Dr.Geo, soit un géomètre

L'environnement GNU de géométrie interactive, version 19.06

Hilaire Fernandes (hilaire@drgeo.eu)

Ce manuel est pour GNU Dr.Geo (version 19.06), un environnement de géométrie interactive et de programmation.

Compilation : 5 août 2019

Merci aux relecteurs de ce manuel : François Audirac, Odile Benassy, Gérard Blanchet, Christophe Gragnic, Marc Leygnac, Yves Ouvrard, Thierry Pasquier, Jean Peyratout, Guy Veyssière.

Sommaire

Table des matières

Introduction

Dr.Geo permet de créer des figures géométriques et de les manipuler interactivement en respectant leurs contraintes géométriques. Il offre également la possibilité d'introduire graduellement la programmation. Il est ainsi utilisable dans des situations d'enseignement allant du niveau primaire au niveau supérieur. L'interface utilisateur de Dr.Geo a été conçue pour allier dans un ensemble harmonieux à la fois simplicité d'utilisation, ergonomie et fonctionnalités avancées.

L'interface simple de Dr.Geopermet au néophyte de se familiariser très rapidement avec les fonctions de base du logiciel. Puis, au cours de sa progression, l'utilisateur découvrira des aspects plus avancés de l'interface et du fonctionnement de Dr.Geo : modes multiples de construction des objets[1], macro-construction, enregistrement multiple, script et figure programmés en Pharo. Ces fonctionnalités plus complexes génèrent peu de surcharge sur l'interface, c'est pour cela que Dr.Geo reste simple à utiliser en enseignement primaire, tout en restant intéressant pour le lycée.

Dans les sections suivantes, les outils de base sont exposés. Ensuite les fonctionnalités avancées sont présentées en détail.

A l'ouverture de Dr.Geo, l'environnement est vide, il suffit de faire ...Clic arrière-plan → Nouveau... pour obtenir une figure vierge.

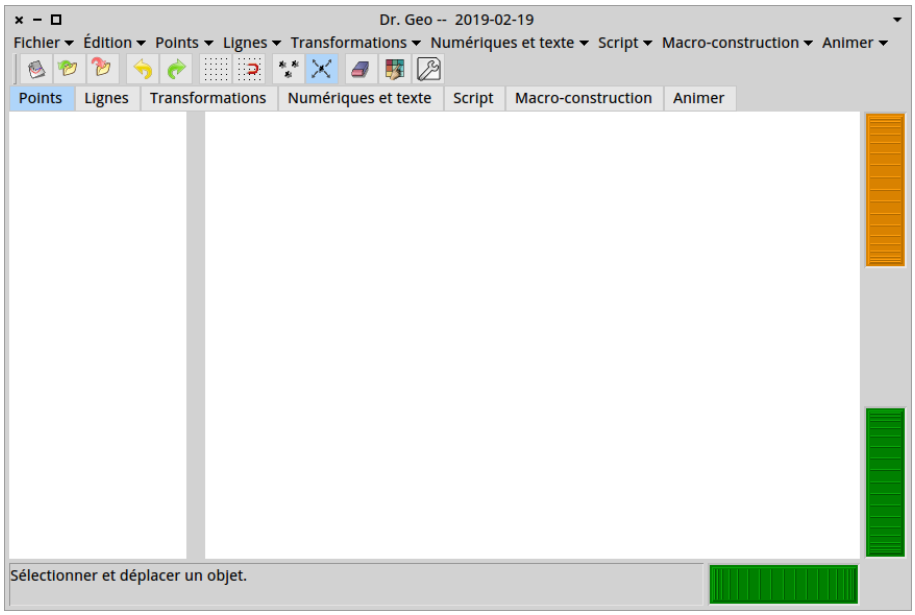

Figure 1: Fenêtre Dr.Geo avec une figure vierge

L'agencement de l'interface est comme suit :

1. La *barre de menu* caractéristique avec : Fichier – Édition – Points – Lignes – Transformations – Numériques et texte – Script – Macro-construction – Animer.

[1] À partir d'une même commande, il s'agit de créer un type d'objet selon des modalités différentes. Par exemple à partir de la commande construction de cercle, l'utilisateur peut créer un cercle défini par son centre et un point, ou bien une longueur, etc. Bien sûr cette commande n'est représentée que par une seule icône, il incombe à Dr.Geo d'anticiper sur la construction de l'utilisateur. L'effet immédiat est donc une diminution de la charge cognitive de l'interface sur l'utilisateur, tout en proposant un nombre important de modes de construction.

2. Une *barre d'outils* : ouvrir et sauver des figures ; défaire/refaire les dernières actions ; afficher la grille ; coller à la grille ; bascule entre édition multiple ou simple ; attraper un objet ; supprimer un objet ; modifier le style et les propriétés d'un objet.

3. La *barre d'outils* regroupant dans divers onglets les fonctions de construction présentes dans la barre de menu ;

4. Dans le coin en bas à droite, l'utilisateur a à sa disposition deux molettes pour déplacer horizontalement et verticalement la figure.

5. La seconde molette à droite, en haut, pour changer l'échelle de la figure.

Pour créer une nouvelle figure géométrique, l'utilisateur choisit la commande Nouveau dans le menu Fichier. Pour plus de concision, nous indiquerons dorénavant les commandes de menu sous la forme ...Fichier → Nouveau... Pour chaque nouvelle figure, une fenêtre distincte est proposée avec ses propres barres de menus et d'outils. L'utilisateur peut alors créer des points, lignes, cercle, etc. et contrôler leurs propriétés.

Dr.Geo II est un logiciel libre[2] multiplate-formes de géométrie interactive. Il est une réécriture complète de Dr.Geo 1.1 en Pharo. Pharo[3] a été utilisé pour ce faire. Dr.Geo 1.1 était écrit en C++ et intégrait un interpréteur Scheme pour la rédaction de scripts et de figures programmées. Dr.Geo II permet également l'intégration de scripts dans les figures géométriques ainsi que l'écriture de figures interactives entièrement décrites avec un langage de programmation.

Le choix d'une réécriture en Pharo fut motivé par les qualités dynamiques uniques de ce langage ; celui-ci nous permet en effet de pousser extrêmement loin nos investigations sur les dimensions interactives entre l'utilisateur et le logiciel. Ainsi Dr.Geo n'est pas seulement un logiciel convivial de géométrie interactive mais aussi, tel que distribué, un environnement complet de programmation dans lequel le logiciel peut être étudié, modifié et amélioré.

Pour s'en convaincre, l'utilisateur est invité à faire ...Clic arrière-plan → Outils → Navigateur système... Le navigateur de classes alors affiché est un outil pour parcourir et modifier le code source de Dr.Geo alors que celui-ci est en fonctionnement.

Cet accès au code source du logiciel, pour l'étudier, le modifier et le redistribuer est complètement ancré dans l'esprit du logiciel libre pour une approche non verrouillé à une informatique autre que de béatitude. Loin de nous l'idée de prétendre que Dr.Geo permet de rendre les esprits plus alertes, néanmoins il y contribue assurément.

[2] Un logiciel est libre lorsque son code source peut être étudié, modifié et redistribué.

[3] Pharo est une implémentation libre du language Smalltalk, `http://pharo.org`

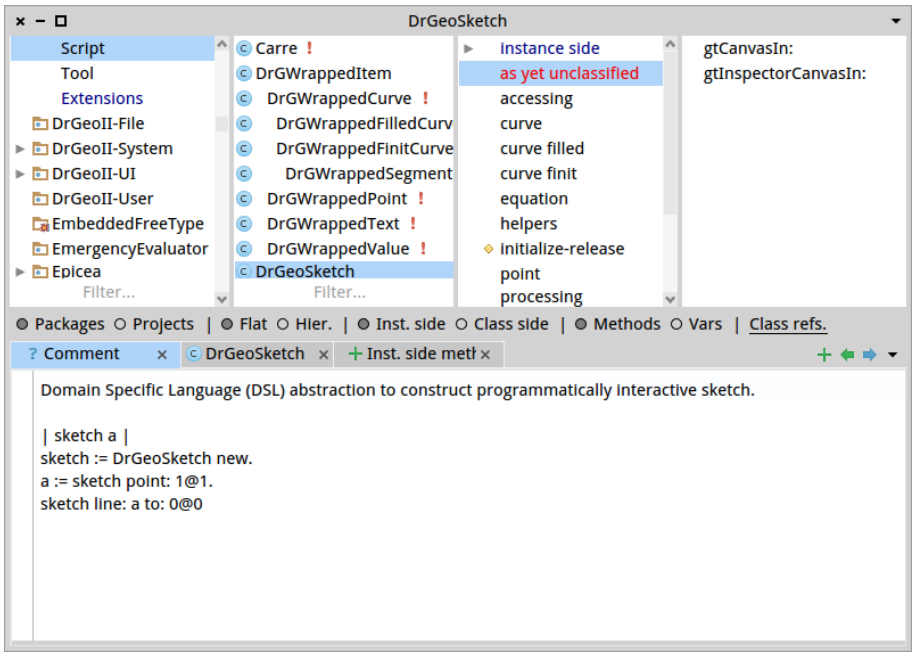

Figure 2: La navigateur du code source de Dr.Geo depuis Dr.Geo lui-même

Avec ce même esprit, les systèmes de figures programmées et de scripts – présentés dans les sections sur les outils avancés – sont adossés à un outillage évolué de mise au point du code : navigateur, débogueur, inspecteur d'objet. Dans la suite du document, nous nommerons indifféremment le logiciel Dr.Geo II ou Dr.Geo.

Dr.Geo dispose de son propre espace web à l'adresse : `http://drgeo.eu`.

Sur cet espace, l'utilisateur trouvera les informations suivantes :

- comment obtenir Dr.Geo ;
- la documentation sur le logiciel ;
- des indications pour s'impliquer dans le projet Dr.Geo ;
- des références sur des exploitations pédagogiques du logiciel.

Partie I
Pour bien démarrer

1 Fonctions de base

Ce chapitre décrit les outils utilisés pour construire une figure géométrique. Ces outils sont rangés dans six groupes accessibles à partir de la seconde barre d'outils de Dr.Geo.

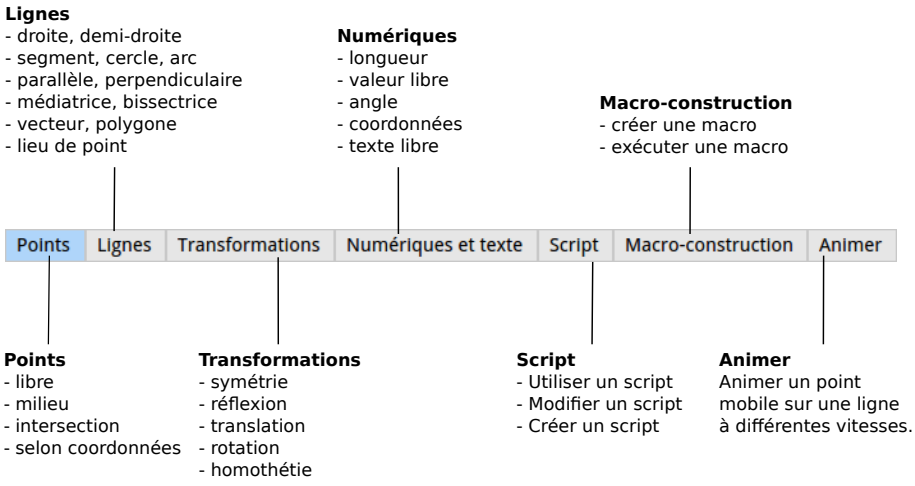

Figure 1.1: Catégories d'outils de Dr.Geo et leurs descriptions

Par défaut, Dr.Geo opère en mode fonctionnel. L'utilisateur clique sur un onglet de fonction puis sur le bouton d'une fonction précise ou il choisit une fonction depuis un menu ; ensuite seulement il sélectionne le ou les objet(s) de la figure pour appliquer la fonction une seule fois. À ce moment, l'application retourne dans son état par défaut où l'utilisateur choisit et déplace les objets. A tout moment, l'utilisateur peut abandonner la fonction avant la fin de la sélection finale avec le bouton "Choisir et Déplacer", ou le menu ...Édition → Choisir et déplacer...

C'est le contraire du mode modale, où le clic sur une fonction place l'application dans un mode permanent où la commande s'utilise plusieurs fois de suite. Pour activer le mode modale, l'utilisateur choisit le menu ...Édition → Création multiple... Les commandes pour éditer les styles et propriétés d'un objet sont toujours modales, cela permet de modifier plusieurs objets sans avoir à réactiver la commande. L'utilisateur doit sortir explicitement de ces fonctions modales en choisissant la commande choisir et déplacer ou toute autre commande de construction.

Lorsque l'utilisateur clique sur un des boutons de cette barre d'outils, une liste supplémentaire de boutons s'affiche immédiatement. Celle-ci regroupe des fonctions d'une même famille.

