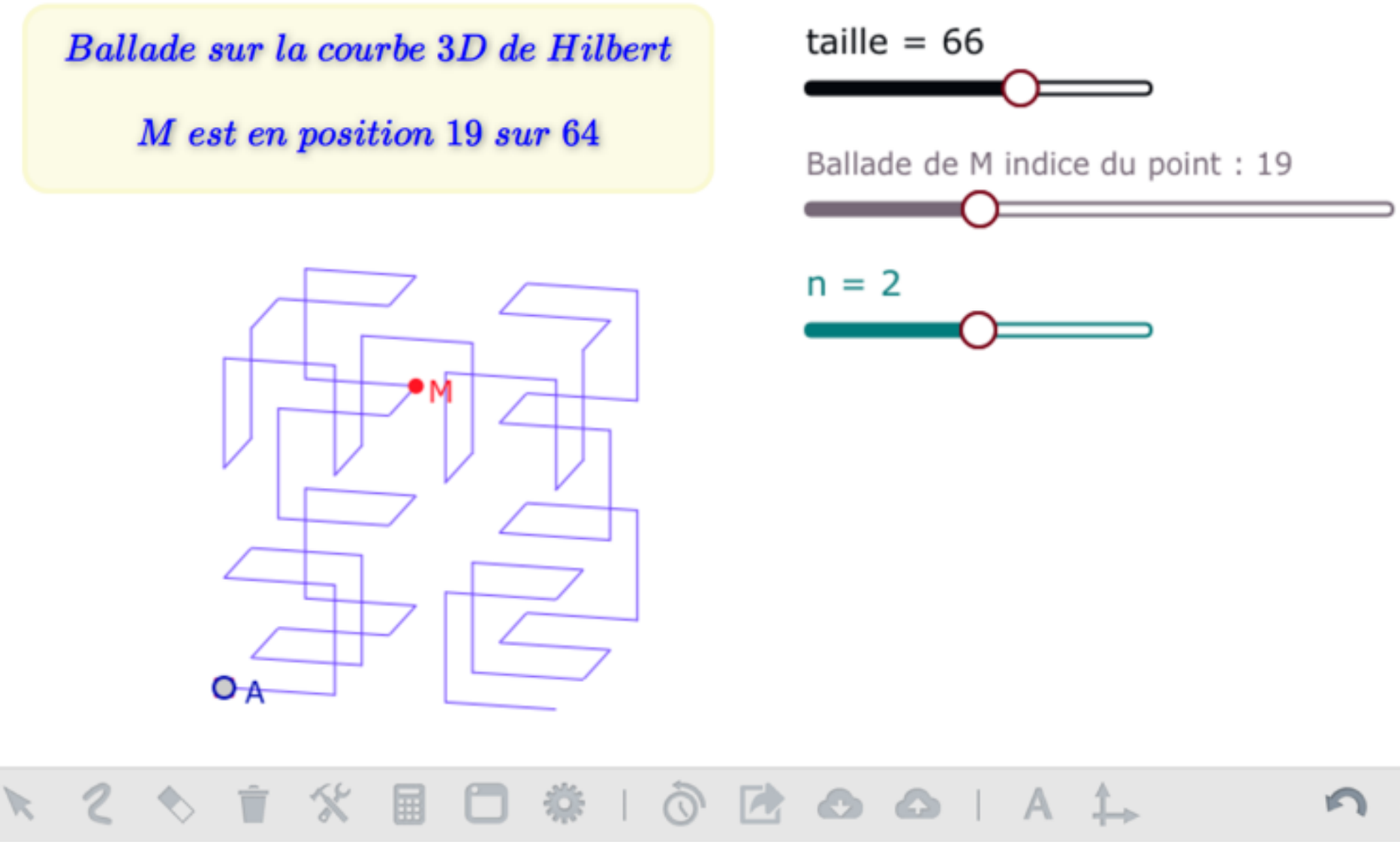


Figure dynamique 9.14 – Ballade sur Hilbert 3D (Patrice Debrabant)



*En mode **consultation** à l'ouverture. Le code est en A (qui est construit sur O l'origine du repère).*

Pavage de Penrose

Deux procédures, **Chevron** et **Flèche**, sont définies l'une par rapport à l'autre (le code est dans les deux points rouges). Le code est le même pour les deux figures, mais dans l'une, l'appel principal est la flèche, dans l'autre, le chevron.

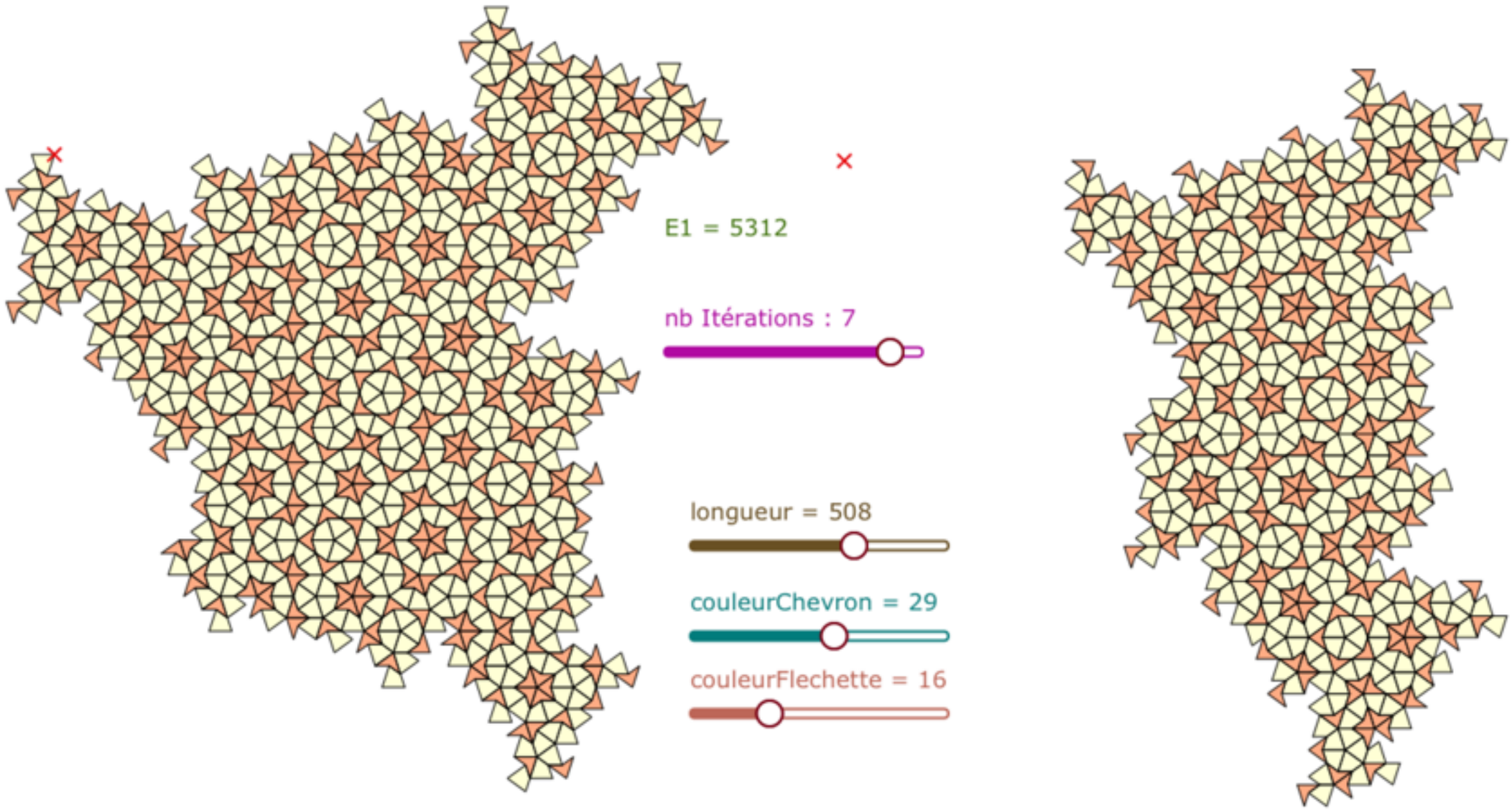
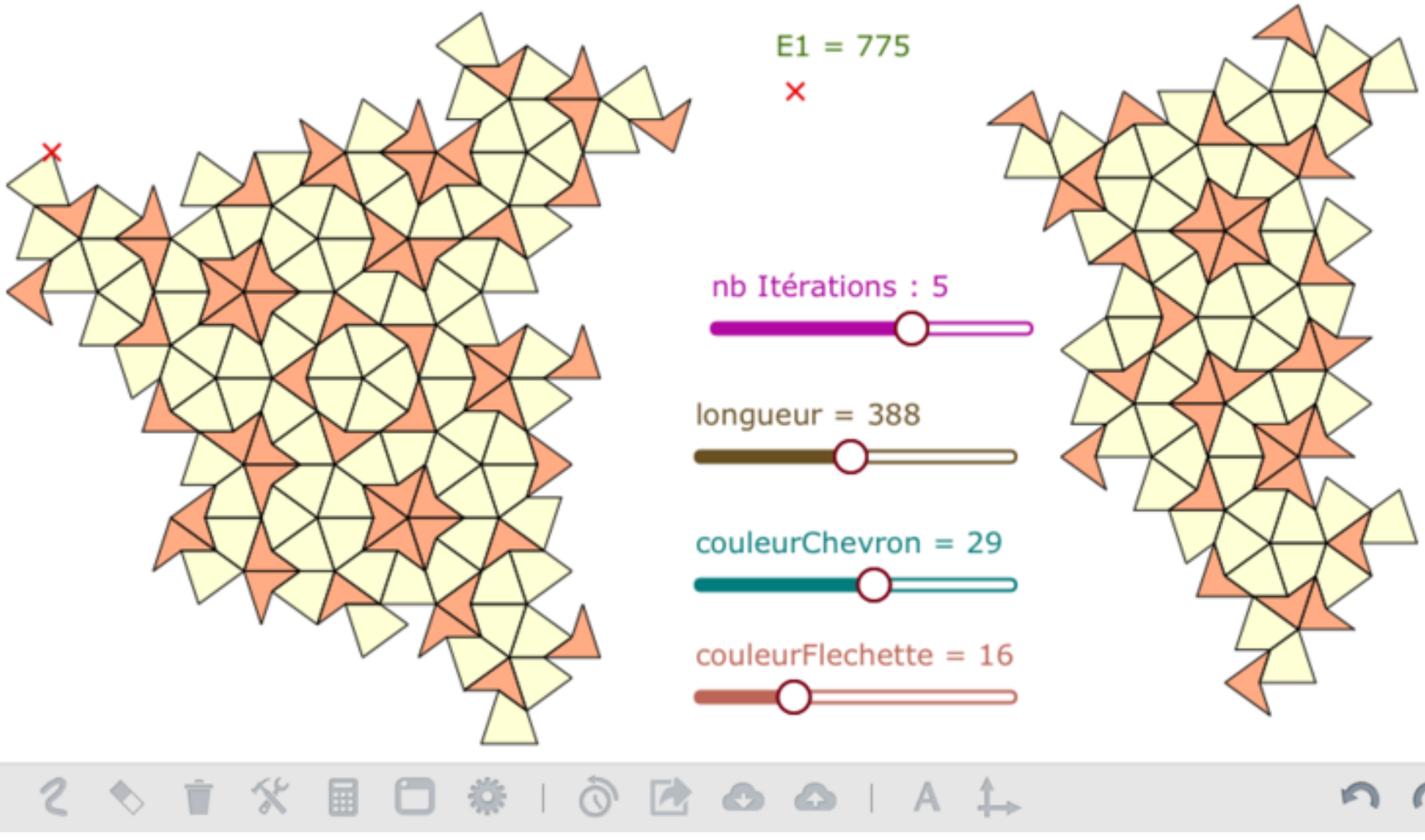


Figure dynamique 9.15 – Flèche et chevron de Penrose (Patrice Debrabant)



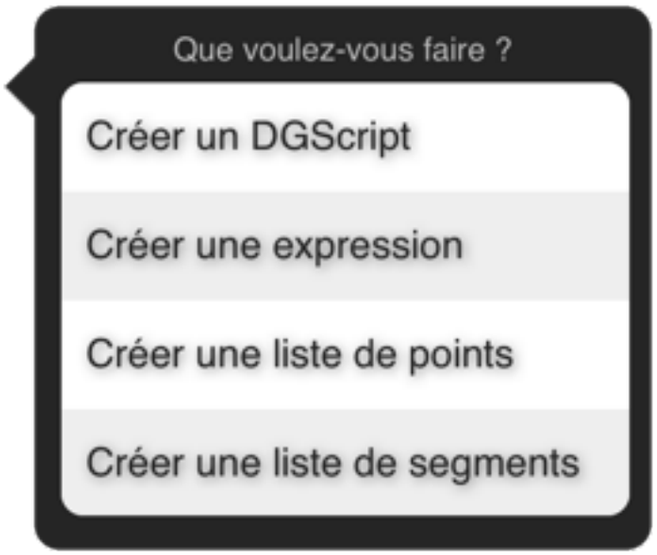
Le code est dans les deux points rouge. Mettre la profondeur à 0 et 1 pour voir le principe de la figure.