De la gauche vers la droite, nous avons accès aux outils pour construire des points et des lignes, utiliser des transformations, calculer des valeurs, animer des points mobiles, gérer les macro-constructions.

Ces fonctions se retrouvent à l'identique dans le menu en haut de chaque fenêtre de Dr.Geo. Nous présentons leur contenu dans les sections suivantes.

1.1 Points

Point

Crée un point libre dans le plan ou sur un objet unidimensionnel (segment, demi-droite, droite, arc de cercle, cercle, lieu) :

1. Dans le premier cas, le point créé est mobile librement dans toute la figure. Pour le construire l'utilisateur clique simplement sur le fond.
2. Dans le deuxième cas, le point n'est libre que dans l'objet unidimensionnel (ligne) où il a été créé ; il est collé sur l'objet. Pour construire ce type de point, l'utilisateur clique sur une ligne (c.-à-d. droite, demi-droite, segment, cercle, arc de cercle, etc.).

Depuis cette commande, l'utilisateur crée un **point d'intersection** entre deux lignes (c.-à-d. droite, demi-droite, segment, arc de cercle, cercle) ; il suffit de cliquer à l'intersection de celles-ci, Dr.Geo l'indique d'ailleurs par une bulle d'aide *Intersection*.

Comment placer un point avec des coordonnées données ?

Une possibilité est de placer deux valeurs libres dans la figure – Voir [outils numériques], page 11, – puis de construire le point ayant pour coordonnées ces deux valeurs – Voir [point défini par ses coordonnées], page 8. Cette possibilité a un avantage sur la précédente, le point ainsi construit ne peut pas être déplacé directement à la souris, le point est en quelque sorte bloqué dans sa position.

Milieu

Crée le milieu de deux points ou d'un segment :

1. Dans le premier cas, l'utilisateur sélectionne deux points.
2. Dans le deuxième cas, l'utilisateur sélectionne simplement un segment.

Intersection

Crée la ou les intersection(s) de deux lignes (i.e. droite, demi-droite, segment, arc de cercle, cercle). L'utilisateur doit sélectionner deux lignes. Lorsqu'une des lignes choisie est un arc de cercle ou un cercle alors deux points d'intersection sont créés.

Coordonnées

Crée un point défini par ses coordonnées. L'utilisateur doit soit sélectionner un script retournant un couple de coordonnées – Voir [coordonées par script], page 42, – soit sélectionner deux nombres : le premier correspond à l'abscisse, le second à l'ordonnée.

Comment placer un point contraint par ses coordonnées ?

Cette fonction est largement utilisée lorsque nous souhaitons par exemple construire le lieu d'un point. Cette construction suppose au préalable l'existence de deux valeurs – Voir [outils numériques], page 11, – le point est ensuite construit en sélectionnant ces deux valeurs.

1.2 Lignes

Droite

Crée une droite définie par deux points. L'utilisateur sélectionne deux points.

Droite parallèle

Crée une ligne parallèle à une direction et passant par un point. L'utilisateur sélectionne un point et une direction (c.-à-d. une droite, une demi-droite, un segment ou un vecteur).

Droite perpendiculaire

Crée une droite perpendiculaire à une direction et passant par un point. L'utilisateur sélectionne un point et une direction (c.-à-d. une droite, une demi-droite, un segment ou un vecteur).

Médiatrice

Crée la médiatrice d'un segment. L'utilisateur sélectionne un segment ou deux points, extrémités du segment.

Bissectrice

Crée la bissectrice d'un angle formé par trois points. L'utilisateur sélectionne soit un angle géométrique (défini par trois points), soit trois points, la bissectrice passe alors par le deuxième point.

Demi-droite

Crée une demi-droite définie par deux points. L'utilisateur sélectionne deux points, le premier est l'origine, le second appartient à la demi-droite.

Segment

Crée un segment donné par deux points.

Vecteur

Crée un vecteur donné par deux points. L'utilisateur sélectionne deux points, le premier est l'origine, le second est l'extrémité. Une fois le vecteur créé, celui-ci s'attrape et se déplace indépendamment des deux points. Ceci reste vrai pour un vecteur construit par une transformation – Voir [transformations], page 10.

Cercle

Crée un cercle. L'utilisateur crée un cercle selon différentes sélections :

1. le centre et un point du cercle ;
2. le centre et un segment dont la longueur est le rayon du cercle ;
3. le centre et un nombre (le rayon du cercle).

Pour créer un cercle à partir de son centre et son rayon, vous avez besoin d'un point et d'une valeur. Cette dernière doit exister avant de créer le cercle. Elle est de n'importe quelle sorte : une valeur calculée comme une distance ou une valeur libre que vous éditez. Pour créer une valeur voir les outils Numériques – Voir [outils numériques], page 11.

Arc de cercle

Crée un arc de cercle passant par trois points. Le premier est l'origine de l'arc, le troisième est l'extrémité, le second est un point de l'arc.

Arc (centre)

Crée un arc de cercle défini par son centre, une origine de l'arc et son ouverture. Le premier point choisi est le centre, le deuxième l'origine et le troisième point donne l'angle au centre de l'arc.

Polygone

Crée un polygone défini par n points. L'utilisateur sélectionne dans la figure n points, sommets du polygone ; puis pour indiquer la fin de sa sélection des sommets, il clique à nouveau sur le premier point choisi. Le premier et le dernier sont donc un seul et même point ce qui indique à Dr.Geo que la sélection est terminée.

Le polygone accepte des points mobiles sur son périmètre, mais il n'est pas possible de faire une intersection avec une autre ligne. En revanche la construction de l'image d'un polygone par une transformation géométrique est acceptée.

Polygone régulier

Crée un polygone régulier défini par deux points et une valeur numérique. L'utilisateur sélectionne son centre, un sommet et une valeur indiquant le nombre de sommets.

Lieu d'un point

Crée un lieu défini par deux points. L'utilisateur sélectionne deux points, l'un des deux est un point sur une ligne, l'autre est un point sous contraintes du premier (c.-à-d. quand l'un bouge, l'autre fait de même).

1.3 Transformations

Symétrie centrale

Crée l'image d'un objet par une symétrie centrale. L'utilisateur sélectionne l'objet à transformer et le centre de symétrie (un point). Quand l'utilisateur veut construire l'image d'un point, le premier point sélectionné est le centre de la symétrie.

Réflexion (symétrie axiale)

Crée l'image d'un objet par une symétrie axiale. L'utilisateur sélectionne l'objet à transformer et l'axe de symétrie (une droite). Quand l'utilisateur veut construire l'image d'une droite, la première droite sélectionnée est l'axe de la symétrie.

Translation

Crée l'image d'un objet par une translation. Quand l'utilisateur veut construire l'image d'un vecteur, le premier vecteur sélectionné est le vecteur de translation.

Rotation

Crée l'image d'un objet par une rotation. L'utilisateur sélectionne l'objet à transformer, le centre et l'angle de la rotation. Quand l'utilisateur veut créer l'image d'un point, le premier point sélectionné est le centre de la rotation.

L'angle de la rotation se définit de plusieurs façons :

- **valeur numérique** : l'angle est alors exprimé en radians (Exemples de valeurs numériques : valeur libre, distance entre deux points, longueur d'un segment, une coordonnée, une valeur retournée par un script Pharo Dr.Geo, etc.) ;
- **la mesure d'un angle géométrique ou orienté formé par trois points** : sa mesure est alors exprimée en degrés. La mesure appartient à l'intervalle [0 ; 180] ou [0 ; 360[. L'angle doit être défini avec une des commandes décrites dans la section angle ;

- **la mesure d'un angle orienté de deux vecteurs** : sa mesure est exprimée en degrés et couvre l'intervalle]-180 ; 180]. L'angle doit être défini avec la commande angle orienté en choisissant deux vecteurs.

Homothétie

Crée l'image d'un objet par une homothétie. L'utilisateur sélectionne l'objet à transformer, le centre et le facteur (c.-à-d. un nombre). Quand l'utilisateur veut créer l'image d'un point, le premier point sélectionné est le centre de l'homothétie.

1.4 Numériques et texte

Distance, longueur & nombre

Crée une valeur numérique. La valeur numérique, selon la sélection de l'utilisateur, est calculée ou bien saisie :

1. pour deux points c'est la distance entre ces deux points ;
2. pour un segment c'est la longueur de ce segment ;
3. pour un vecteur c'est la norme de ce vecteur ;
4. pour un cercle c'est le périmètre de ce cercle ;
5. pour un arc de cercle c'est la longueur de cet arc ;
6. pour une droite c'est la pente de cette droite ;
7. pour une droite et un point c'est la distance entre ce point et la droite ;
8. un **clic souris directement sur le fond de la figure** permet à l'utilisateur d'entrer une nouvelle valeur (c.-à-d. une valeur libre).

Cette dernière possibilité est très intéressante dans certaines situations. Elle permet de fixer une longueur, le rayon d'un cercle, la mesure d'angle (en radians) ou les coordonnées d'un point. La valeur est ensuite utilisée à partir des outils spécifiques de construction de cercle, de rotation et de point défini par ses coordonnées.

Angle orienté

Calcule la mesure d'un angle orienté défini par trois points ou deux vecteurs. Dans le premier cas, la mesure de l'angle appartient à l'intervalle [0 ; 360[, dans le second cas à l'intervalle]-180 ; 180].

Angle géométrique

Calcule la mesure d'un angle géométrique défini par trois points dont la mesure appartient à l'intervalle [0 ; 180].

Coordonnées, équation

Crée les coordonnées (abscisse et ordonnée) d'un point ou d'un vecteur. Donne l'équation d'une droite.

Texte

Ajoute un bloc de texte dans la figure. Cliquer à l'emplacement souhaité, puis éditer directement le texte.

Pour éditer à nouveau le texte, choisir l'outil de modification des propriétés des objets – Voir [éditer propriétés], page 15.

1.5 Script

Utiliser un script

Insère dans la figure un script préalablement défini. Après avoir choisi le script, la commande attend de l'utilisateur de choisir 0 ou n objets de la figure, puis un lieu désigné de la figure où afficher son résultat.

Un script s'utilise pour ses effets de bord sur la figure – modifications des autres objets de la figure – ou pour sa valeur de retour alors utilisée dans les constructions suivantes.

Modifier un script

Édite un script Pharo. Un éditeur de scripts est ouvert dans lequel l'utilisateur édite le script choisi.

Créer un script

Crée un script Pharo. L'utilisateur choisit le nom, une description et les paramètres du script.

Un éditeur de script est ouvert dans lequel l'utilisateur édite le script choisi. Pour créer de nouvelle méthode, il suffit de sauvegarder avec la commande `Ctrl-s`.

Les scripts Pharo Dr.Geo sont couverts en détails dans la partie des fonctionnalités avancées – Voir [fonctionalités avancées], page 27, – et exactement à la section script – Voir [script], page 34.

1.6 Macro construction

Construire une macro

Extrait une séquence de construction d'une figure et la transforme en macro-construction.

Exécuter/éditer une macro

Exécuter une macro-construction. La macro-construction est soit fraîchement créée ou chargée depuis un fichier.

Les macro-constructions sont présentées en détail dans la section macro-construction – Voir [macro-construction], page 28.

1.7 Autres outils

1.7.1 Supprimer un objet

Un objet d'une figure est supprimé en activant le menu ...Éditer → Supprimer... puis en le choisissant dans la figure. Les objets dépendants de l'objet effacé sont également supprimés. Lors de la suppression d'une ligne, les points utilisés pour la construire sont conservés. Cette action s'annule avec le bouton défaire de la barre d'outils ou du menu Éditer.

1.7.2 Style d'un objet

Chaque objet géométrique a ses propres attributs de style comme la couleur, l'épaisseur, l'étiquette, la taille et la forme. De plus, il est possible de cacher temporairement un objet sans le supprimer. Par exemple, pour cacher des constructions intermédiaires sans les supprimer. Tous ces attributs s'ajustent depuis le panneau latéral qui apparaît lorsque l'utilisateur choisit un objet de la figure. Pour cela il faut d'abord se mettre dans le mode d'édition de style en cliquant sur le bouton dédié de la barre d'outils.

Style de point

Le panneau latéral de style d'un point concerne tous les types de points. Il est possible d'ajuster la couleur, la forme, la taille, le nom et la visibilité. Pour les points libres dans le plan ou sur une ligne, une option de verrouillage à leur emplacement est proposée. Enfin l'option commentaire désactive l'affichage de sa bulle d'information lors de son survol par le curseur souris.

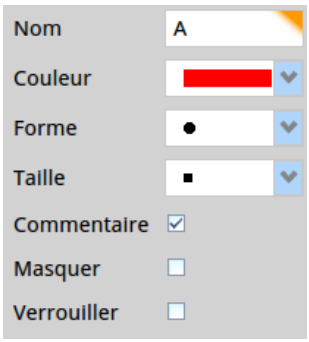

Figure 1.2: Panneau d'édition de style d'un point

Style de ligne

Le panneau latéral de style d'une ligne concerne les droites, les demi-droites, les segments, les vecteurs, les cercles, les arcs de cercle, les lieux de points et les polygones. Il est possible d'ajuster la couleur, l'épaisseur, le style de trait, d'adjoindre des flèches pour les segments et d'éditer le nom et la visibilité. Certaines propriétés ne s'appliquent que pour des types précis de lignes, comme les marques et flèches sur segments.

Par ailleurs, pour les segments, des options supplémentaires "Flèche" et "Marque" offre la possibilité d'adjoindre des flèches et des marques sur les segments.

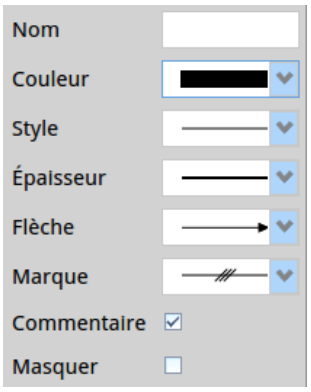

Figure 1.3: Panneau d'édition de style d'une ligne

Style de valeur

Le panneau latéral de style d'une valeur concerne toutes les sortes de valeurs : saisie par l'utilisateur ou correspondant à une mesure. Il permet de modifier la couleur d'affichage, d'éditer le nom et la visibilité de la valeur. Une option de verrouillage à l'emplacement est également proposée.

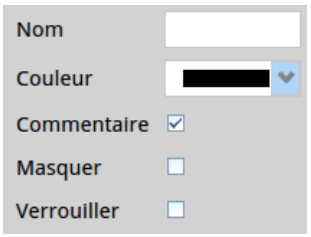

Figure 1.4: Panneau d'édition de style d'une valeur

Style de texte et script

Le panneau latéral de style de texte et de script permet d'ajuster, entre autres choses, la couleur, l'arrière-plan, la bordure, la taille

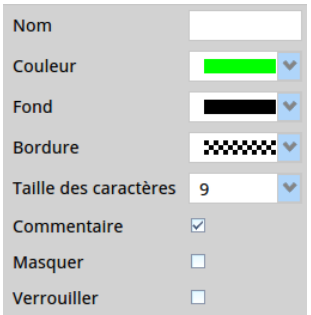

Figure 1.5: Panneau d'édition de style d'un texte ou script

1.7.3 Propriété d'un objet

Les propriétés de certains objets sont modifiables numériquement. Sont concernés les points libres dans le plan ou sur une courbe, les valeurs numériques libres et les scripts. Pour ce faire, après avoir sélectionné cet outil, choisir un de ces objets ; une boîte de dialogue s'affichera alors :

Point libre

En choisissant un point libre, une boîte de dialogue permet de modifier son abscisse et son ordonnée.

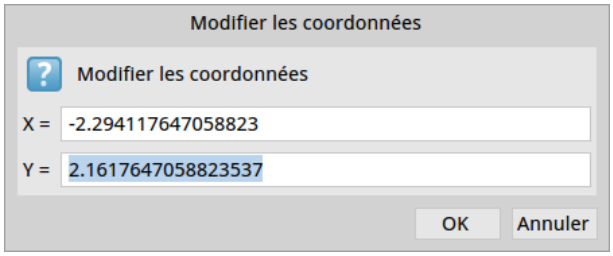

Figure 1.6: Éditer les coordonnées d'un point libre

Point libre sur une courbe

En choisissant un point libre sur une courbe, une boîte de dialogue permet de modifier son abscisse curviligne. Cette dernière est normalisée sur l'intervalle [0 ; 1].

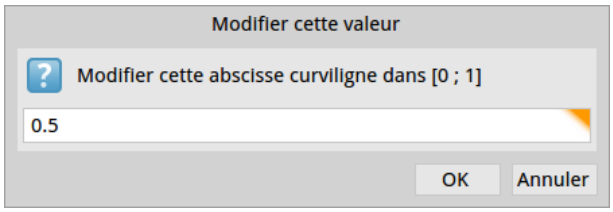

Figure 1.7: Éditer l'abscisse curviligne d'un point libre sur une ligne

Valeur libre

En choisissant une valeur libre, une boîte de dialogue permet d'éditer sa valeur.

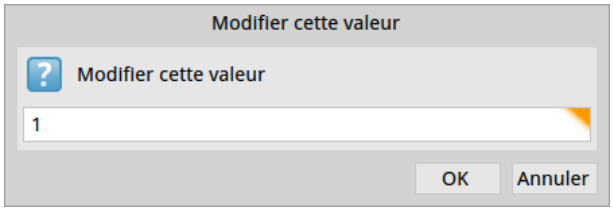

Figure 1.8: Éditer une valeur libre

Script

En choisissant un script, un navigateur de code permet de l'étudier et de le modifier. Pour sauvegarder toute modification apportée au script, utiliser la combinaison de touche `Ctrl-s` ou l'entrée *Do-it* dans le menu contextuel au-dessus du script (clic droit de la souris pour l'afficher).

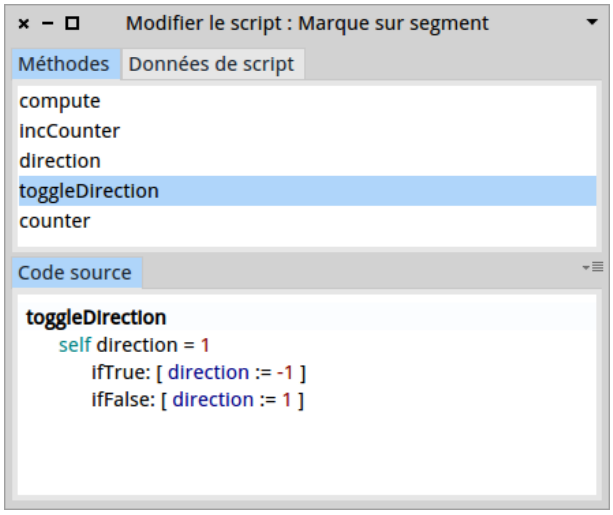

Figure 1.9: Éditer un script

Texte

En choisissant un texte, une boîte de dialogue permet de l'éditer. Dans celle-ci, le texte peut être mise en forme sur plusieurs lignes à l'aide de retour chariot – touche clavier `Entrée`.

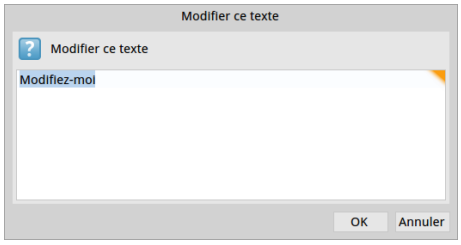

Figure 1.10: Éditer un texte

1.7.4 Déplacer, zoomer la figure

La figure se déplace à l'aide des molettes dans le coin en bas à droite de la figure ou bien directement avec le bouton droit de la souris.

L'échelle de la figure se modifie à l'aide de la molette en haut à droite de la fenêtre. La molette de la souris offre cette même fonction ; presser simultanément la touche `Shift` du clavier augmente la vitesse de grossissement.

1.7.5 Déplacer un objet

Un objet est déplacé par glisser-déposer. La figure est alors redessinée en respectant ses propriétés. Quasiment tous les objets géométriques peuvent être déplacés. Si nécessaire, Dr.Geo déplace les points libres associés. Par exemple, quand l'utilisateur déplace une droite définie par deux points, Dr.Geo déplace les deux points simultanément.

Muter un point

Dans ce mode, il est également possbile de changer la nature d'un point d'une des natures suivantes :

- point libre dans le plan
- point libre sur une ligne
- point d'intersection

vers un point d'une des natures suivantes :

- point libre dans le plan
- point libre sur une ligne
- point d'intersection

Par exemple transformer un point libre dans le plan en un point d'intersection. Il existe toutefois une contrainte : il n'est pas possible de muter un point vers une ligne (libre ou intersection) plus "jeune" que le point. Plus jeune signifie que la ligne a été créée après le point.

La touche *Shift* enfoncée en même temps qu'un point est attrapé et déplacé indique toujours par des bulles d'information les points qui peuvent être mutés et les destinations (plan, ligne, intersection de lignes) possibles.

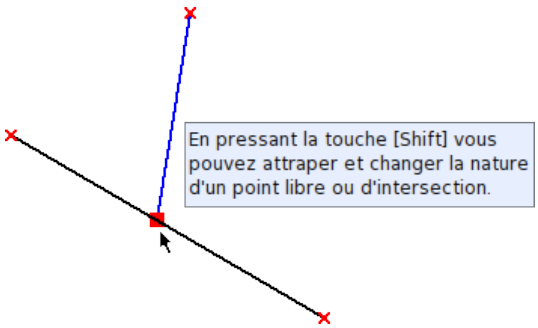

Figure 1.11: Appuyer sur *Shift* pour muter un point

Fusion de points

Dans ce mode, deux points libres peuvent être fusionés : pour cela, déplacer un point au dessus du deuxième point destination et attendre quelques secondes avec le bouton souris toujours appuyé, les points sont fusionnés.

Cloner des lignes

Différents types de lignes peuvent être clonées : pour cela presser et garder appuyé le bouton de la souris sur la ligne à cloner, une ligne clonée s'affiche alors prête à être positionnée à l'emplacement souhaité par l'utilisateur.

1.7.6 Grille et axes

Il est possible d'afficher ou cacher une grille dans toute figure de Dr.Geo, la commande est accessible depuis le bouton  de la boîte d'outils. La grille s'ajuste selon l'échelle

de la figure. Le menu ...Édition → Axes... donne lui accès aux axes des abscisses et des ordonnées. Enfin, lors de la sauvegarde d'une figure, l'état de la grille est également sauvegardé (affichée ou non affichée).

Lors de la création ou du déplacement de points, il est possible de coller ceux-ci à la grille – ou au mieux selon la contrainte du point. Pour cela il suffit d'activer la fonction grille aimantée avec le bouton  de la boîte d'outils.

2 Préférences

Ce chapitre décrit le système des préférences de l'environnement Dr.Geo. Il permet de régler les styles par défaut des objets d'une figure, l'aspect de l'interface graphique et quelques préférences réseaux.

La navigateur de préférences s'ouvre par ...Clic arrière-plan → Préférences...

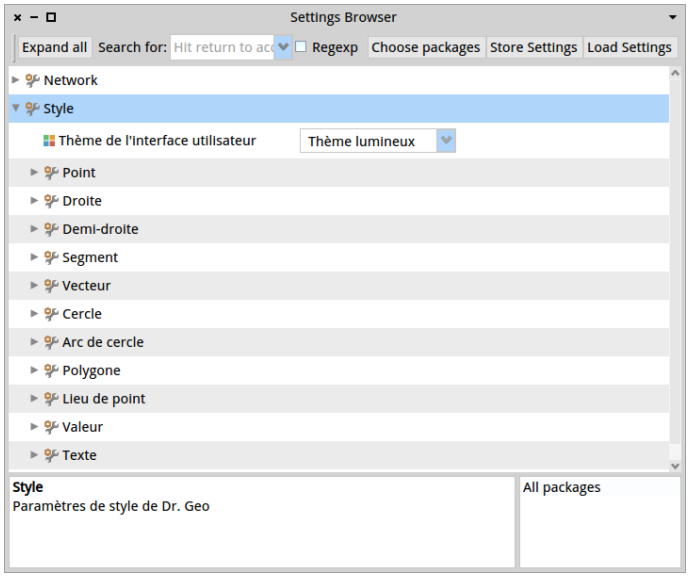

Figure 2.1: Le navigateur de préférences avec la liste des styles par défaut dépliée

Pour rendre ces préférences permanentes – lors du prochain lancement de Dr.Geo elles seront toujours effectives – il faut enregistrer la session Dr.Geo ...Clic arrière-plan → Outils → Enregistrer la session...

2.1 Thèmes graphiques

Dr.Geo propose deux thèmes graphiques pour l'interface utilisateur :

- **Thème sombre**. Pour une interface utilisateur sombre, plus adapté à un écran d'ordinateur
- **Thème lumineux**. Pour une interface utilisateur lumineuse, plus adapté à un écran de vidéoprojecteur.

2.2 Styles d'objets

Depuis une figure géométrique, le style des objets est réglable par l'utilisateur : la couleur d'une ligne, la taille d'un point, l'épaisseur d'un segment, etc. Nombres de ces paramètres peuvent recevoir une valeur par défaut : par exemple que tout point soit une petite croix noir.

2.3 Réseau

Dr.Geo propose deux fonctions réseau : sauvegarder/charger des figures sur un serveur FTP (File Transfer Protocol), partager des figures dans un réseau local. Ces options sont rangées dans la liste "Network" du navigateur de préférences.