Blockly et les listes de DGPad

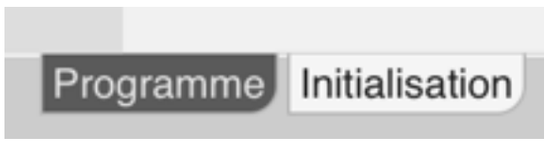
La tortue de DGPad n'est qu'une catégorie des outils de Blockly, on peut faire de la programmation par bloc dans DGPad sans utiliser la tortue, sur les points bien entendu, mais aussi sur les listes. L'intérêt esthétique sur les listes est dans la possibilité d'utiliser le **RGBColor,** non disponible avec la tortue. L'intérêt sur les points est dans l'onglet **Expression**, déjà utilisé avec la trace de la tortue, mais aussi dans les onglets **Relâché** et ***Appuyé***, déjà illustrés au chapitre 2.

Le prochain chapitre sera consacré à l'utilisation de Blockly sur les points. Ici on regarde les possibilités plus générales sur les listes ou la récursivité. On ne détaillera pas sur les DGscripts qui peuvent avoir une partie initialisation. Voir les figures de Eric Hakenholz dans le chapitre 5 pour les DGScripts.

Un touché long sur l'écran ouvre ce menu



Créer un **DGScript** renvoie à cette interface, avec un onglet **programme** et un onglet **initialisation**.



Créer une liste (de points ou de segments) crée un rectangle, soit une liste de 4 points ou 4 segments.

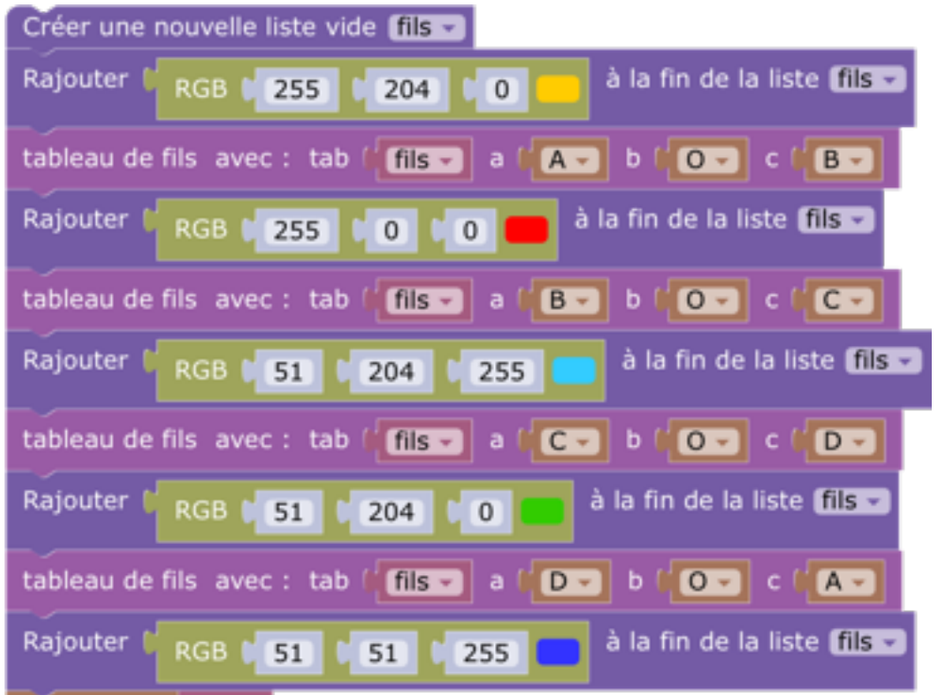
Une liste ainsi créée (ou autrement), on peut lui appliquer un comportement Blockly. Ce chapitre propose quelques exemples d'utilisation spécifique des blocs listes de Blockly.

Colorisation par RGBColor

En général, le programme principal commence par **créer une liste vide** (ici **fils**) et se termine – en général – par l'instruction **Retourner**, qui transforme la liste initiale sur laquelle on a appliqué le comportement en la liste programmée en Blockly. Ce qui est bien plus simple que l'exemple proposé au chapitre 1 sur le cœur de tournesol.

Sur cet exemple, on voit une méthode générique de remplissage standard de la liste. On verra des méthodes plus fines un peu plus loin.

Cet exemple élémentaire a été choisi pour illustrer que l'on peut **insérer des lignes de couleurs dans la liste générale**, les objets ayant alors leurs couleurs lissées d'un marqueur de couleur à l'autre, comme ci-contre.



La page suivante reprend cette figure. La suivante propose un autre exemple.

Attention : contrairement à la tortue, le code n'est pas dans les points, mais toujours dans la liste.

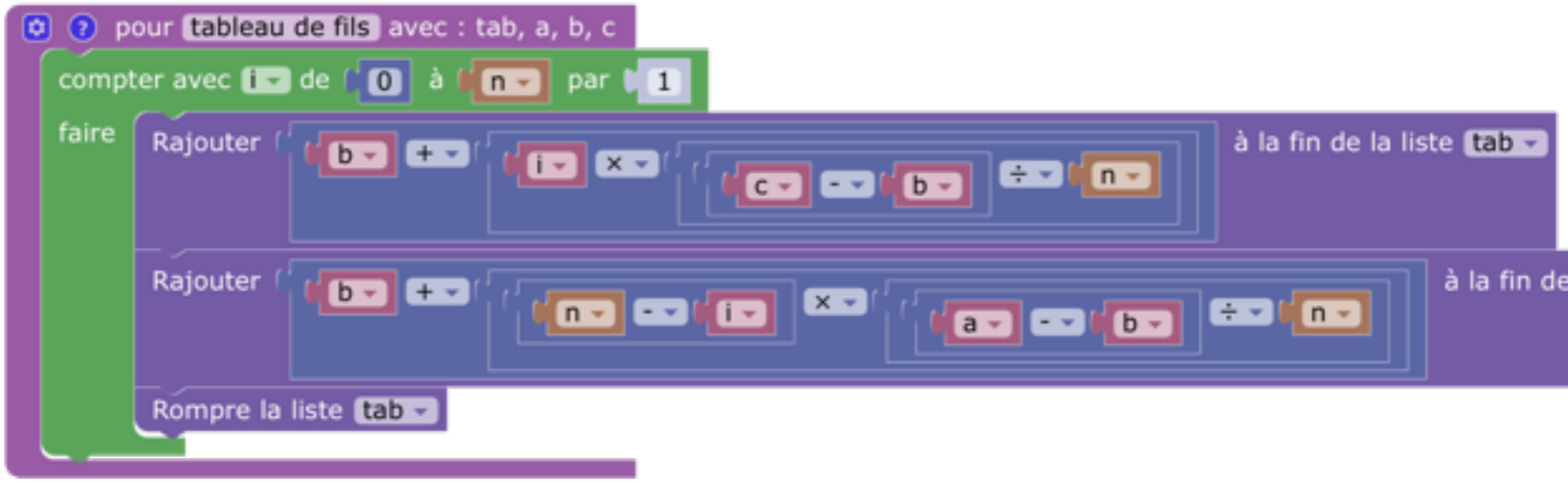
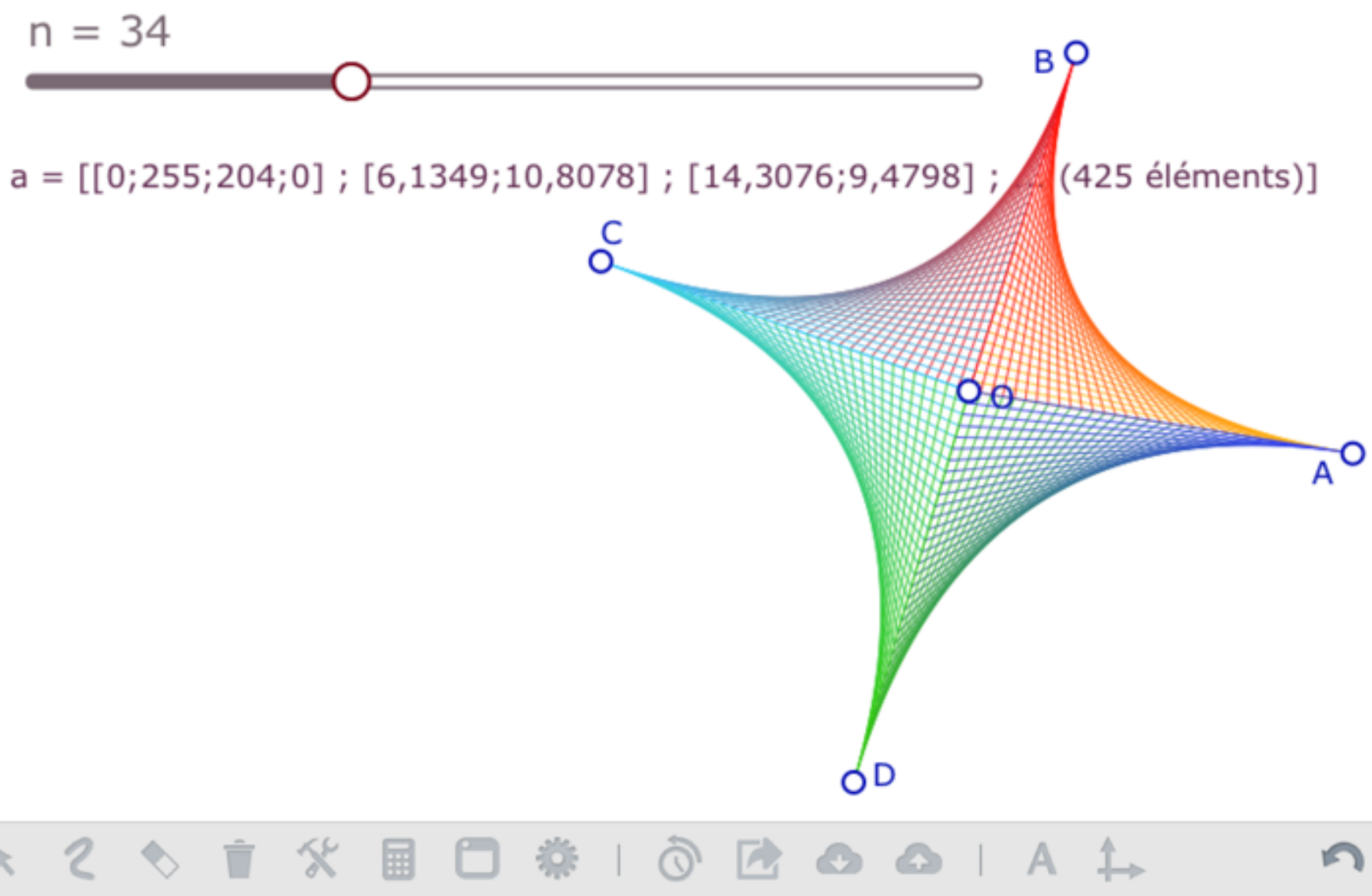
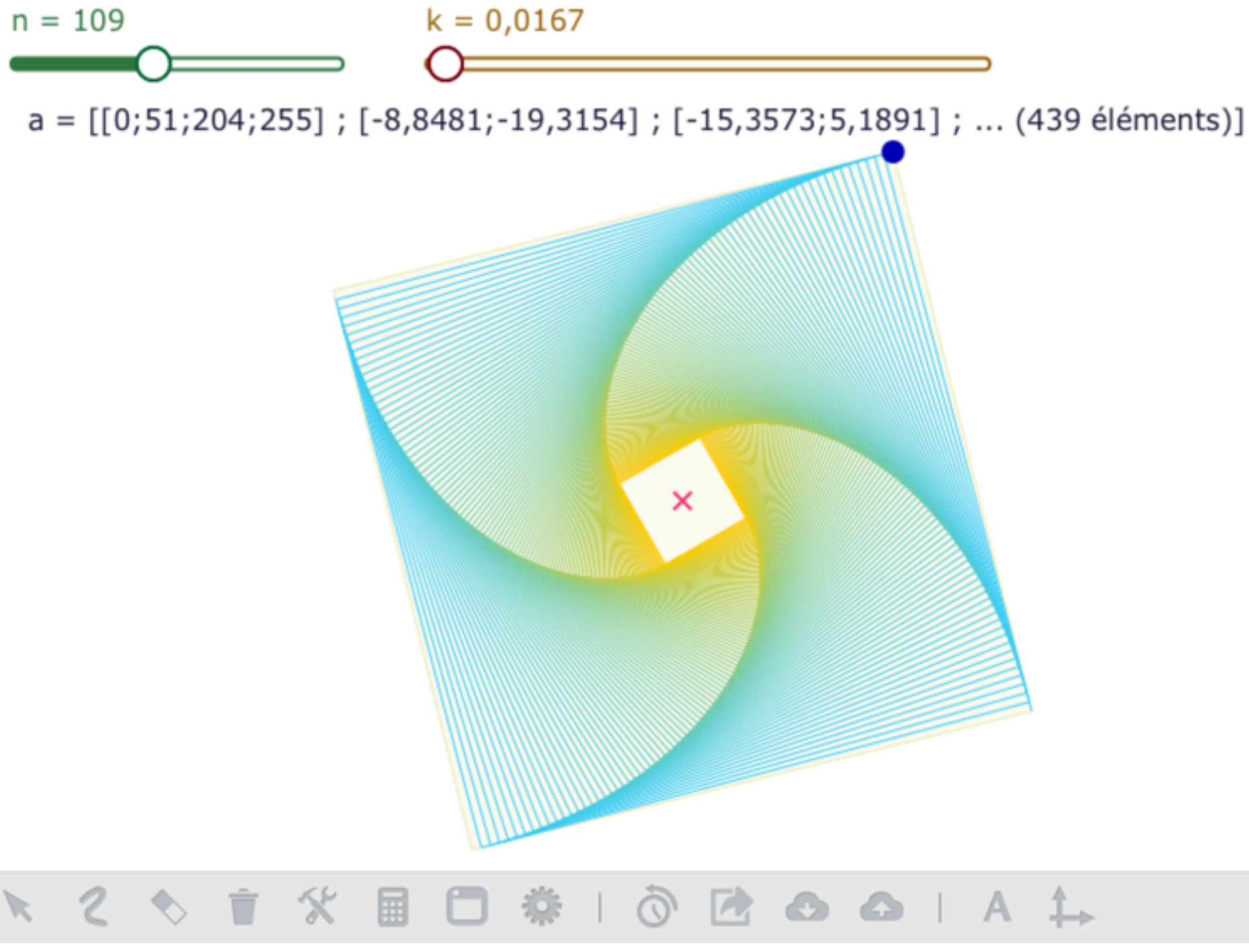


Figure dynamique 10.1 – Fils dynamiques créés et coloriés par une liste Blockly



*En **consultation** à l'ouverture. Passer en **mode standard** pour jouer à modifier les couleurs par exemple. Le code est dans la liste.*

Figure dynamique 10.2 – Tableau de fils sur carré – par liste Blockly - et sa colorisation



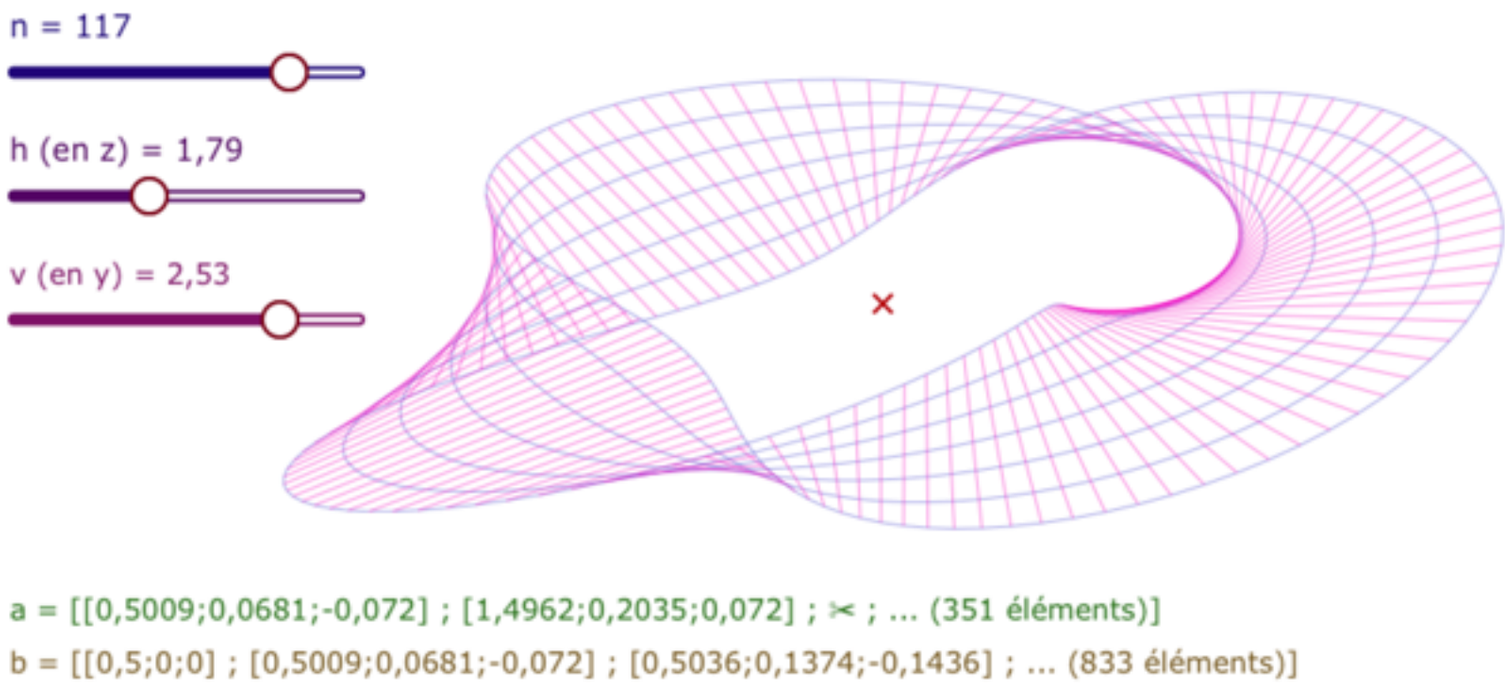
*En mode **consultation** à l'ouverture. Le code est significativement différent de celui de la même figure avec la tortue.*

Aspect dynamique sur une liste

Cette construction du **ruban de Moëbius** est l'occasion d'illustrer deux points particuliers.

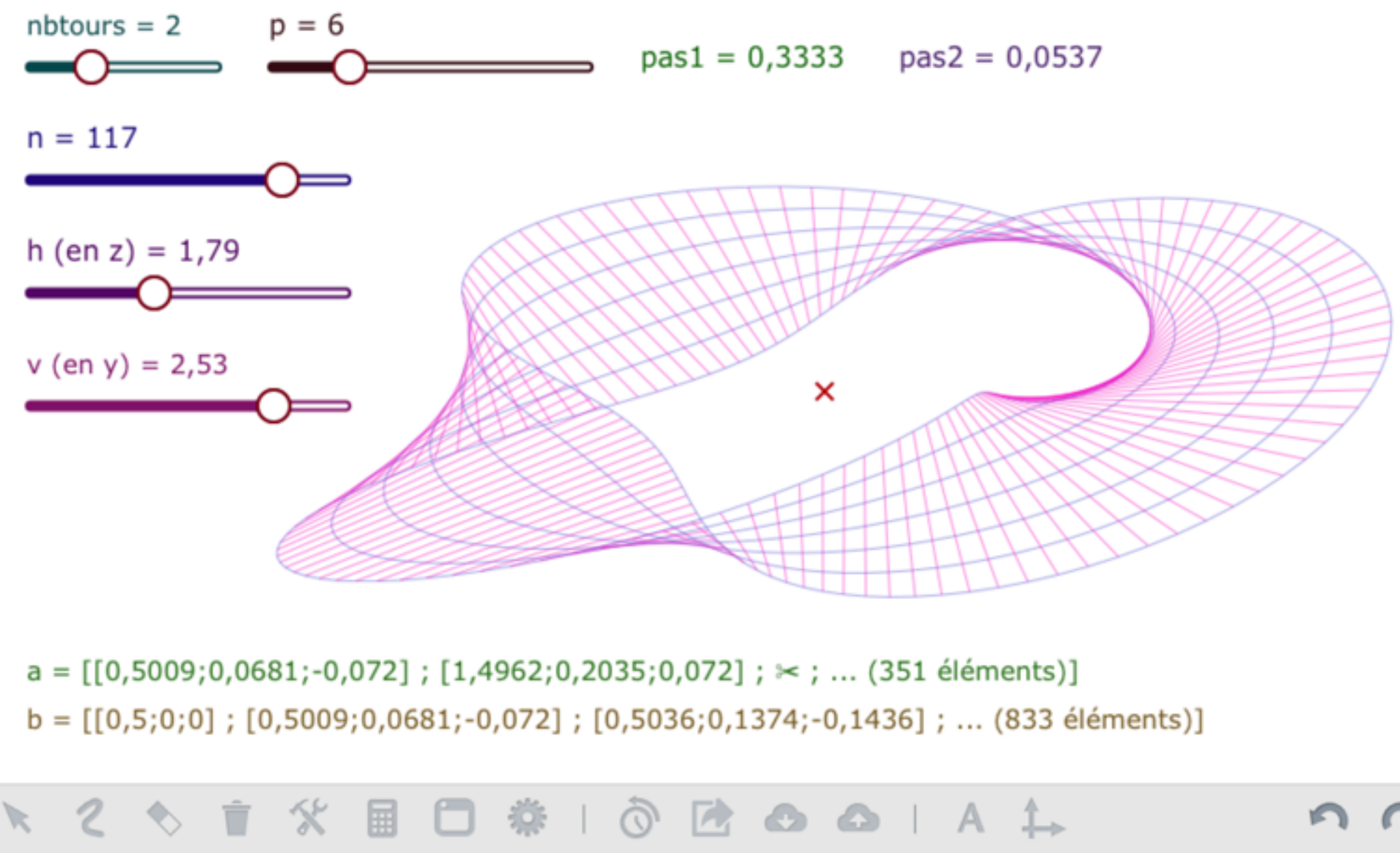
On peut **briser les listes de segments**. C'est ce que l'on voit avec les ciseaux sur la liste **a** qui trace les segments verticaux roses.

On peut utiliser des **blocs sur l'orientation du trièdre** dans toutes sortes de situations, et ici on en a placé dans la couleur du ruban.