2.3.1 Utiliser des ressources réseau

Dans le navigateur de préférences, activer l'option **Utiliser des ressources réseau**, elle donne alors accès aux réglages pour se connecter à un serveur de partage de figures, en lecture et en écriture :

- **Type de serveur.** L'interface propose ftp et http, mais seul le premier est fonctionnel pour l'instant.
- **Hôte.** C'est le nom du serveur ou une IP, par exemple `ftp.drgeo.eu`.
- **Nom d'utilisateur.** C'est le login de connexion au serveur.
- **Mot de passe.** Le mot de passe de l'utilisateur sur le serveur.
- **Partage.** C'est le nom du partage sur le serveur où lire et enregistrer les figures, concrètement c'est un dossier à la racine du partage du serveur.

Une fois ces éléments configurés, la fenêtre d'ouverture d'une figure propose un bouton "Réseau" avec le partage visible et modifiable. Il suffit de cliquer sur ce bouton pour charger la liste des figures en partage. Pour le constater, faire ...Clic arrière-plan → Ouvrir une figure → Réseau...

De même la fenêtre pour sauver une figure propose une option réseau où enregistrer. Pour le constater, depuis n'importe quelle fenêtre Dr.Geo, faire Fichier → Enregistrer ou Enregistrer sous → Partage réseau...

2.3.2 Partage de réseau local

Plus souple, le partage dans un réseau local ne nécessite pas de serveur dédié. Au lieu de cela, un poste maître avec Dr.Geo en fonctionnement fait office de serveur de figures.

Dans le navigateur de préférences, activer l'option **Partage de réseau local**. Une fois fait, le Dr.Geo en cours de fonctionnement partage alors toutes ses figures situées dans le dossier `DrGeo/MyShares`.

Depuis les autres postes, élèves par exemple, à l'ouverture d'une figure cliquer sur le bouton "Partage enseignant" pour parcourir la liste des figures partagées par le poste maître. Pour le constater, faire ...Clic arrière-plan → Ouvrir une figure → Partage enseignant...

Sur le poste maître il faudra avant tout copier les figures à partager dans le dossier `DrGeo/MyShares`. Cela se fait soit depuis le gestionnaire de fichiers de l'ordinateur, en copiant les fichiers `.fgeo` et `.png` associés dans le dossier `DrGeo/MyShares`. Soit depuis Dr.Geo lui-même en enregistrant la figure dans le dossier `DrGeo/MyShares` par Fichier → Enregistrer à → MyShares → nom de figure... A répéter pour chacune des figures à partager.

3 Fichiers et documents

Les constructions peuvent être enregistrées de deux manières. Une construction par fichier ou un ensemble de constructions et de macro-constructions dans un seul fichier (c.-à-d. une session Dr.Geo). Nous vous rappelons que les documents sont sauvegardés avec l'extension `.fgeo`.

3.1 Renommer une figure

Depuis le menu ...Fichier → Renommer..., l'utilisateur change le nom et le titre de la fenêtre de la figure active. Cette fonction est particulièrement utile lorsque l'utilisateur sauvegarde un ensemble de figures dans un unique fichier de session.

3.2 Enregistrer

À partir du menu ...Fichier → Enregistrer..., la figure de la vue active est enregistrée dans un fichier.

Dr.Geo sait travailler avec plusieurs figures en même temps, chaque figure ayant sa propre fenêtre. L'utilisateur passe d'une figure à l'autre en utilisant la barre des tâches en bas de l'environnement Dr.Geo.

Avec le menu ...Fichier → Enregistrer sous..., l'utilisateur sauvegarde le document sous un autre nom.

Les documents sont habituellement sauvegardés dans la structure de l'application Dr.Geo, dans le dossier `DrGeo/MySketches`. Pour sauver dans un emplacement arbitraire du disque dur, utiliser le menu ...Fichier → Enregistrer à....

Cela est utile lorsque l'enseignant souhaite partager des fichiers avec ses étudiants dans un réseau local. Il enregistre les figures dans le dossier `DrGeo/MyShares` de Dr.Geo, puis active le partage réseau local depuis ...Clic arrière-plan → Préférences → Réseau → Partage de réseau local... Les étudiants accède aux figures partagées en choisissant **Partage enseignant** lors de l'ouverture d'une figure.

Lors de la sauvegarde d'un document, une option réseau est proposée dans la boîte de dialogue[1] Lors de la sélection de cette option, il faut également donner un nom de partage réseau – du choix de l'utilisateur – en plus du nom du document. Cette option permet de partager des documents entre plusieurs utilisateurs connectés en réseau.

Enregistrement d'une session

Une session est un ensemble de données de Dr.Geo que l'utilisateur enregistre d'un seul coup dans un fichier. Cela permet de placer un ensemble de données (figures et macro-constructions) dans un seul fichier, afin de faciliter leurs réutilisations.

À partir du menu ...Fichier → Enregistrement multiple..., l'utilisateur accède à la boîte de dialogue de sauvegarde d'une session.

[1] Seul le partage sur un serveur ftp fonctionne pour l'instant, il se configure depuis ...Clic arrière-plan → Préférences → Réseau → Utiliser des ressources réseau...

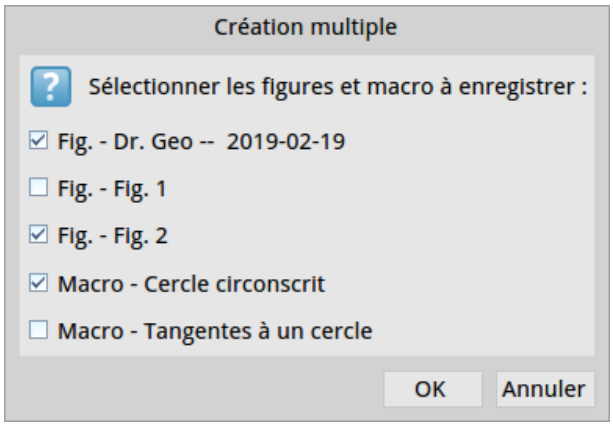

Figure 3.1: La boîte de dialogue de session Dr.Geo

Dans cette boîte de dialogue, la liste de toutes les données actives est présentée dans une colonne. Chaque ligne est préfixée du tag **Fig.** ou **Macro** selon son type, la seconde partie de la ligne contient le nom de la donnée.

Actuellement, une session peut contenir deux types de données : figure interactive 2D et macro-construction.

L'utilisateur sélectionne les données à enregistrer en cochant leur case puis valide avec le bouton "OK".

Le menu ...Fichier → Enregistrement multiple... est le seul moyen d'enregistrer une macro-construction dans un fichier.

Enregistrer une macro-construction

Pour enregistrer une ou des macro-constructions dans un fichier, procéder comme pour l'enregistrement d'une session – enregistrement multiple. Depuis la boîte de dialogue de sauvegarde d'une session, sélectionner la ou les macro-constructions à sauvegarder, puis donner le nom du fichier. C'est tout !

Il est ainsi possible de constituer des librairies de macro-contructions, une par fichier ou plusieurs regroupées selon un même thème dans un seul fichier.

3.3 Exporter une figure

Le menu ...Fichier → Exporter image... exporte la figure de la fenêtre active dans un fichier image au format standard PNG[2]. Les images sont sauvées dans la structure de l'application Dr.Geo, dans le dossier `DrGeo.app/MyExports`.

3.4 Ouvrir un fichier

Que l'utilisateur ait sauvegardé une seule figure ou une session avec différents types de documents, la procédure pour l'ouverture est la même par le menu ...Fichier → Ou-

[2] `http://www.w3.org/Graphics/PNG`

vrir... Si la session ouverte contient des macro-constructions, celles-ci sont directement disponibles depuis l'outil pour jouer des macro-constructions. Les macro-constructions sont alors disponibles depuis toutes les fenêtres ouvertes de Dr.Geo, actuelles ou futures.

Partie II
Fonctionnalités avancées

4 Introduction

Dans cette partie, nous présentons les fonctionnalités qui étendent les possibilités de Dr.Geo pour l'adapter à un usage, à une situation pédagogique donnée ou bien pour produire une démonstration interactive.

La première est la *macro-construction* qui extrait une portion de construction pour la placer dans un enregistrement. Cet enregistrement est ensuite exécutable autant de fois que souhaité, sauvegardé dans un fichier et ouvert ultérieurement dans une autre figure.

Les *scripts Pharo* représentent une autre fonctionnalité pour étendre Dr.Geo. Ce sont des items d'une figure au même titre que les items géométriques. Ils sont paramétrés par zéro ou plus arguments, références d'items géométriques de la figure. Le code du script effectue un traitement et retourne un objet placé dans la figure. Un script peut également produire un effet de bord en modifiant les attributs d'autres items de la figure.

Un script est une instance d'un objet[1] greffée dans une figure, il est évalué à chaque mise à jour de la figure (c.-à-d. lorsque la figure a besoin d'être reconstruite). Les scripts Pharo sont utiles pour la valeur qu'ils retournent ou leur effet de bord, cela dépend de ce que l'utilisateur souhaite réaliser.

Enfin, dans le prolongement des scripts, Dr.Geo propose de définir entièrement des figures avec du code, ce sont les *figures Pharo*. Cette fois il s'agit de décrire une figure géométrique complètement sous la forme d'un code source écrit dans le langage Pharo. L'intérêt de cette approche est de s'appuyer sur les possibilités offertes par un langage de programmation comme les boucles, les blocs de code[2], les branchements conditionnels, la récursivité, etc. pour construire une figure interactive. Ainsi la construction par le code d'une figure n'est plus uniquement déclarative comme c'est le cas avec l'interface graphique. Pour ce faire le code s'appuie sur une API[3] pour programmer l'équivalent des constructions à la souris et bien plus encore.

[1] Classe dans laquelle une ou plusieurs méthodes sont définies par l'utilisateur pour les besoins du script.

[2] Aussi connu sous le nom de fonction anonyme.

[3] API (Application Programming Interface), Interface de Programmation Applicative

5 Macro-construction

Une macro-construction ressemble un peu à une procédure qui reçoit des items d'une figure en entrée et qui retourne un ou plusieurs items de figure, construits par la macro-construction. Une macro est construite à partir d'un modèle défini par l'utilisateur. Cela signifie que l'utilisateur doit réaliser la séquence de construction une première fois dans une figure puis demander à Dr.Geo de l'enregistrer dans une macro-construction. La macro-construction peut ensuite être sauvegardée dans un fichier au même titre qu'une figure.

Pour enregistrer une séquence de construction, Dr.Geo doit connaître les items initiaux de la séquence ainsi que les items finaux. Évidemment, les items finaux ne doivent dépendre *que* des items initiaux[1], sinon Dr.Geo ne sera pas capable de déduire les items finaux à partir des items initiaux.

Ainsi, Dr.Geo déduit la logique de la séquence de construction et l'enregistre dans une macro-construction. L'utilisateur peut alors répéter cette séquence en jouant la macro-construction, elle demande seulement des items initiaux (du même type) de la figure et construit les items résultants.

Les items de figure intermédiaires et invisibles sont aussi construits par la macro-construction. Ils sont nécessaires pour construire les items résultants.

Pour illustrer la fonctionnalité macro-construction, nous prenons l'exemple de la construction du cercle passant par trois points.

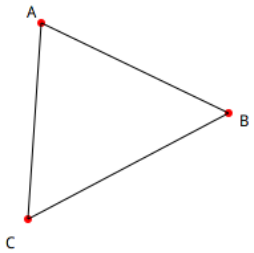

Figure 5.1: La figure initiale

Avant la création de la macro-construction, l'utilisateur doit construire la figure finale, elle est considérée comme modèle par la macro-construction.

[1] Cette contrainte a depuis été assouplie et permet d'aller encore plus loin avec les macro-constructions.

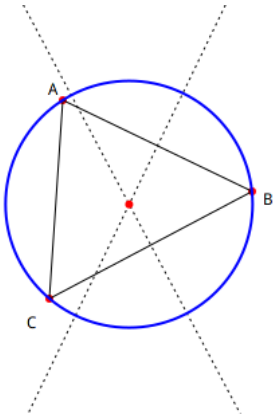

Figure 5.2: La figure avec la construction finale

5.1 Créer une macro construction

À cette étape, la séquence de construction est faite. L'utilisateur doit maintenant avertir Dr.Geo qu'il veut une macro-construction à partir de cette séquence. Il doit appeler la fonction `Construire une macro` à partir de la barre d'outils ou depuis le menu ...Macro-construction → Construire une macro... de la fenêtre de la figure.

Depuis la boîte de dialogue qui s'affiche alors, l'utilisateur sélectionne les paramètres d'entrée et de sortie, le nom et la description de la macro-construction.