Dans l'illustration du code ci-dessous on voit **la rupture de la liste** (la suite des segments). La partie horizontale de Moëbius est coupée à chaque cycles de **n** segments.
Ensuite, on voit l'utilisation de **l'angle théta** du trièdre et que, comme **aspect**, le bloc **RGB color** peut aussi s'appliquer non pas à la liste **a** ou **b** (comme vu dans les pages précédentes) mais à sa **représentation graphique**, notée automatiquement **l1** ici.
On distingue donc bien les listes (**a** et **b** dans l'illustration ci-dessus) et leurs représentations graphiques, comme on fait la différence entre la définition d'une fonction et sa représentation graphique.
Le bloc de couleurs peut s'appliquer à l'une comme à l'autre.

Figure dynamique 10.3 – Ruban de Moëbius par listes de Blockly – Colorisation selon phi et théta du repère

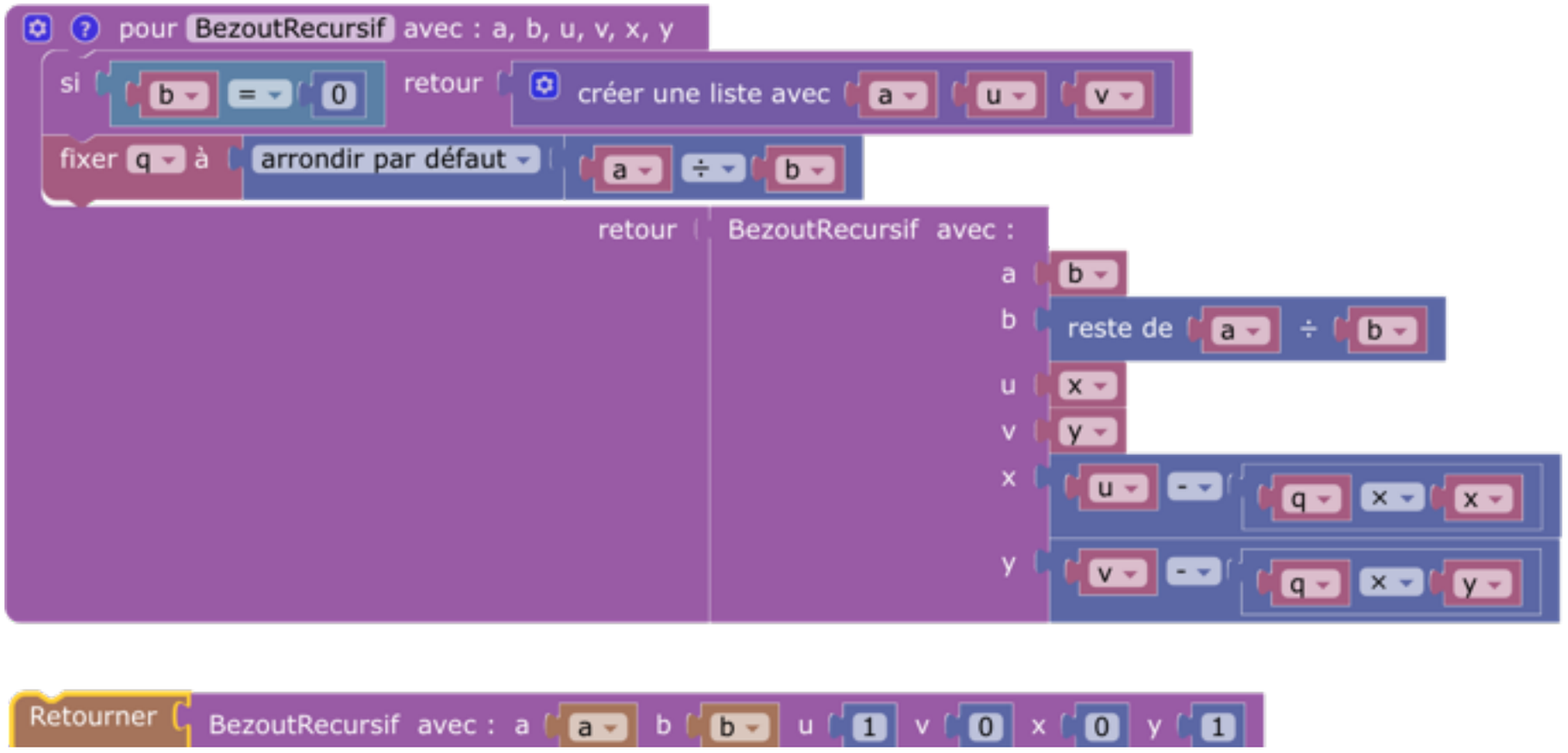


Déplacer au doigt le repère et observer que la couleur du ruban varie quand le trièdre change d'orientation.

Blockly et récursivité

Dans cette section nous ne verrons que deux exemples, ce premier, arithmétique : puis nous reviendrons sur une version «listes de Blockly» de la fractale de Pythagore.

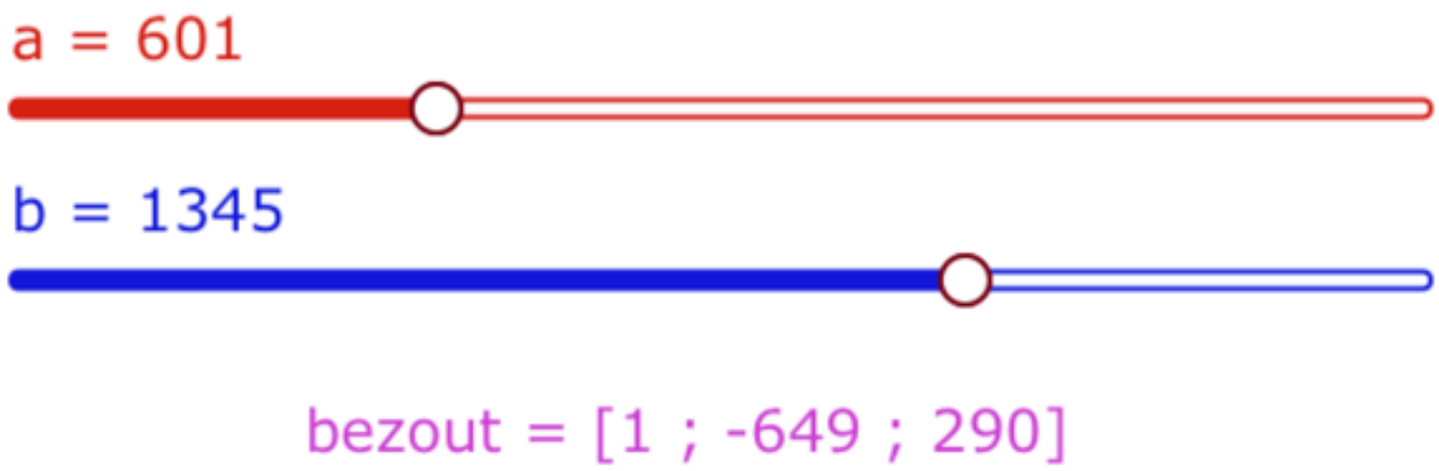
D'un point de vue arithmétique, voici l'exemple de l'algorithme de Blankenship qui réécrit l'algorithme de Bézout avec une notation matricielle qui permet une écriture rapide.



On l'a déjà mentionné, mais l'important ici est de remarquer que l'appel récursif est dans la sortie de la fonction.

On note qu'ici on sort de la fonction quand b est nul. La modification pour ajouter le tracé de l'anthyphérèse est alors élémentaire.

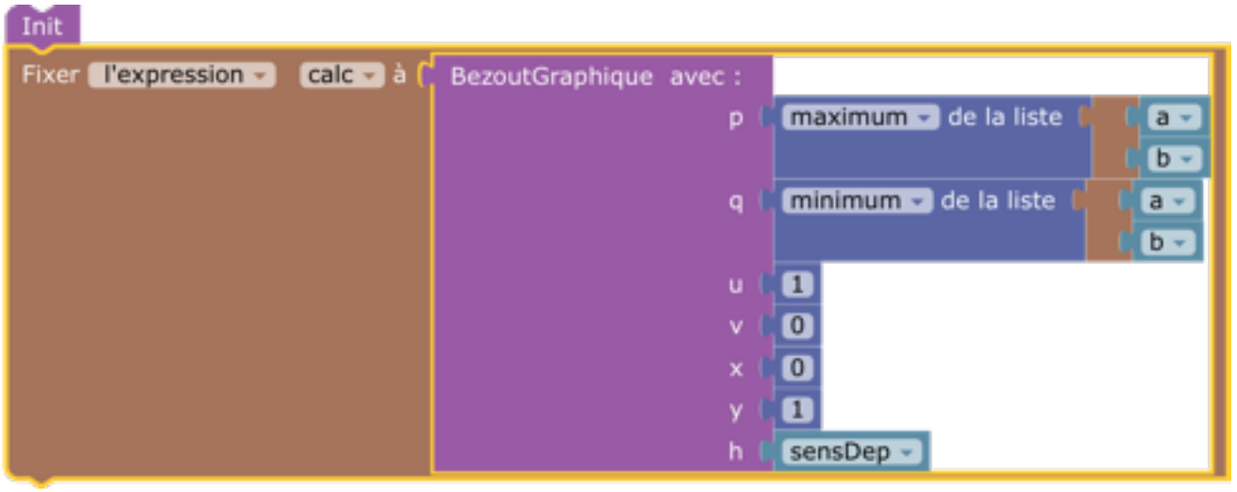
Figure dynamique 10.4 – Les coefficients de Bézout



*En mode **consultation** à l'ouverture de la figure. Le code est dans la liste **bezout**.*

Coefficients de Bézout et PGCD graphique

On conserve l'algorithme précédent. Il suffit d'ajouter le code de tracé des carrés du chapitre précédent.
La différence réside dans le fait que cette fois le code est dans le point A.
Comme la fonction renvoie une liste, cette liste est transmise à l'expression **calc** par un bloc **Fixer**, alors que dans le simple Bézout récursif, on utilisait un bloc **Retourner**.



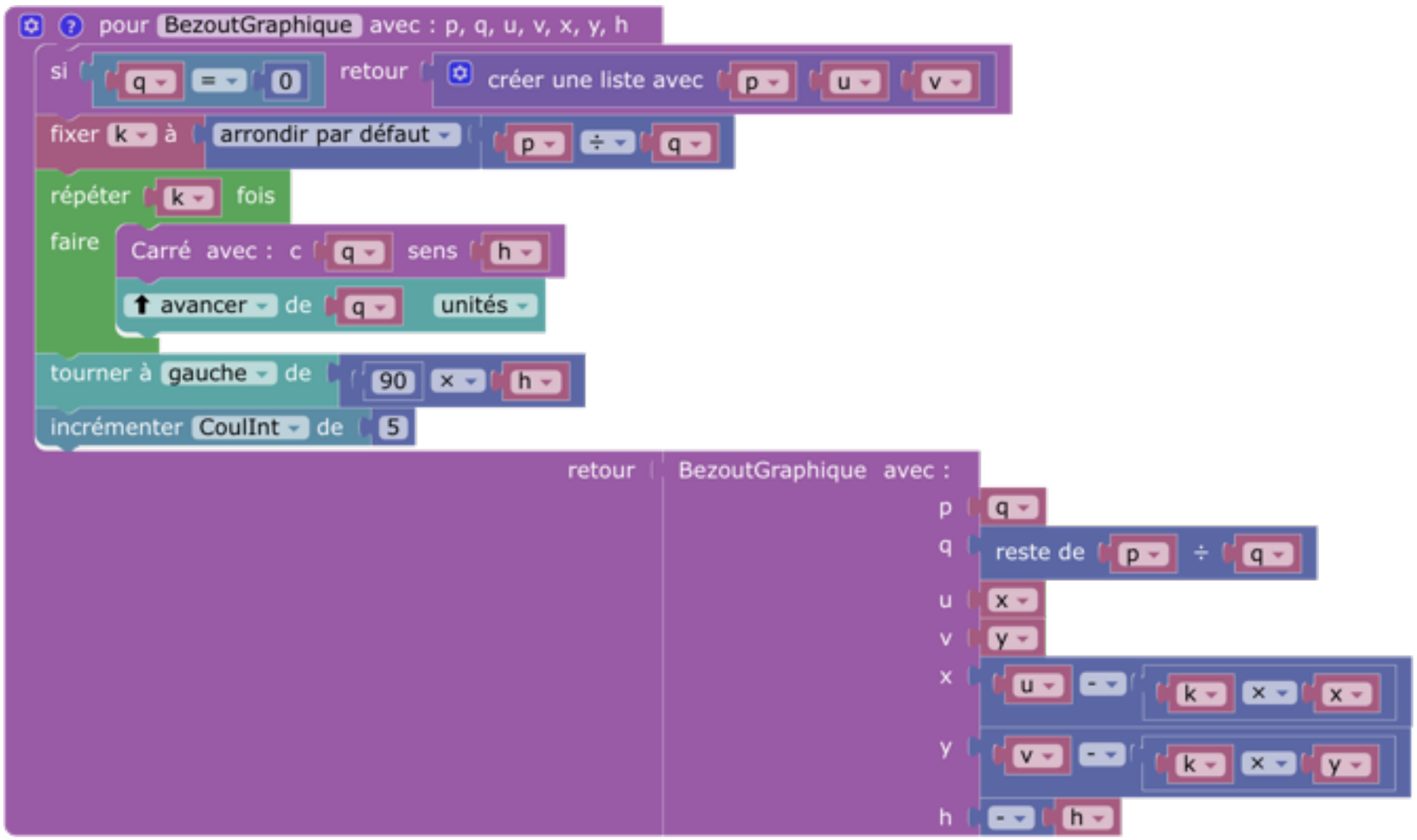
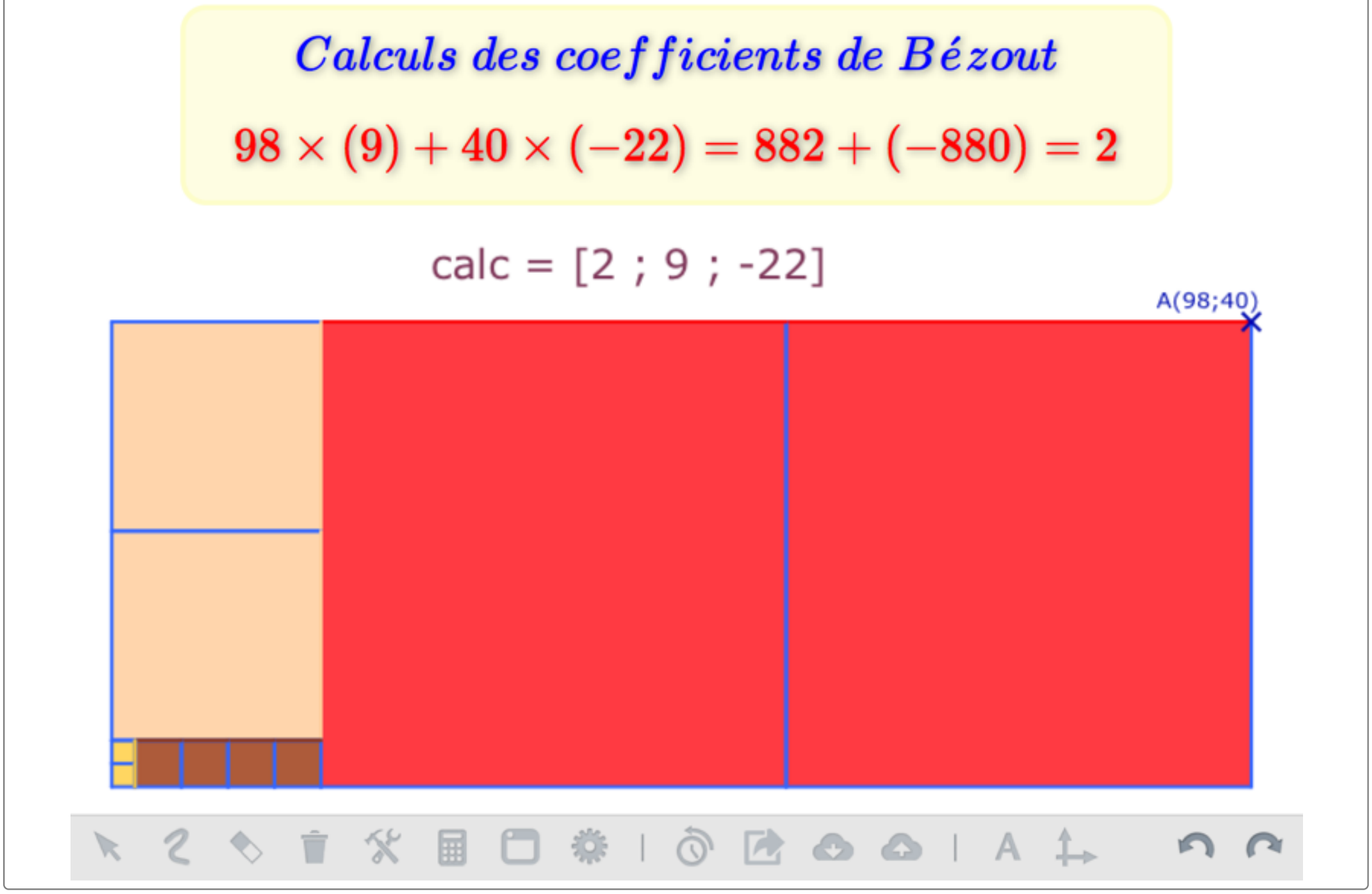
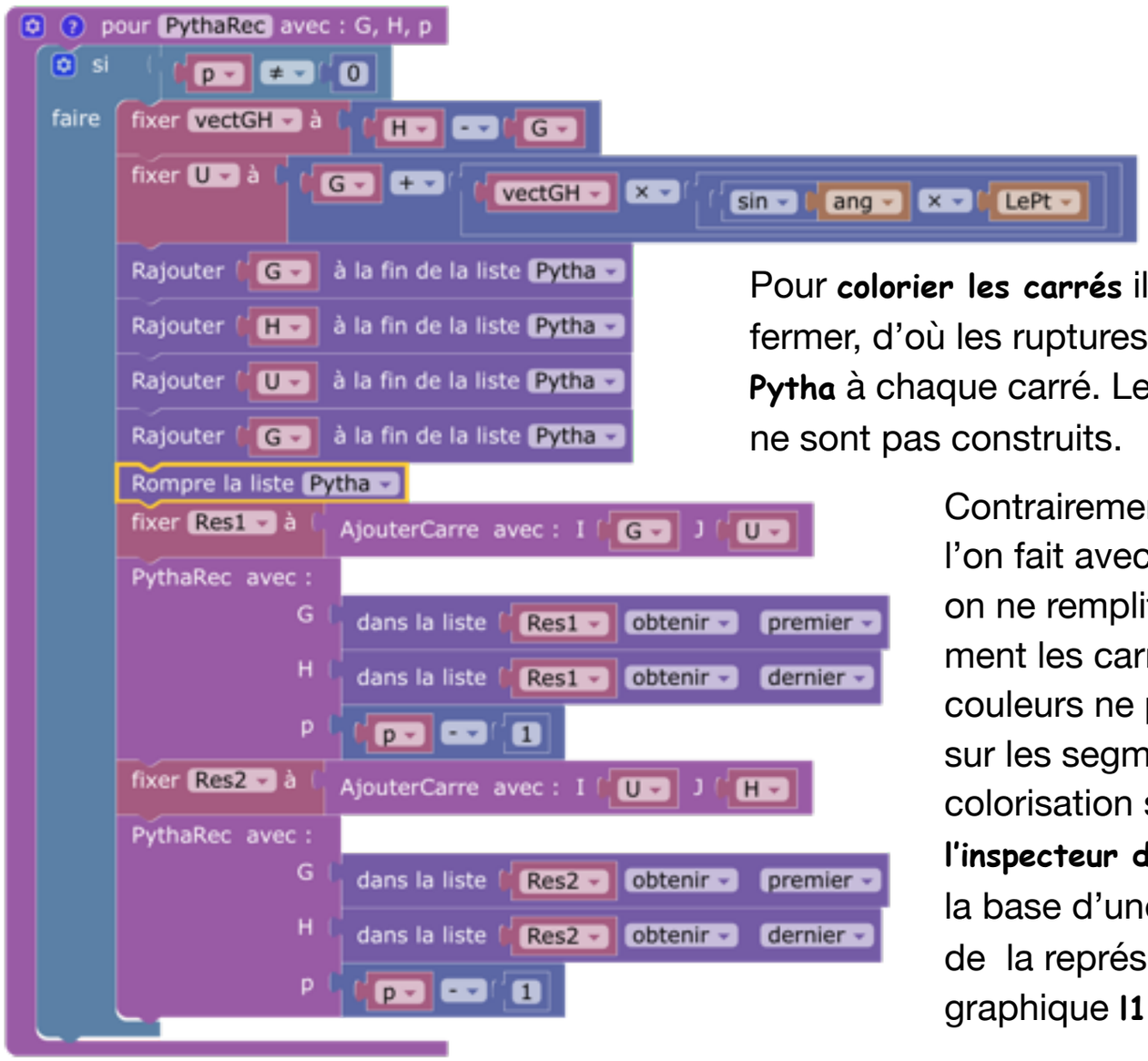


Figure dynamique 10.5 – PGCD graphique et coefficients de Bézout



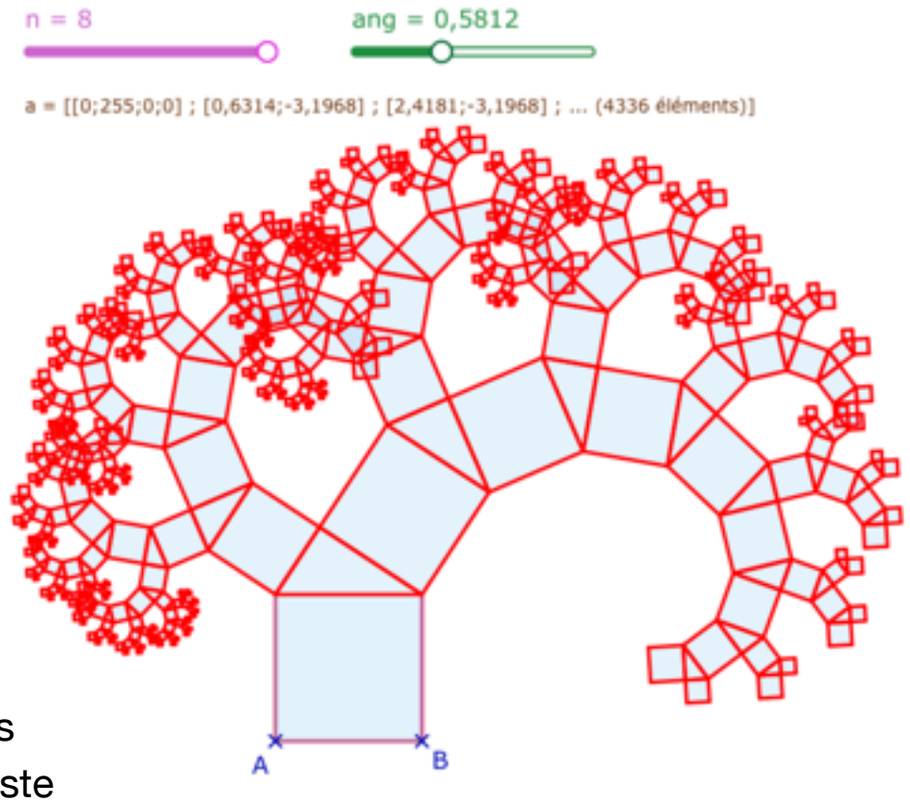
Version « Liste de Blockly » pour la fractale de Pythagore

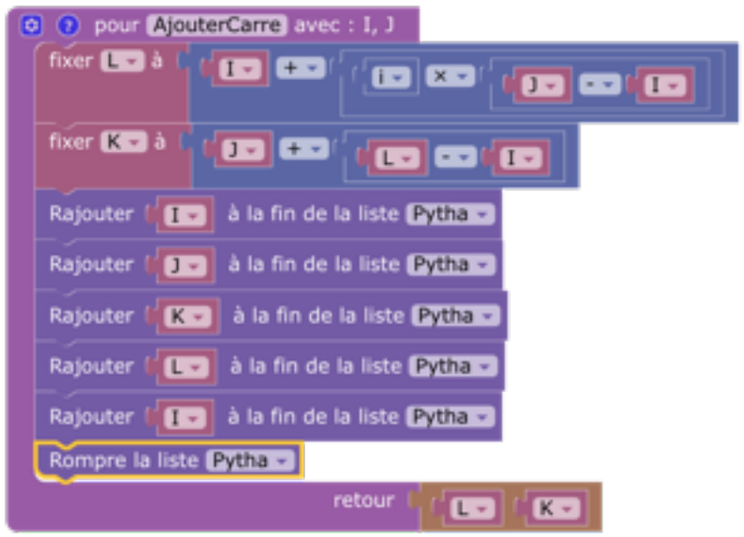
La construction de la fractale par la tortue a utilisé la **dimension algébrique** de la tortue pour affecter les variables locales aux sommets du futur carré. Ici, c'est le passage des sommets par la procédure de construction des carrés qui la remplace : la procédure **AjouterCarre** renvoie une liste de 2 sommets, aussitôt utilisée pour l'appel récursif.