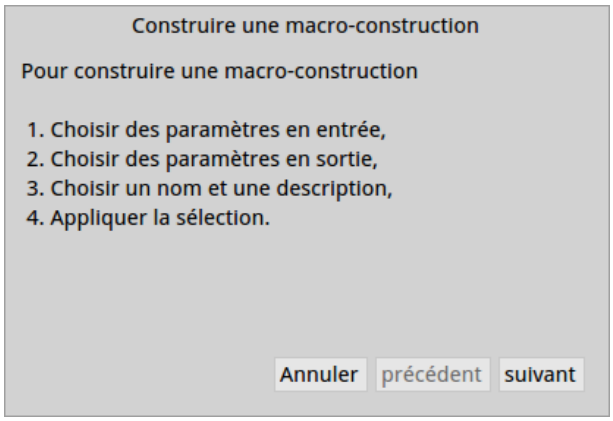

Figure 5.3: Première page introductive de la boîte de dialogue de l'assistant pour construire une macro-construction

La seconde page de la boîte de dialogue sert à sélectionner les paramètres d'entrée. Dans notre exemple, ce sont les trois points initiaux. L'utilisateur a juste besoin d'aller jusqu'à cette seconde page et il sélectionne les trois points A, B et C de la figure. Les points sélectionnés clignotent alors et leur nom s'affiche dans la page de la boîte de dialogue.

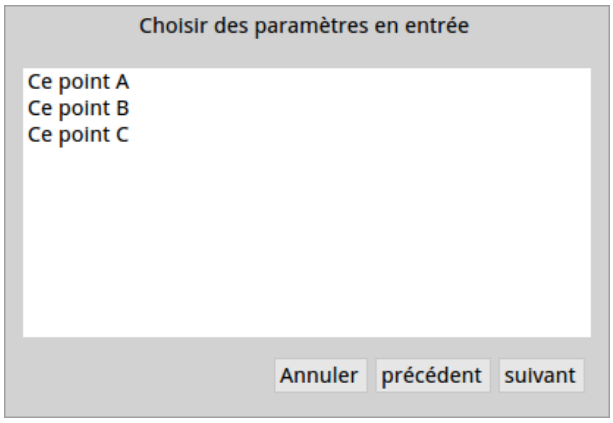

Figure 5.4: La seconde page : les trois points sont sélectionnés

À partir de la troisième page, l'utilisateur choisit les paramètres de sortie. Dans notre exemple, nous voulons le cercle et son centre comme résultat de la macro-construction. L'utilisateur clique alors sur ces deux objets dans la figure, lorsque sélectionnés ces derniers clignotent et leur nom apparaît dans la boite de dialogue.

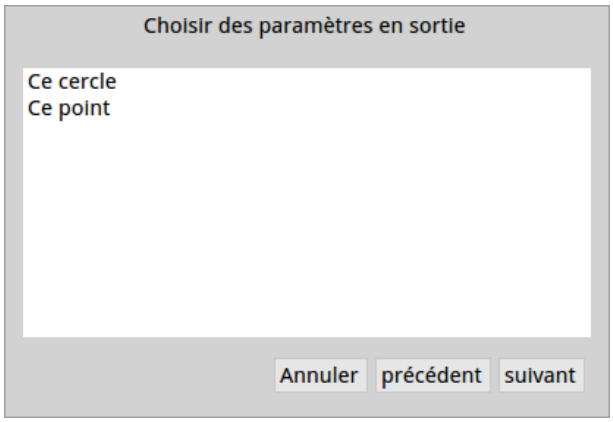

Figure 5.5: La troisième page : le cercle et son centre sont sélectionnés

Dans la quatrième page, l'utilisateur entre le nom et la description de la macro-construction. Ces informations sont affichées lorsque l'utilisateur exécute une macro-construction. Ceci permet de distinguer les macro-constructions entre elles.

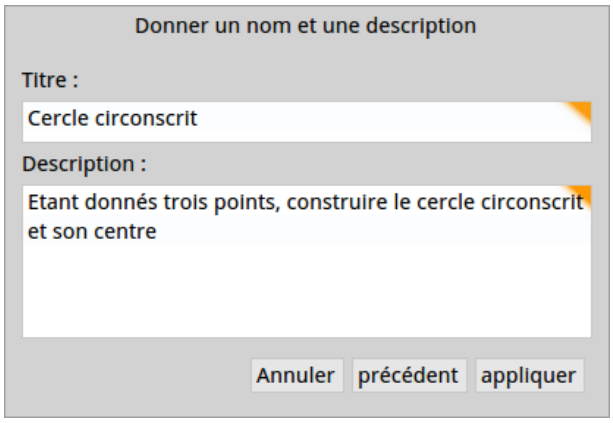

Figure 5.6: La quatrième page : le nom et la description de la macro-construction

L'utilisateur valide ensuite la définition de la macro-construction en appuyant sur le bouton `Appliquer`. Il peut aussi revenir aux étapes précédentes pour ajuster les paramètres de la macro-construction.

Si la sélection des paramètres d'entrée et de sortie ne correspond pas (Dr.Geo ne peut pas extraire la logique de la construction), la macro-construction ne peut pas être créée. Dans ce cas, l'utilisateur doit reconsidérer la sélection des paramètres d'entrée et de sortie. Il peut revenir à la seconde ou la troisième page de la boîte de dialogue de l'assistant pour ajuster ses choix.

À cette étape, la macro-construction est créée et enregistrée dans Dr.Geo. Dans la prochaine section, nous verrons comment l'utiliser.

5.2 Jouer une macro construction

À l'aide de la boîte de dialogue

Pour exécuter une macro-construction, l'utilisateur choisit dans la barre d'outils ou le menu Macro-construction → Jouer une macro de la fenêtre de la figure. Une boîte de dialogue décrivant la procédure s'affiche alors.

À partir de celle-ci, dans la seconde page, il sélectionne la macro-construction dans la liste en haut de la boîte de dialogue. Une fois la macro sélectionnée, il peut directement choisir les paramètres d'entrée dans la figure – avant cliquer sur le fond de la figure afin que les bulles d'aide s'affichent lors du survole des objets. Dès que tous les paramètres d'entrée sont sélectionnés, la macro-construction est exécutée et les paramètres finaux sont construits.

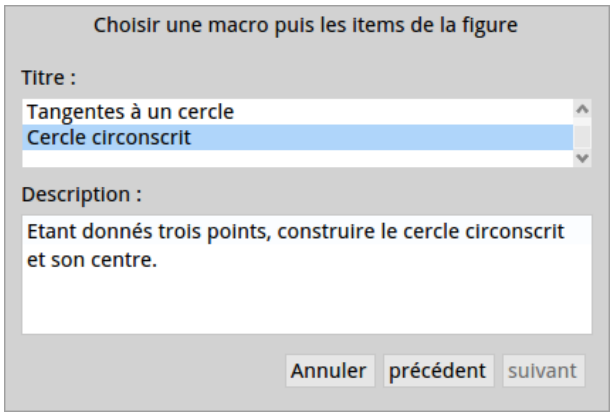

Figure 5.7: L'utilisateur sélectionne les paramètres d'entrée dans la figure

Dans notre exemple, la macro-construction nécessite trois paramètres d'entrée (trois points) et elle construit un point et un cercle. Pour exécuter notre macro-construction, il faut une figure avec au moins trois points.

Figure 5.8: Une figure avec trois points

Une fois que notre macro-construction est appliquée à ces trois points, nous avons le cercle souhaité et son centre.

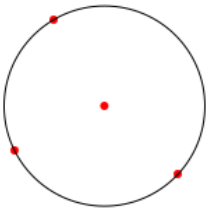

Figure 5.9: La figure finale avec le cercle et son centre

À l'aide du menu macro-construction

Les macro-constructions chargées en mémoire sont listées dans le menu de fenêtre "Macro-construction". En sélectionnant une de ces macro-constructions dans le menu, il est alors possible de choisir directement les paramètres d'entrées dans la figure pour exécuter la macro-construction.

C'est un mode d'exécution d'une macro-construction sans boîte de dialogue. Par ailleurs, en faisant une pause sur un des items de ce menu, la description détaillée de la macro-construction est affichée.

6 Script Pharo Dr.Geo

Dr.Geo est une application dynamique écrite en Pharo. Cela signifie qu'il est possible de le modifier alors qu'il est en cours de fonctionnement. Nous exploitons cette possibilité pour définir dans Dr.Geo des items de la figure qui sont en fait des scripts Pharo – des bouts de codes, Cela permet d'étendre dynamiquement, à l'infini, les possibilités de Dr.Geo. Mais qu'est-ce que Pharo ?

> Pharo est un langage de programmation orienté objet, réflexif et dynamiquement typé. Il est une implémenation et extension de Smalltalk. Ce dernier fut l'un des premiers langages de programmation à disposer d'un environnement de développement intégré complètement graphique. Il a été créé en 1972. Il est inspiré par Lisp et Simula. Il a été conçu par Alan Kay, Dan Ingals, Ted Kaehler, Adele Goldberg au Palo Alto Research Center de Xerox. Le langage a été formalisé en tant que Smalltalk-80 et est depuis utilisé par un grand nombre de personnes. Smalltalk est toujours activement développé.
>
> Smalltalk a été d'une grande influence dans le développement de nombreux langages de programmation, dont : Objective-C, Actor, Java et Ruby.
>
> Un grand nombre des innovations de l'ingénierie logicielle des années 1990 viennent de la communauté des programmeurs Smalltalk, tels que les Design Patterns (appliquées au logiciel), l'Extreme Programming (XP) et le refactoring. Ward Cunningham, l'inventeur du concept du Wiki, est également un programmeur Smalltalk.
>
> —*Wikipedia, Smalltalk, 2 janvier 2011*

Cet extrait de la préface du livre *Pharo By Example* décrit précisément la plate forme Pharo utilisée pour Dr.Geo :

> Pharo est une implémentation moderne, libre et complète de l'environnement et langage de programmation Smalltalk.
>
> Pharo s'attache à offrir une plate forme robuste et stable pour du développement professionnel en langages et environnement dynamiques.
>
> —*Pharo By Example, introduction*

Dr.Geo exploite l'environnement Pharo pour proposer, d'une part, un environnement convivial d'écriture de scripts et, d'autre part, pour donner accès à l'interface des items géométriques ou numériques constitutifs d'une figure. L'interface est en fait l'ensemble des méthodes d'instance – fonctions de classe – de ces items.

Ainsi l'utilisateur peut écrire des scripts pour manipuler les items des figures ; et puisque ces scripts sont des items de figure au même titre que d'autres, ils n'ont pas besoin d'être dans un fichier séparé, ils sont enregistrés dans le fichier de la figure.

L'autre grande force des scripts est de s'appuyer sur l'environnement de développement de Pharo ; l'utilisateur bénéficie ainsi d'outils évolués pour mettre au point ces scripts : navigateur de classes, inspecteur, débogueur, etc. L'utilisateur souhaitant exploiter au mieux la puissance des scripts est donc invité à étudier le livre *Pharo By Example*, il y apprendra le langage Pharo et son environnement.

Les scripts ont deux facettes :

- l'écriture du script lui même ;
- l'utilisation du script dans la figure, un même script est utilisable plusieurs fois, avec des paramètres différents.

Les outils pour créer et éditer un script sont disponibles depuis les menus ...Script → Créer un script... et ...Script → Modifier un script... Ces fonctions se trouvent également

dans la barre d'outils. Pour utiliser un script choisir le menu ...Script → Utiliser un script... également disponible dans la barre d'outils.

Un script est un objet de première classe[1]. Il est défini comme une classe Pharo et une instance de celle-ci est créée pour chacune de ses utilisations dans une figure géométrique.

Lors de la sauvegarde d'une figure comprenant une ou plusieurs instances de scripts, le code source de leur classe est également enregistré ; pour s'en convaincre ouvrir avec un éditeur de texte un fichier `.fgeo` d'une figure utilisant un script.

Dans son utilisation, un script peut recevoir de 0 à n paramètres d'entrée. Après le choix du script à insérer dans la figure, il suffit alors de cliquer sur les objets de type paramètres d'entrée – comme précisé lors de la création du script – puis sur le fond de la figure pour y placer le résultat du script.

Par la suite nous vous proposons de travailler sur quelques exemples de scripts pour comprendre leur fonctionnement. Les scripts, comme les macro-constructions, donnent une dimension particulière à Dr.Geo, ils permettent – chacun avec un positionnement différent[2] – d'aller là où les auteurs du logiciel ne sont pas allés ou ne souhaitent pas aller.

Il est aussi important de comprendre que l'ensemble des fonctionnalités de Pharo sont disponibles depuis les scripts. C'est particulièrement vrai pour ses bibliothèques de fonctions[3], nous allons bien sûr les utiliser intensément.

6.1 Script sans paramètre

Principe de construction d'un script

Lors de la construction d'un script, depuis ...Script → Créer un script... trois éléments sont demandés pour le définir :

1. son nom ;
2. sa description ;
3. les types de paramètres d'entrée.

Le troisième point indique l'ordre et les objets sur lesquels cliquer (point, segment, etc.) pour ensuite créer une instance du script dans la figure.

Premier exemple

La procédure pour créer et utiliser un script sans paramètre d'entrée est la suivante :

Édition du script.

1. Choisir ...Script → Créer un script... le constructeur de script s'affiche alors – Voir [scriptConstructor], page 100.