Pour **colorier les carrés** il faut les fermer, d'où les ruptures de la liste **Pytha** à chaque carré. Les triangles ne sont pas construits.

Contrairement à ce que l'on fait avec la tortue, on ne remplit pas réellement les carrés, les couleurs ne portent que sur les segments. La colorisation se fait avec **l'inspecteur d'objets** sur la base d'une **opacité** de la représentation graphique **I1**.





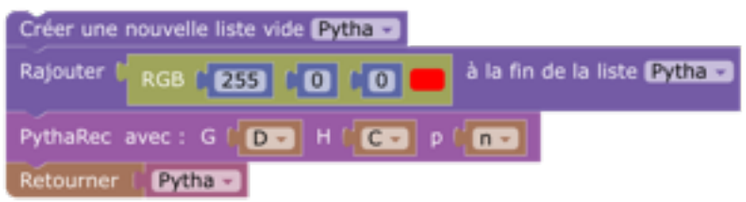
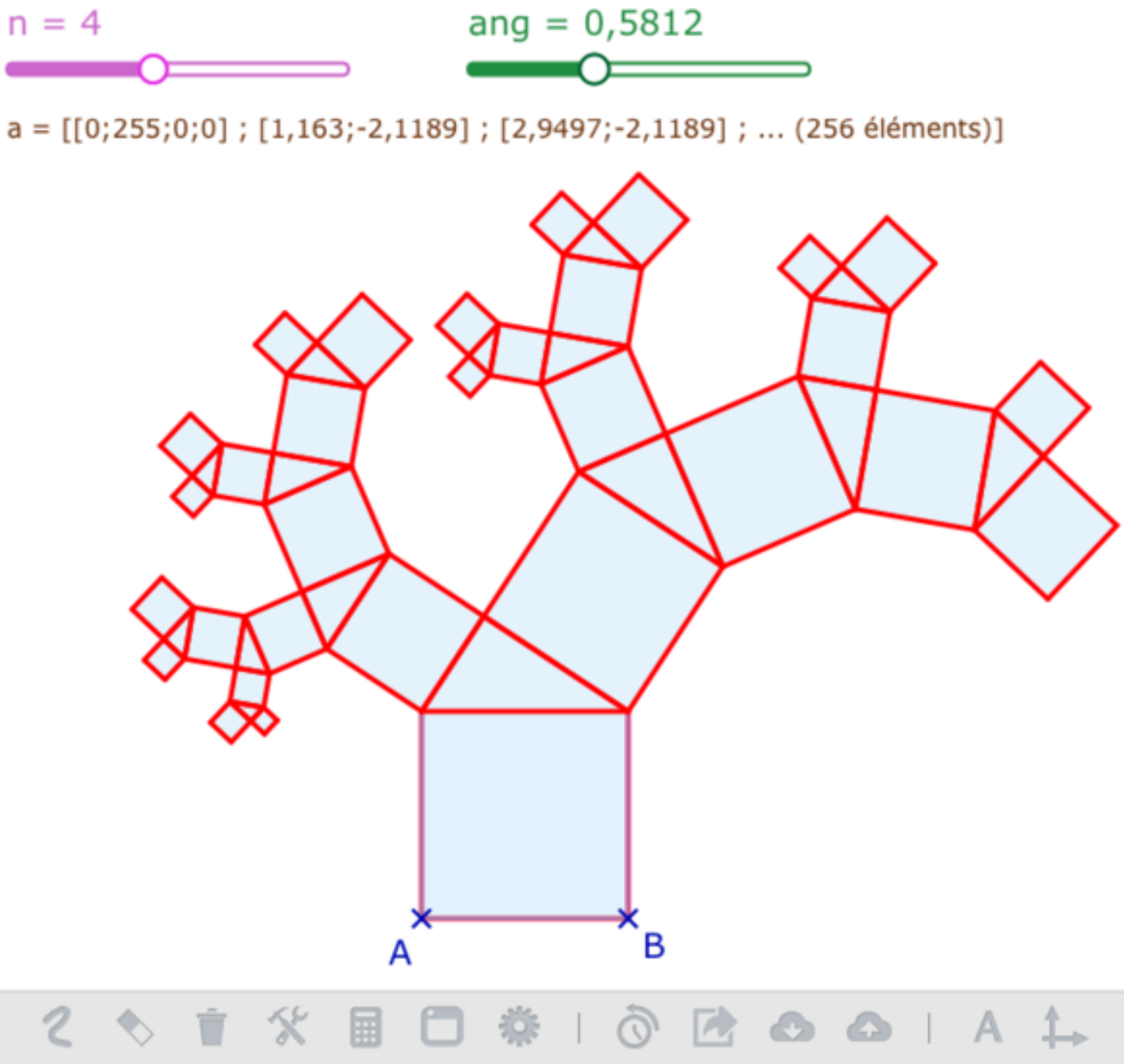


Figure dynamique 10.6 – Fractale de Pythagore par les listes Blockly



*Ne pas oublier que le code est dans la liste a (et pas dans le point A). En mode **consultation** à l'ouverture.*

Blockly et le déterminisme des figures

La subtilité de l'affectation par Blockly

On a déjà observé, dans le chapitre sur l'algébrisation de la tortue, que l'affectation par un programme Blockly d'un point a cette particularité extraordinaire à la fois d'être dynamique et en même temps de ne pas être une construction. Par exemple, si on affecte un point G à être, par positionnement de la tortue, sur l'isobarycentre d'un triangle, cette affectation est dynamique : quand on déplace A, B ou C, G est mis à jour. Mais G **n'est pas un point construit** comme le serait l'intersection de deux médianes : on peut le déplacer. Ce n'est pas directement une conséquence de l'implémentation de Blockly dans DGPad, c'est un choix de l'auteur de DGPad pour permettre ce que l'on va exposer maintenant.

L'auto-référence des objets

L'auto-référence des points ou des expressions dans un logiciel de géométrie dynamique n'est pas une nouveauté, c'est une propriété qui **est déjà présente dans CaR** (Compas and Rulers) le logiciel de René Grothmann, moteur original de CaRMetal. C'est donc, par héritage, une propriété de CaRMetal que nous avons déjà plusieurs fois exposé sur le site de l'IREM de La Réunion ou dans certains anciens articles de *MathémaTICE*.
L'auteur de CaRMetal a voulu faire, sur ce point aussi, mieux – bien mieux – avec DGPad.

En effet si la mise en oeuvre était assez ésotérique avec CaRMetal (ci-contre de l'auto-référence croisée des points P et Q), elle va devenir complètement transparente avec Blockly dans DGPad.

```
x(P) = x(@O)
y(P) = if(y(P)<y(@yMax);y(@O)+sqrt((x(@B)-x(@A))^2-(x(@Q)-
x(@O))^2)+0*x(P);y(@yMax))
```

Auto-référence d'un point - l'exemple archétypique

Déterminisme - rupture du déterminisme : la mémoire de l'action

L'exemple introductif est toujours le même, car c'est l'archétype de la rupture conceptuelle qu'il procure dans la géométrie dynamique. En effet, une des règles de base du cahier des charges des logiciels de géométrie dynamique est, d'une certaine façon, la référence implicite causale : si un élève remet les points de base d'une figure dans une position initiale, toute la figure retrouve sa position initiale. C'est ce que l'on appelle le **déterminisme**. Et c'est à tout le moins, d'un point de vue didactique, ce que l'on attend en classe.

Sans entrer dans les détails théoriques (théorème l'universalité de Mnev) **ce déterminisme n'est pas compatible avec la notion de continuité** : avec le déterminisme, les instances d'une figure (les dessins à l'écran), si on veut conserver les orientations locales, ne sont alors pas nécessairement toutes atteintes. Le dilemme de la géométrie dynamique est donc de conserver le plus possible le déterminisme tout en maintenant la continuité le plus loin possible, et en particulier celle de l'intersection des objets.

Les logiciels ont fait beaucoup de progrès depuis les toutes premières versions de Cabri-géomètre. Des techniques systématiques permettent de repérer, en interne, les intersections des cercles et des droites ou des cercles, par exemple, et d'en tenir compte de manière transparente pour l'utilisateur. Pourtant, il reste toujours facile de mettre en défaut leur fonctionnement standard, que ce soit des logiciels en « mode continuité », ou bien entendu CaRMetal ou DGPad. Il faut juste savoir exactement ce que l'on fait et comment le corriger si nécessaire. Cela dit on peut faire (beaucoup) de géométrie dynamique en classe sans jamais vraiment rencontrer ces questions.

L'exemple symbolique du thermomètre à mémoire (on s'offre ici un thermomètre horizontal)

On se donne une droite parallèle à l'axe des abscisses, et un point M sur cette droite. On se propose de construire deux points **Min** et **Max** qui sont les « **curseurs-mémoire** » des positions minimales et maximales de M quand l'utilisateur déplace M sur la droite. Si on arrive à construire ces deux points, on est dans la **brisure du déterminisme,** car en remettant M à la position initiale on veut que ces deux points restent à leur place, et donc gardent une trace de l'histoire de la manipulation.
D'une certaine façon, de manière simplifiée, on insère le temps de l'utilisateur dans la figure. Or, l'implémentation de Blockly dans DGPad permet de faire cela tout simplement.

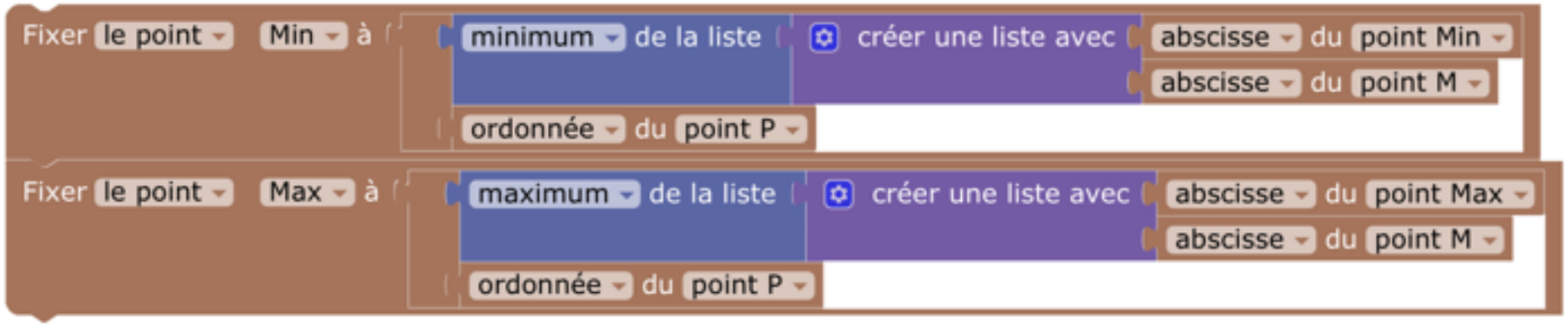
Rompre et rétablir le déterminisme

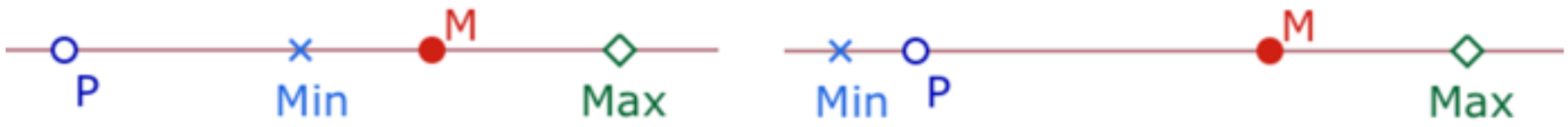
Le code en M a du sens car Blockly permet l'auto-référence : **Min** est défini à partir de **Min**.

On observera que, dans ce contexte, **l'initialisation** de cette démarche récursive **est l'existence** du point : le fait de créer un point **Min** l'initialise.

Le point M est un point sur objet de la droite parallèle à (Ox) – l'axe – passant par un point P.

On **réinitialise le processus** dans le déplacement du point P qui replace **Min** et **Max** à ce que l'on peut appeler les conditions initiales. Et donc, si on brise le déterminisme, on sait aussi construire les outils qui permettent de le rétablir.





En haut le code en M dans l'onglet Déplacé et son application

Ci-contre le code en P dans l'onglet Déplacé

Ci dessous à gauche l'application de P, à droite les points ne sont pas liés à la droite.

Techniquement on remarque qu'ici les coordonnées de **Min** et **Max** sont déterminées dans des blocs de comportement de M, et la réinitialisation dans des blocs de P.
Avec CaRMetal, les codes étaient nécessairement dans les coordonnées des points **Min** et **Max** eux-mêmes.

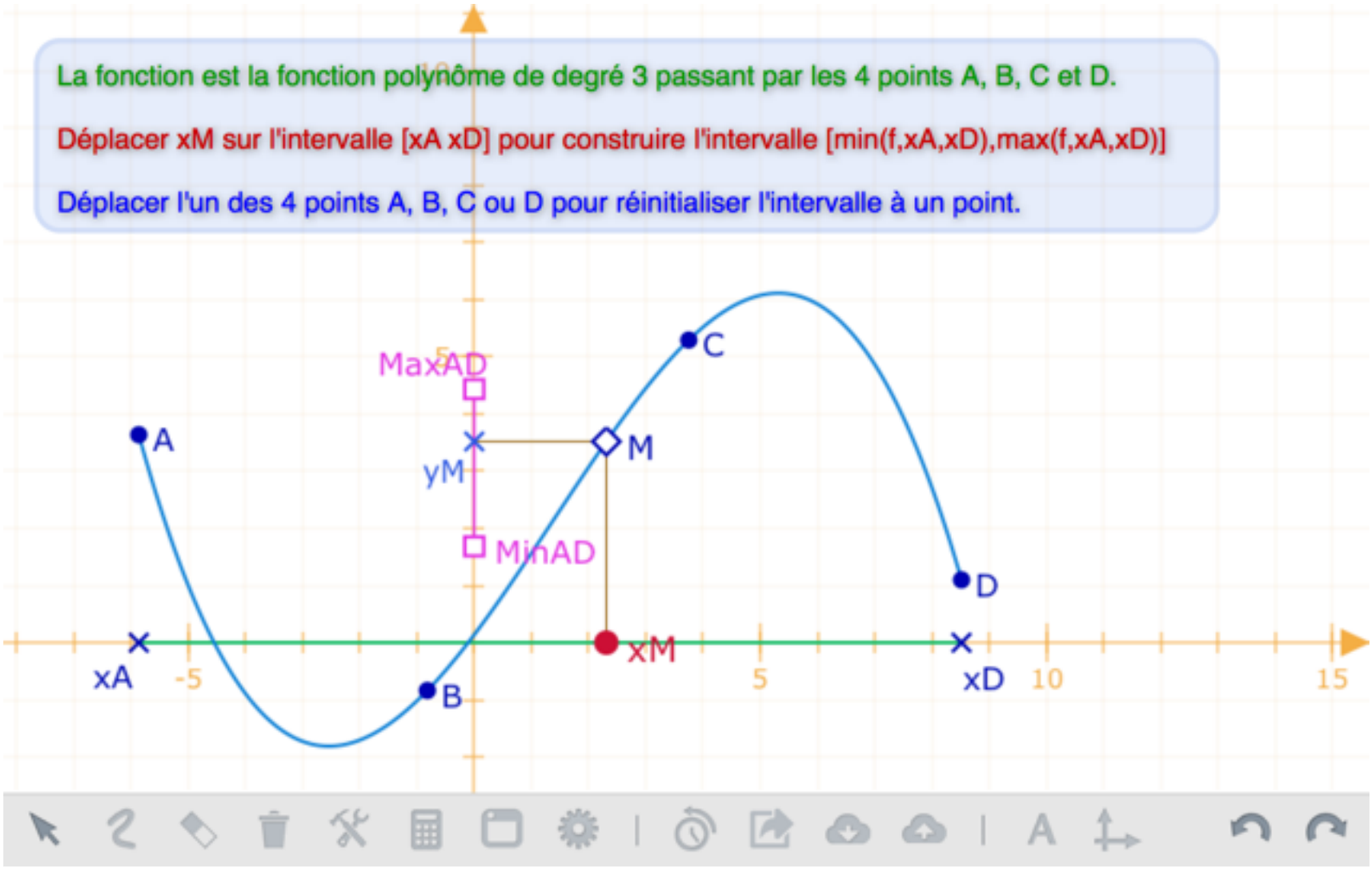
Figure dynamique 11.1 – Abscisses Min et Max à mémoire – Rupture et restitution du déterminisme

*La figure est en mode **standard** à l'ouverture. Faire les différentes manipulations décrites dans la page précédente.*

Dans la page suivante, on applique cette technique à des fins didactiques pour faire prendre conscience, kinesthésiquement, avec le doigt, que **le sens de variation d'une fonction sur un intervalle est un concept global** alors que souvent les élèves en font, en acte, un concept local, limité aux extrémités de l'intervalle étudié.

L'expérience utilisateur de la tablette a une dimension tactile parfois d'une grande pertinence, pour aborder autrement certains concepts. Ici la figure est construite sur la nécessité de parcourir tout un intervalle pour voir se construire – temporellement – les extrema de la fonction : **la rupture du déterminisme devient un outil au service de l'apprentissage** d'une dimension nouvelle et difficile du champ fonctionnel : la globalité d'un concept (alors que l'algèbre est plus sur la localité).

Figure dynamique 11.2 – Application : parcourir l'intervalle [xA,xD] pour visualiser les extrema de la fonction

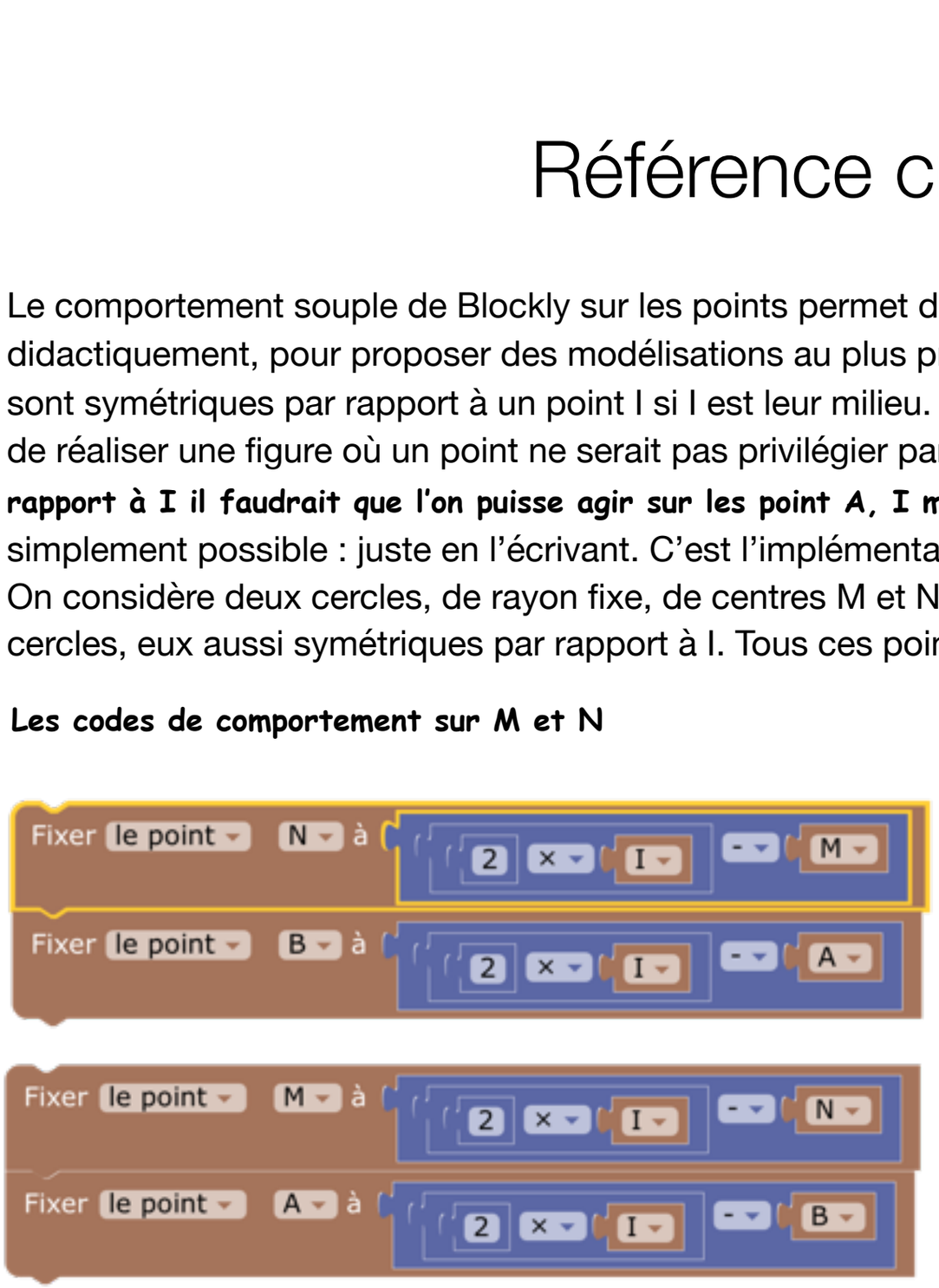


Conceptuellement, déplacer M est dans la rupture du déterminisme alors que déplacer un des points A, B C ou D le rétablit.