[1] `https://fr.wikipedia.org/wiki/Objet_de_première_classe`

[2] Les macro-constructions ont une approche géométrique tandis que les scripts ont une approche numérique mais aussi et surtout nous pouvons les utiliser dans un esprit de bidouillage ("hacking" en anglais).

[3] En particulier, les fonctions mathématiques.

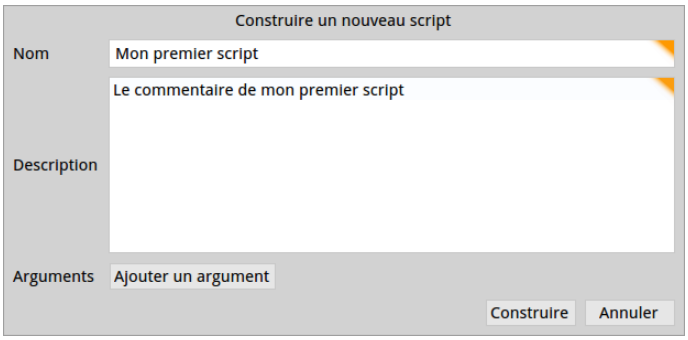

Figure 6.1: Le constructeur de script

Il comprend trois parties :

- **Nom**. Son nom, saisir `Mon premier script`. Dans le menu "Script", ce script est alors affiché avec ce nom, le choisir donc avec soin. Il est modifiable ultérieurement.
- **Description**. Sa description, saisir `Le commentaire de mon premier script`. Dans le menu "Script", ce script a une bulle d'information avec ce texte. Inscrire donc ici l'ensemble des détails nécessaires à son utilisation : ce qu'il fait et les types d'objets sur lesquels cliquer pour le faire fonctionner.
- **Arguments**. Les types d'objets utilisés en paramètres d'entrée, ceux sur lesquels cliquer pour activer le script. Dans ce premier exemple, nous n'ajoutons pas d'argument car nous créons un script sans paramètre d'entrée.

2. Presser le bouton `Construire`, Dr.Geo construit alors une classe pour ce script et affiche un éditeur de script sur celle-ci – Voir [scriptEditor], page 36.

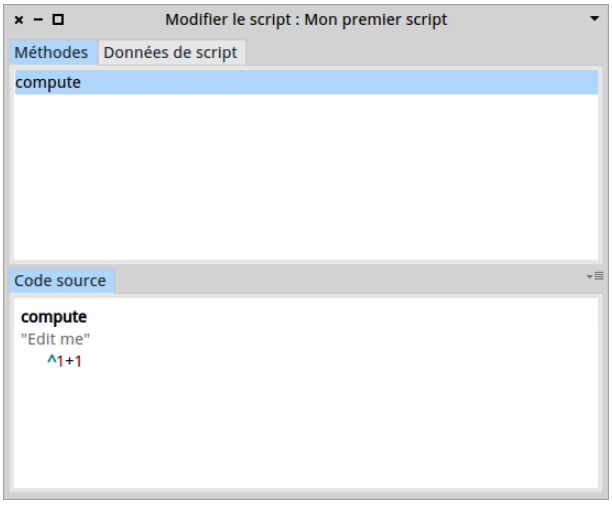

Figure 6.2: Editeur de script sur *mon premier script*

De la gauche vers la droite et du haut vers le bas :

- L'onglet "Méthodes" présente les méthodes du code source du script.
- L'onglet "Données de script" avec les méthodes de définition du script : titre, description, arguments du script.
- La liste des méthodes du code source ou de définition selon l'onglet sélectionné. En cliquant sur une méthode, son code source est affiché dans l'éditeur de texte en

bas. **La méthode `compute` est créée par défaut par Dr.Geo. C'est cette méthode qui retourne le résultat à afficher dans la figure.** Attention à ne pas la supprimer cela provoquerait une erreur, il est toutefois possible de la recréer le cas échéant.

- Un éditeur de code source pour éditer une méthode. Pour valider une modification, il suffit de presser les touches *`Ctrl-s`*.

Saisir le code source ci-dessous dans le panneau du bas de l'éditeur de script :

```
compute
    "Je dis bonjour"
    ^ 'hello !'.
```

Sauvegarder le script en cliquant sur le bouton "Enregistrer" à droite de l'onglet "Code source" ou par la combinaison de touches *`Ctrl-s`*.

La première ligne d'une méthode, ici `compute`, désigne toujours le nom de la méthode, la suite est le code source du script. La deuxième ligne de celle-ci, entre guillemets, est un commentaire de ce que fait la méthode, nous conseillons de bien documenter losrque vous créer plusieurs méthodes dans un script.

Pour créer une nouvelle méthode, il suffit de changer le nom de la méthode affichée et de sauver avec le bouton "Enregistrer" à droite.

L'éditeur de script peut maintenant être fermé.

Utilisation du script dans la figure.

Choisir ...Script → Utiliser un script... dans le menu. Dans la boîte de dialogue qui s'affiche alors choisir le script `Mon premier script` que nous avons créé précédemment. Noter qu'à chaque fois qu'un script est choisi, son commentaire descriptif est affiché en dessous.

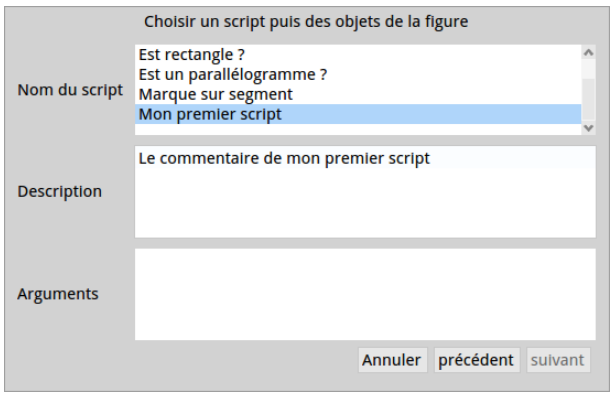

Figure 6.3: Choisir un script

Une fois le script sélectionné, cliquer dans la figure à l'emplacement souhaité, le script y est inséré ; dans cet exemple il ne fait que retourner le message "hello !". La valeur retournée par la méthode `compute` est toujours celle affichée dans la figure.

Dans les exemples ci-dessous, nous donnons uniquement le code source de la méthode `compute`. En reprenant les étapes précédentes, à vous de créer le script en choisissant à votre convenance ses nom et description, nous ne revenons pas sur ces étapes.

Un générateur de nombres aléatoires et autres

Si vous souhaitez un générateur de nombres entiers aléatoires entre 0 et 10, rien de plus simple, c'est ce que fait le script suivant :

```
compute
```

```
"Je retourne un nombre aléatoire"
   ^ 10 atRandom.
```

À chaque mise à jour de la figure, il génère un nombre aléatoire entier dans l'intervalle [0 ; 10]. Si vous préférez un nombre flottant dans l'intervalle [0 ; 1], utilisez ce script :

```
compute
"Je retourne un nombre décimal aléatoire entre 0 et 1"
   ^ 100 atRandom / 100.
```

Quelques précisions :

- La valeur retournée par le script est le résultat de l'expression après le symbole `^` ;
- La valeur retournée peut être de n'importe quel type[4], Dr.Geo en affiche une représentation sous forme de chaîne de caractères ;
- Si l'on souhaite retourner la valeur d'une variable, il suffit de mettre son nom après le symbole `^` .

Calculer des valeurs usuelles

Pour afficher une valeur approchée de **pi** ou de **e** :

```
compute
   ^ Float pi
compute
   ^ Float e
```

Les valeurs retournées par ces scripts sont ensuite utilisables comme toutes les autres valeurs numériques générées par Dr.Geo. Pour toutes ces petites choses les scripts sont donc vos amis. Mais ils peuvent faire bien plus de choses intéressantes lorsqu'ils reçoivent des paramètres en entrée.

En effet ici les scripts n'avaient aucun argument, il n'y avait donc pas lieu de sélectionner des items de la figure lors de l'insertion des scripts dans la construction. Bien sûr leur intérêt réside dans les traitements numériques qu'ils permettent sur des items et la restitution de ce résultat dans la figure, sous la forme d'un objet qui lui même peut-être utilisé par d'autres scripts. Dans les sections suivantes nous montrerons de tels enchaînements de scripts.

6.2 Script avec un paramètre

L'exemple

Prenons l'exemple d'un script qui étant donné un polygone va :

- Afficher un message pour indiquer si c'est un rectangle.
- Colorier en bleu le polygone lorsqu'il est un rectangle, sinon en rouge.

Objectif

Cet exemple montre comment faire des calculs à partir de l'état d'un objet – un polygone – et modifier un attribut visuel d'un objet – sa couleur. Étant donné que le langage informatique Pharo des scripts est le même que celui de Dr.Geo, nous montrons que l'étude du fonctionnement interne de Dr.Geo permet d'utiliser au mieux les scripts.

Enfin les trois constituants d'un objet mathématique sont exposés : hiérarchies des classes `DrGMathItem` pour le modèle mathématique, `DrGMathItemCostume` pour la représentation visuelle et `DrGCostumeStyle` pour gérer les attributs visuels (couleur, formes,...).

[4] Une instance de n'importe quelle classe pour être précis car les types n'existent pas en Pharo

Créer le script

La procédure pour créer un script avec un paramètre d'entrée est sensiblement la même. Toutefois un argument de type polygone est nécessaire, en effet lors de son utilisation il faut choisir un polygone dans la figure. Les paramètres du constructeur de ce script – à invoquer par le menu ...Script → Créer un script... ressembleront à la figure ci-jointe – Voir [scriptConstructorOneArg], page 39.

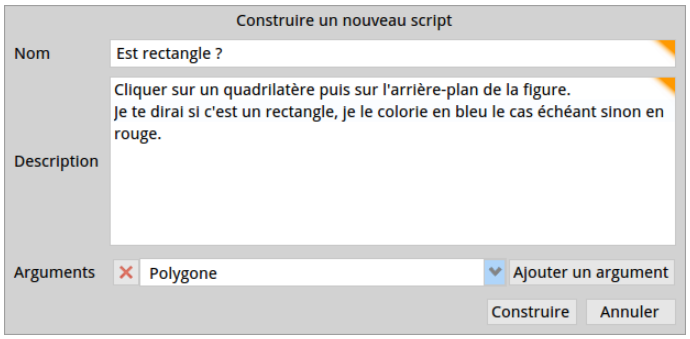

Figure 6.4: Le constructeur de script avec un argument demandé

Logique de fonctionnement

La logique de ce script est implémentée dans sa méthode `compute`. Elle est la suivante :

1. colorier en rouge le polygone
2. tester si le polygone est un quadrilatère, dans la négative retourner le message `'Je ne suis même pas un quadrilatère'`.
3. tester si ses diagonales sont isométriques et partagent le même milieu :
   1. si oui, colorier en bleu le polygone et retourner le message `'Je suis un rectangle`.
   2. si non, retourner le message `'Je ne suis PAS un rectangle'`.
4. fin de la méthode `compute`

Le code source

La méthode principale `compute` du script :

```
compute
   | diag1 diag2 points mid1 mid2|
   self paintRed.
   self isQuad ifFalse:
      [ ^ 'Je ne suis même pas un quadrilatère !' ].
   points := self arg1 points.
   diag1 := points first - points third.
   mid1 := points first + points third / 2.
   diag2 := points second - points fourth.
   mid2 := points second + points fourth / 2.
   (diag1 r = diag2 r and: [mid1 = mid2])
      ifTrue: [
         self paintBlue.
         ^ 'Je suis un rectangle :)' ]
      ifFalse: [ ^ 'Je ne suis PAS un rectangle :(' ].
```

Quelques explications :

- Les méthodes `paintRed` et `isQuad` sont invoquées sur `self`, cela signifie qu'elles sont implémentées dans la classe de ce script ou une classe parente, nous y reviendrons. Dans la terminologie Pharo, il est dit que le message `#paintRed` est envoyé à `self`.
- Le message `#arg1` retourne le premier argument du script (`self arg1`). Ici obligatoirement un polygone, à savoir une instance de la classe `DrGPolygonItem`[5]. Si notre script était défini avec deux arguments, le message `#arg2` donne le deuxième objet choisi par l'utilisateur.
- `DrGPolygonItem` a une méthode `points` qui retourne une collection de ses sommets. Nous en extrayons les informations nécessaires à nos calculs.
- Les deux vecteurs des diagonales du quadrilatère sont calculés et placés dans les variables locales `diag1` et `diag2`. Ce sont des instances de la classe standard `Point`.
- Les milieux des diagonales sont calculés et également placées dans les variables locales `mid1` et `mid2`.
- Les longueurs des diagonales et les milieux sont comparés, selon le résultat des pharases différents sont retournées. Pour obtenir la longueur d'une diagonale, le rayon en coordonnées polaire est demandé avec le message `#r`. Explorer la classe `Point` pour lire le code source de cette méthode.

Passons maintenant aux trois autres méthodes de ce script, à savoir `paintRed`, `paintBlue` et `isQuad` :

```
paintRed
   self costume1 style color: Color red
paintBlue
   self costume1 style color: Color blue
```

Le message `#costume1` demande le costume du premier argument du script, à savoir le costume du polygone. Dans Dr.Geo, le costume d'un objet permet à la fois l'accès au modèle mathématique et à sa représentation graphique ainsi que ses attributs associés de style[6]. Ces deux méthodes demandent donc le style du costume (message `#style`) pour lui envoyer ensuite le message à mot clé `#color:` avec comme argument la couleur souhaitée.

Les messages à mot clé sont une spécificité de Pharo : les arguments sont intercalés dans le nom du message.

La dernière méthode annexe est triviale, elle demande au modèle (le polygone retourné par la méthode `arg1`) sa liste des sommets, puis la taille de cette liste. Le polygone est un quadrilatère si cette taille est de 4 :

```
isQuad
   ^ self arg1 points size = 4
```

Après la saisie de chacune de ces méthodes, le code source doit être sauvé et compilé par le raccourci clavier *`Ctrl-s`*.

Pour l'utilisation du script (...Script → Utiliser un script...) Dr.Geo attend que l'utilisateur clique sur un polygone, puis sur un emplacement de la figure.

Selon le type d'objet en référence, diverses méthodes sont disponibles ; qui pour obtenir sa valeur, qui pour obtenir ses coordonnées, etc. Le répertoire des méthodes est disponible depuis la section Méthodes de référence des scripts Voir [api-dgs], page 44.

[5] Pour découvrir le protocole de cette classe, écrire son nom dans un Workspace, le sélectionner à la souris puis presser les touches *`Ctrl-b`*, un navigateur de classes s'affiche alors sur cette classe, il permet de naviguer et d'étudier son code source.

[6] Pour être précis, un script reçoit toujours par sa variable `arguments` les costumes des objets sur lesquels l'utilisateur a cliqué ; la méthode `arg1` donne alors le modèle par un appel `arguments first mathItem`, tandis que la méthode `costume1` se contente de faire `arguments first`.

6.3 Script avec deux paramètres

Pour calculer la distance entre deux points[7], nous créons alors un script avec deux arguments : deux points. L'unique méthode `compute` est alors :

```
compute
"Calcule la distance entre deux points"
   ^ self arg1 point dist: self arg2 point
```

Ici les méthodes `arg1` et `arg2` retournent des objets Dr.Geo de type point, dans la hiérarchie de classe `DrGPointItem`. Cette classe a une méthode `point` qui retourne ses coordonnées.

`#dist:` est un message à mot clé[8] de la classe `Point` qui attend comme unique argument un autre point, elle calcule la distance entre ces deux instances. Elle peut se comprendre comme : "distance entre arg1 point et arg2 point".

Pour utiliser ce script, procéder comme dans les exemples précédents : choisir deux points de la construction et un emplacement de la figure pour y placer le résultat du script.

6.4 Exemple détaillé de figure avec plusieurs scripts

Dans la section suivante, nous présentons une figure plus complexe intégrant un enchaînement de scripts pour la construction d'une portion de courbe représentative d'une fonction et la tangente en un point mobile de cette portion de courbe.

La figure finale est disponible dans le dossier `examples` de Dr.Geo, elle s'appelle `Curve and slope.fgeo`.

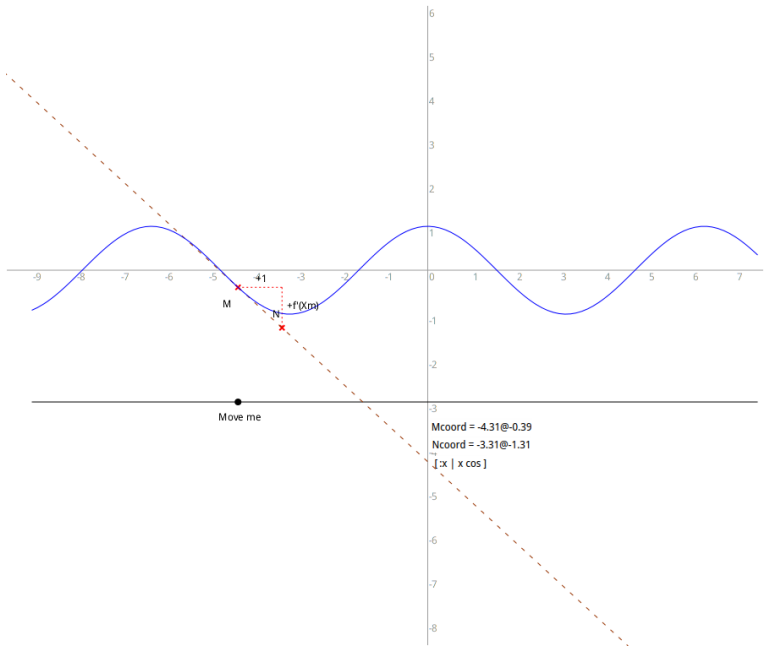

Figure 6.5: Courbe et tangente en un point

Dans une nouvelle figure, nous commençons par construire un segment horizontal, nous y plaçons un point libre appelé "Move me". Ce point servira de base à la construction de la courbe comme lieu d'un point.

[7] Cette fonctionnalité est nativement présente dans Dr.Geo, il s'agit donc là d'un prétexte à un exercice.

[8] Nom de message comprenant des "**:**"

Définir un fonction

Comme un script est capable de retourner n'importe quel type d'objet, le premier de notre construction définira simplement la fonction utilisée. Pour ce faire nous utilisons des objets Pharo de type bloc de code – fonction anonyme en Lisp. Nous nommons ce script `Function`, sans arguments, sa méthode `compute` est :

```
compute
"La définition de notre fonction"
   ^ [:x | x cos]
```

Ensuite nous le plaçons dans la figure[9]. Ainsi le bloc de code retourné par `Function` attend un argument `:x` et retourne le cosinus de celui-ci. Nous verrons dans la suite comment manipuler ce script.

Image d'une valeur par une fonction

Maintenant nous calculons les coordonnées d'un point appartenant à la courbe. Nous utilisons notre point "Move me" et notre fonction. Ce script `Mcoord` aura comme arguments ce point et le script `Function` (dans cet ordre) :

```
compute
   ^ (self arg1 x) @ (self arg2 compute value: self arg1 x)
```

L'abscisse de ses coordonnées est la même que celle du point de départ, son ordonnée est l'image de son abscisse par la fonction.

Noter :

- `self arg2`, l'accès au script `Function`, qui est un bloc de code définissant la fonction ;
- la définition de l'ordonnée, le passage de l'argument à la fonction doit se comprendre comme `Function(item x)`.

Maintenant utilisons ce script `Mcoord` avec comme argument le point "Move me!" ; le résultat du script est de la forme `1.2@0.5`, cela représente un couple de coordonnées.

Avec l'outil point ...Points → Coordonnées... créons un point ayant ses coordonnées contraintes par le résultat de ce script.

L'outil lieu d'un point ...Lignes → Lieu de point... donne ensuite la courbe en sélectionnant nos deux points.

Pente en un point de la courbe d'une fonction et tangente

Pour ce faire, nous calculons une valeur approchée de la pente en un point de la courbe, avec celle-ci nous déduirons un deuxième point de la tangente.

$p = (f(x + 0.0001) - f(x))/0.0001$

Cela se traduit par un script `Ncoord` avec comment argument le point où calculer une approximation de la pente et le script `Function` :

```
compute
| p x f |
   f := self arg2 compute.
   x := self arg1 point x.
   p := ((f value: x + 0.0001) - (f value: x)) / 0.0001.
   ^ self arg1 point + (1 @ p)
```

Nous plaçons ensuite ce script dans la figure.

[9] Il est important de le référencer dans la figure afin qu'il soit inclus dans la description de celle-ci lors d'une opération de sauvegarde sur fichier.

Noter :

- La déclaration de variables temporaires `| p x f |`. Les variables ne sont pas typées, pas de soucis de ce côté là.
- La référence du bloc de code avec une variable `f := self arg2 compute`. Le symbole pour assigner une valeur à une variable est "`:=`".
- Les parenthèses ! Pharo ne connait pas la priorité des opérateurs, en fait ils n'existent pas dans ce langage. Le lecteur est invité à étudier la section sur les messages Pharo du livre *Pharo By Exemple*.

Utilisons ce script avec comme arguments le point de notre courbe et notre script `Function`. Nous obtenons un deuxième couple de coordonnées. Avec celles-ci construisons un point, la tangente est la droite définie par ce point et celui de la courbe.

En déplaçant le point "Move me!", la tangente est recalculée. Tout aussi intéressant : modifier le script `Function` actualise correctement l'ensemble de notre construction. Quelques exemples de modifications :

```
^ [:x | x * x / 10]

^ [:x | x cos + (10 * x) sin]

^ [:x | (x * 5) cos + x abs]
```

6.5 Éditer un script comme un pro

Dr.Geo propose de modifier les scripts avec son éditeur de script, c'est en fait un éditeur de classe simplifié. Il est possible d'utiliser Calypso, le navigateur de classe de l'environnement Pharo. Il offre plus de possibilités mais il est plus complexe. Nous le présentons brièvement.

Pour ouvrir Calypso ...Clic arrière-plan → Outils → Navigateur système... Ensuite dans la fenêtre nouvellement ouverte, à gauche clic sur "DrGeoII-User" pour afficher les scripts – Voir [scriptClass], page 43.

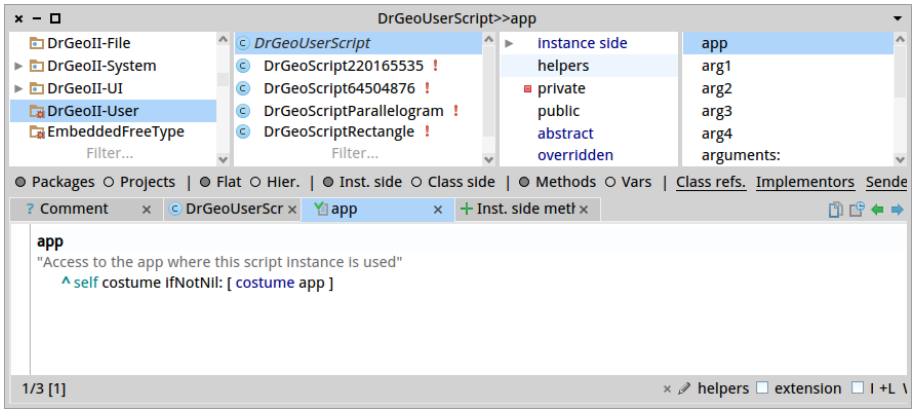

Figure 6.6: Navigateur de classe Calypso

De la gauche vers la droite et du haut vers le bas y sont présentées :

- Les bibliothèques de classes, ici `DrGeoII-User`, c'est l'emplacement des scripts utilisateur.
- La liste des scripts utilisateur, de la forme `DrGeoScriptxxxx`, les classes représentant les scripts.

- Les catégories de méthodes par classe (script) pour les ranger, ici une catégorie dédiée `public`. L'utilisateur en crée d'autres si besoin. La catégorie `public` activée, c'est dans celle-ci que toute nouvelle méthode est placée.
- Les méthodes de la catégorie sélectionnée. Ici, dans la catégorie `public`, se trouve l'unique méthode `compute`. En cliquant sur une méthode, son code source est affiché dans l'éditeur de texte en bas. **La méthode `compute` est créée par défaut par Dr.Geo. C'est cette méthode qui retourne le résultat à afficher dans la figure.** Attention à ne pas la supprimer cela provoquerait une erreur, il est toutefois possible de la recréer lors d'une telle erreur depuis le navigateur de classe.
- Sur toute la largeur, la zone texte pour édition du code source de la méthode sélectionnée. Pour valider une modification, il suffit de presser les touches *`Ctrl-s`*.

6.6 Disséquer un script

Quelques éléments d'explication sur la représentation d'un script :

- **First class citizen.** Comme déjà expliqué, un script est une classe Pharo.
- **Instance.** Tant que des paramètres d'entrée différents sont choisis, un même script s'utilise plusieurs fois dans une même figure, ce sont des instances du script – de sa classe pour être précis.
- **Etat.** A son utilisation dans une figure correspond une instance de celle-ci. L'état d'une instance – la valeur des variables d'instance – est donc différent de celui d'une autre instance et il perdure tout au long du cycle de vie de la figure et de l'instance. Au delà, des traitements d'un script, cela permet donc de conserver et de faire évoluer son état tout au long du cycle de vue d'une figure.
- **Coté classe.** Dans la navigateur de classes (Voir [scriptClass], page 43), en pressant le bouton à coché `Class side`, Dr.Geo affiche des méthodes de classe. Ce sont des méthodes communes à toute les instances de ce script, on y trouve son nom, sa description et la liste de ses arguments. Ils sont modifiables.