Référence croisée de points

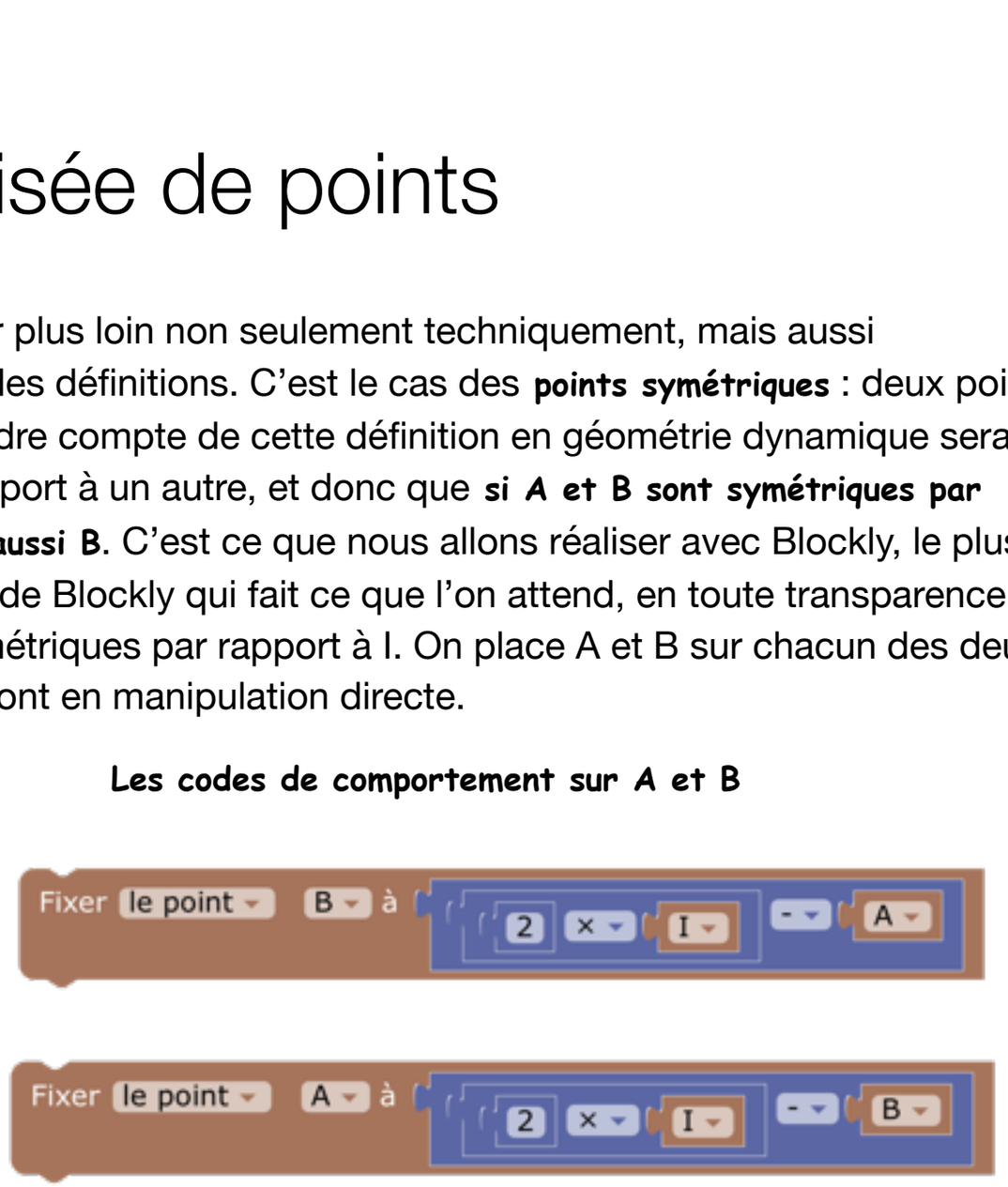
Le comportement souple de Blockly sur les points permet d'aller plus loin non seulement techniquement, mais aussi didactiquement, pour proposer des modélisations au plus près des définitions. C'est le cas des **points symétriques** : deux points sont symétriques par rapport à un point I si I est leur milieu. Rendre compte de cette définition en géométrie dynamique serait de réaliser une figure où un point ne serait pas privilégier par rapport à un autre, et donc que **si A et B sont symétriques par rapport à I il faudrait que l'on puisse agir sur les point A, I mais aussi B**. C'est ce que nous allons réaliser avec Blockly, le plus simplement possible : juste en l'écrivant. C'est l'implémentation de Blockly qui fait ce que l'on attend, en toute transparence. On considère deux cercles, de rayon fixe, de centres M et N symétriques par rapport à I. On place A et B sur chacun des deux cercles, eux aussi symétriques par rapport à I. Tous ces points sont en manipulation directe.

Les codes de comportement sur M et N




```
20 STL(B,"c:#0000b2;s:6;sn:true;f:24;mg:[C2,5000]");
21 STL(A,"c:#0000b2;s:6;sn:true;f:24;mg:[C1,5000]");
```

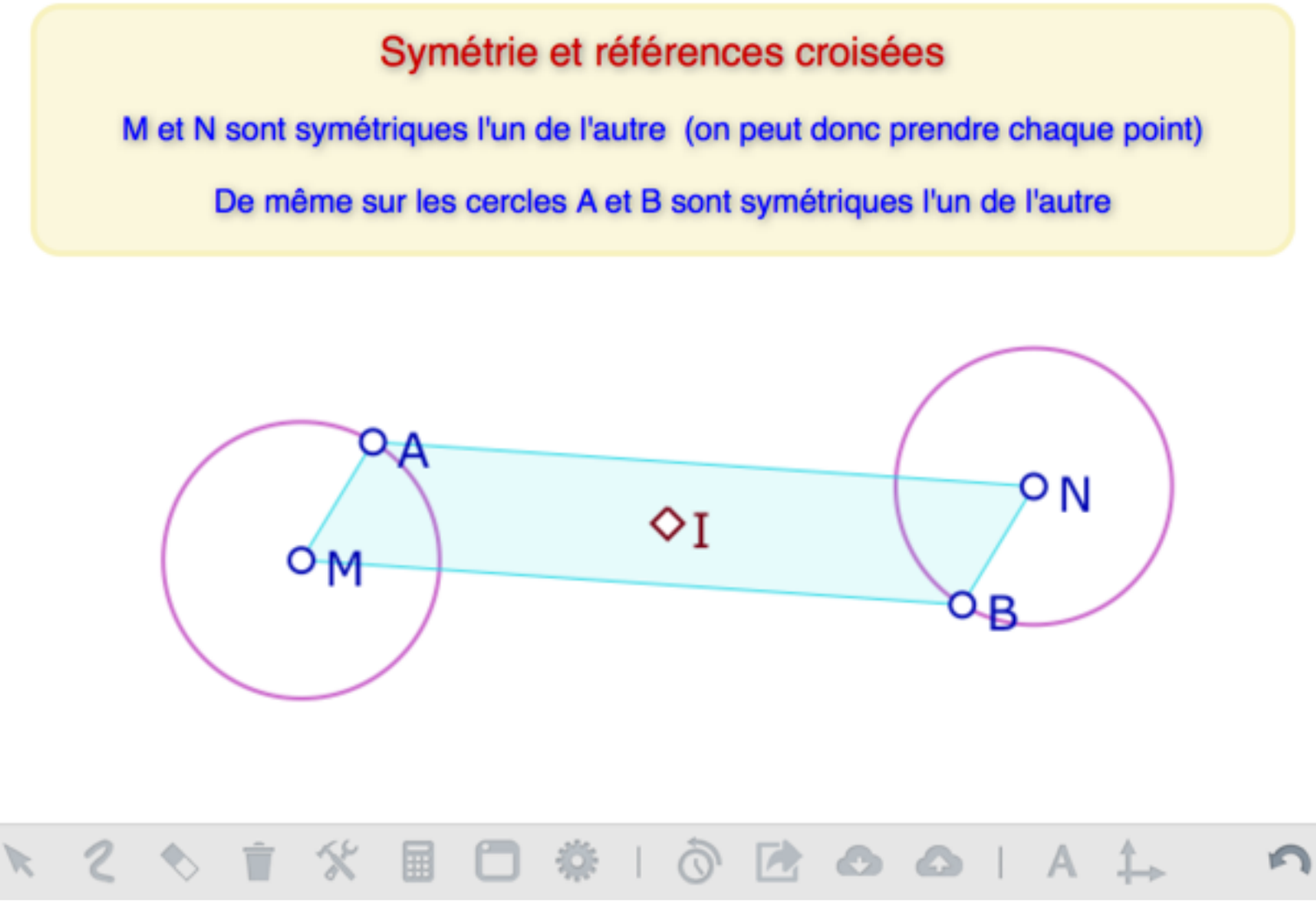
Les codes de comportement sur A et B



Mais comment est-ce possible ? Pourquoi A et B restent sur les cercles quand on les prend en main ?

C'est la seule question à se poser. La réponse est toute simple : il y a un mélange de type. **A et B sont aussi aimantés** à 5000 pixels (soit très fortement) aux cercles ; donc ils restent sur les cercles. On le voit dans le code du fichier ci-contre, **mg** étant le codage interne du magnétisme.

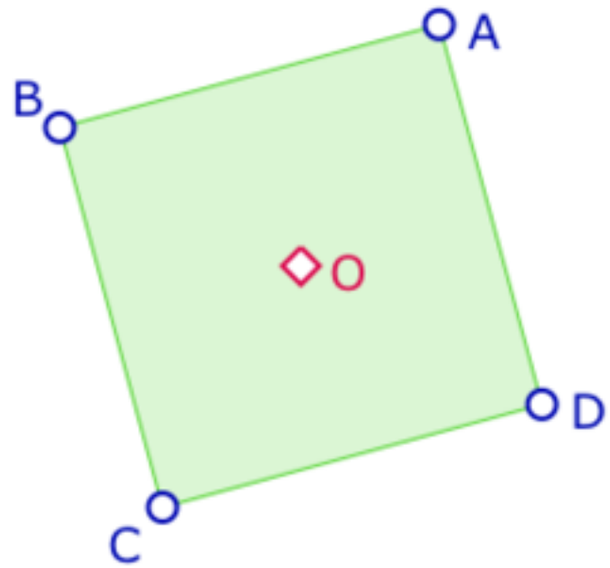
Figure dynamique 11.3 – Utilisation didactique de la référence croisée – Mise en oeuvre des points symétriques



*En mode **consultation** à l'ouverture. On peut déplacer les 5 points de la figure.*

Extension de virtualisation des définitions. Dans la figure suivante on étend le principe de référence croisée à la **référence croisée multiple**. On a construit un carré ABCD de centre O, construction pour laquelle on peut déplacer A mais aussi B, C ou D, et même O. Le carré est donc de centre O passant par mais aussi de centre O passant par B. C'est un autre type de carré...

Figure dynamique 11.4 – Références croisées multiples – Exemple d'un carré



On peut déplacer tous les points de la figure. C'est un peu moins stable quand on déplace O car il faut garder trace de la translation pendant qu'on l'effectue.

Ressources

DGPad est un logiciel écrit pour tablettes, il est disponible sous iOS et Android. Toutefois la partie Blockly n'est pas encore implémentée dans la version Android. Il est libre, sur GitHub : https://github.com/dgpad/dgpad

C'est aussi une webApp (tout environnement) : www.dgpad.net/index.php

Il existe une version Mac OS : http://docs.dgpad.net/downloads/macos/dgpad_install.dmg qui contient explicitement un export pour les livres interactifs. Il existe aussi une version locale linux (distribution Debian, dont Ubuntu)

Les nouveautés sur le logiciel se trouvent à la fois sur la chaine YouTube de l'auteur, mais aussi sur le site maintenu par Patrice Debrabant : carmetal.org qui contient un forum sur DGPad ainsi qu'une galerie de figures.

DGPad est aussi interfacé pour les DocTools (Google Drive), toujours de Eric Hakenholz. Ces outils permettent plusieurs choses, et par exemple de poser des quizz avec des questions ouvertes, entre autres sous DGPad, ce qui permet d'avoir une infinité - continue - de réponses exactes tout en pouvant les valider.

Concernant **la tortue de DGPad**, ce livre interactif est une version courte de cet article de la revue en ligne MathémaTiCE : http://revue.sesamath.net/spip.php?article875

L'IREM de Toulouse publie aussi régulièrement des compte rendus de formation, dont ceux-ci, de Monique Gironce, sur la tortue de DGPad, avec un regard différent, plus sur le calcul algébrique. Le premier est une présentation aux journées nationales de l'APMEP, d'octobre 2016, le second date de mars 2017. Ces liens contiennent des pages dynamiques.

http://www.ires-tlse-mathsetnumerique.fr/AtelierLyon/index.html
http://www.ires-tlse-mathsetnumerique.fr/AtelierLimoges/index.html