6.7 Méthodes de référence

Un argument passé à un script est toujours une référence vers une instance de classe de la hiérarchie `DrGMathItem`, elle est le modèle de base pour représenter tout objet d'une figure. Ainsi pour connaître les messages compris par un argument d'un script, il convient d'examiner la hiérarchie de `DrGMathItem`. Celle-ci comprend plus de 80 classes, mais du fait des héritages, seules quelques unes sont intéressantes pour les usages courants des scripts :

- `DrGMathItem`
- `DrGPointItem`
- `DrGDirectionItem` concerne segment, droite, demi-droite, vecteur
- `DrGArcItem`
- `DrGCircleItem`
- `DrGLocus2ptsItem`
- `DrGPolygonNptsItem`
- `DrGValueItem`

Pour explorer ces classes, utiliser par exemple Spotter, l'outil de recherche de Pharo. Pour l'invoquer *`Shift-Enter`*, puis saisir le nom d'une classe, par exemple “DrGPointItem” et choisir dans le résultat de la recherche. Le navigateur de classe Calypso s'affiche alors sur la classe recherchée.

Les sections suivantes contiennent la description de quelques messages utiles. Elles sont présentées par classe. Pour plus de concision dans la description de l'API, le préfixe `DrG` dans le nom des classes est omis.

6.7.1 Item math

Cette section regroupe des méthodes de la classe `DrGMathItem`, classe mère de la hiérarchie des objets d'une figure.

Ces messages peuvent donc être envoyés à tous les types d'objets passés en argument à un script.

`<String> safeName` [Méthode sur `MathItem`]

⇒ une chaîne de caractères représentant le nom de l'item

```
nom := point1 safeName
^nom asUppercase.
```

`<Boolean> exist` [Méthode sur `MathItem`]

⇒ un booléen indiquant si l'item est dans un état permettant son existence

Du fait de la dimension dynamique d'une figure géométrique, un objet peut ne plus exister temporairement. C'est par exemple le cas pour le point d'intersection de deux segments, cette méthode permet alors de vérifier l'existence de l'intersection.

```
line exist ifTrue: [ position := line origin ]
```

`<Collection> parents` [Méthode sur `MathItem`]

⇒ une collection d'items, parents de l'item

```
point1 := segment parents first
```

`move:` *unVecteur* [Méthode sur `MathItem`]

unVecteur, instance de `Point`, vecteur de coordonnées (x,y) représentant le déplacement

Déplace un item dans une direction donnée, tout en tenant compte de ses contraintes.

```
circle move: 2@1
```

`<Point> closestPointTo:` *unPoint* [Méthode sur `MathItem`]

⇒ coordonnées du point sur l'item le plus proche de *unPoint* *unPoint*, un couple de coordonnées

Cette méthode prend tout son sens sur des items de type ligne comme droite, cercle, arc, polygone, etc.

```
position := segment closestPointTo: 2@1

position := arc closestPointTo: 2@1
```

6.7.2 Point

Un item point – objet point defini dans une construction Dr.Geo – passé en argument à un script est un objet très complexe. Il peut être un point libre dans le plan, sur une ligne, une intersection, etc. Quelques méthodes spécifiques permettent d'exploiter ce type d'objet depuis les scripts.

`<Point> point` [Méthode sur `PointItem`]

⇒ coordonnées de ce point

L'objet retourné est une instance de *Point*, son abscisse et ordonnée s'obtiennent par le message *#x* et *#y* respectivement.

```
abscissa := pointA point x
```

`point:` *aPoint* [Méthode sur `PointItem`]

aPoint, couple de coordonnées

Modifie les coordonnées de l'item point tout en respectant ses contraintes propres. Pour un point libre, la nouvelle position sera exactement celle donnée en argument, pour un point contraint (sur ligne, intersection) cela sera sans effet.

```
pointA point: 5@2
```

`<Float> abscissa` [Méthode sur `PointOnCurveItem`]

⇒ abscisse curviligne de ce point sur sa ligne, elle est normalisée sur [0 ; 1]

Cette méthode est réservée au point libre sur une ligne

```
a := mobile abscissa
```

`abscissa:` *uneValeur* [Méthode sur `PointOnCurveItem`]

uneValeur, valeur décimale de [0 ; 1]

Modifie l'abscisse curviligne d'un point libre sur une ligne.

```
pointItem abscissa: 0.5
```

`moveAt:` *unPoint* [Méthode sur `PointItem`]

unPoint, couple de coordonnées (x,y) où déplacer le point

Déplace le point à la position donnée en argument.

```
point moveAt: 2@1
```

6.7.3 Ligne droite ou courbe

Ces messages sont à envoyer aux objets de type ligne comme droite, cercle, polygone, etc.

`<Float> abscissaOf:` *unPoint* [Méthode sur `CurveItem`]

⇒ une valeur de [0 ; 1], abscisse curviligne de *unPoint* sur la ligne

unPoint, un point (x,y)

```
a := curve abscissaOf: 2@1
```

`<Point> pointAt:` *uneValeur* [Méthode sur `CurveItem`]

⇒ coordonnées du point de la ligne d'abscisse curviligne *uneValeur*

uneValeur, un nombre de [0 ; 1]

```
myPoint := curve pointAt: 0.5.
^ myPoint x
```

`<Boolean> contains:` *unPoint* [Méthode sur `CurveItem`]

⇒ un booléen indiquant si *unPoint* est sur la ligne

unPoint, un point (x,y)

```
(curve contains: 0@1) ifTrue: [^ 'Yes!']
```

6.7.4 Droite, demi-droite, segment, vecteur

Ensemble de méthodes dédiées aux lignes droites.

`<Point> origin` [Méthode sur `DirectionItem`]

⇒ un point origine de cette ligne

D'un point de vue interne les droites sont graduées avec une origine. Cela permet de positionnement de points sur celle-ci.

```
segment origin
```

`<Point> direction` [Méthode sur `DirectionItem`]
⇒ un vecteur (x,y) indiquant la direction de la ligne

```
v := droite direction.
pente := v y / v x
```

`<Point> normal` [Méthode sur `DirectionItem`]
⇒ un vecteur unitaire normal à la direction de la ligne

```
n := vecteur normal
```

6.7.5 Segment

Ensemble de méthodes dédiées aux segments.

`<Float> length` [Méthode sur `SegmentItem`]
⇒ longueur du segment

```
segment := canvas segment: 0@0 to: 5@5.
l := segment length
```

`<Point> extremity1` [Méthode sur `SegmentItem`]
⇒ coordonnées de l'extrémité 1 du segment

```
segment := canvas segment: 0@0 to: 5@5.
p := segment extremity1.
```

`<Point> extremity2` [Méthode sur `SegmentItem`]
⇒ coordonnées de l'extrémité 2 du segment

```
segment := canvas segment: 0@0 to: 5@5.
p := segment extremity2
```

`<Point> middle` [Méthode sur `SegmentItem`]
⇒ coordonnées du milieu du segment

```
segment := canvas segment: 0@0 to: 5@5.
m := segment middle
```

6.7.6 Cercle, arc de cercle, polygone

Ensemble de méthodes réservées aux cercle, arc et polygone.

`<Point> center` [Méthode sur `CircleItem|ArcItem`]
⇒ point, centre du cercle ou de l'arc de cercle

```
c := arcAB center
```

`<Float> radius` [Méthode sur `CircleItem|ArcItem`]
⇒ rayon du cercle ou de l'arc de cercle

```
rayon := monCercle radius
```

`<Float> length` [Méthode sur `CircleItem|ArcItem|PolygonItem`]
⇒ longueur du cercle, de l'arc de cercle ou du polygone

```
perimetre := triangle length
```

6.7.7 Valeur

`<Float> valueItem` [Méthode sur `ValueItem`]
⇒ valeur de cet item

```
n1 := item2 valueItem.
n2 := item2 valueItem.
n1 + n2.
```

`valueItem:` *uneValeur* [Méthode sur `ValueItem`]
uneValeur, valeur décimale

Modifie la valeur d'un item de type valeur libre.

```
item valueItem: 5.2.
```

`position:` *unPoint* [Méthode sur `ValueItem`]
unPoint, point (x,y)

Déplace dans la figure l'item à la position *unPoint*.

```
maValeur position: 0.5@2.
```

6.7.8 Angle

`<Integer> degreeAngle` [Méthode sur `AngleItem`]
⇒ une mesure en degrés de cet angle orienté ou géométrique

```
angle1 := a1 degreeAngle.
```

`<Float> radianAngle` [Méthode sur `AngleItem`]
⇒ une mesure en radian de cet angle orienté ou géométrique

```
angle1 := a1 radianAngle.
```

6.8 Aspect des objets

Un script, en plus de faire des calculs, peut modifier l'aspect d'un objet mathématique passé en argument, de lui même ou de tout autre objet d'une figure (i.e. canevas).

Précédemment, nous avons discuté des modèles d'objet géométrique, ils sont tous des instances de classe de la hiérarchie `DrGMathItem` ; leur représentant graphique à l'écran sont appelés **costumes**, instances de classe de la hiérarchie `DrGMathItemCostume`. Accéder au costume d'un modèle offre la possibilité d'éditer son aspect : couleurs, style de traits, etc. Quelques méthodes sont décrites dans la section suivante pour accéder aux costumes et aux styles. Le lecteur curieux explorera les hiérarchies des classes `DrGMathItemCostume` et `DrGCostumeStyle` pour en découvrir davantage.

`<MathItemCostume> costume` [Méthode sur `DrGeoUserScript`]
⇒ costume du script lui-même. Il faut toujours tester l'existence du costume avant son utilisation (voir exemple)

```
compute
   self costume ifNotNil:
      [self costume style color: Color blue].
```

`<MathItemCostume> costume1` [Méthode sur `DrGeoUserScript`]
⇒ le costume du modèle passé en premier argument à ce script

```
paintBlue
   self costume1 style color: Color blue
```

`<MathItem> arg1` [Méthode sur `DrGeoUserScript`]
⇒ le modèle passé en premier argument à ce script. Son type exact dépend des arguments avec lesquels l'instance du script a été créée.

```
isParallelogram
"Test my polygon argument is a parallelogram"
   | points mid1 mid2|
   points := self arg1 points.
   mid1 := points first + points third / 2.
   mid2 := points second + points fourth / 2.
   ^ points size = 4 and: [mid1 = mid2]
```

Des paires de méthodes pour les arguments suivants sont disponibles : `costume2`/`arg2`, `costume3`/`arg3` et `costume4`/`arg4`.

`<DrGeo> app` [Méthode sur `DrGeoUserScript`]
⇒ l'instance Dr.Geo où ce script existe. Entre autres choses, donne accès aux costumes de la figure.

```
polygonCostumes
"Return the list of polygon costume in the canvas"
   ^ self app costumes select: [ :costume |
      costume mathItem isPolygonItem].
```

7 Figure Pharo

Les *Figures Pharo de Dr.Geo* – ou plus simplement figure dans la suite de ce texte – sont des figures écrites en langage Pharo. Il ne s'agit donc plus de construire une figure à l'aide de l'interface graphique de Dr.Geo mais plutôt de décrire une figure avec le langage Pharo et une API (interface de programmation dédiée)[1].

7.1 Exemples de figure Pharo

En lui-même Pharo est un langage de très haut niveau. Lorsqu'une figure est définie dans ce langage, nous disposons également de toute sa puissance pour par exemple définir récursivement telle partie de la figure, ou bien pour placer aléatoirement certains objets de telle sorte qu'à chaque ouverture de la figure, celle-ci soit légèrement différente. Bref, les FSD sont libérées du carcan de l'interface graphique tout en étant renforcées par le langage Pharo.

Une FSD est un code source Pharo à exécuter dans un espace de travail ...Clic arrière-plan → Outils → Espace de travail...[2] C'est une fenêtre texte depuis laquelle du code Pharo est écrit et exécuté. On peut aussi y coller du texte par `Ctrl-v`. Voir [workspace], page 82, pour plus d'information sur cet outil.

Nous allons étudier plusieurs exemples, chacun d'eux sera écrit dans un espace de travail et exécuté en sélectionnant le code puis la séquence de touches `Ctrl-d` pour *Do-it!*[3].

Commençons par étudier un exemple simple :

```
DrGeoFigure nouveau
```

C'est la plus petite description produisant une figure. Lors de son exécution, celle-ci va simplement créer une nouvelle figure vide. La figure Dr.Geo est affichée dans une fenêtre simplifiée puisqu'elle ne comporte pas la barre d'outils, seulement la barre des menus et les molettes.

Abordons un deuxième exemple :

```
| c item |
c := DrGeoFigure nouveau.
item := c point: 1.2 @ -2.
item nommer: 'A'.
```

Cette description définit une figure avec un point libre A de coordonnées initiales (1,2 ; -2). Quelques explications sur ce code :

- Deux variables temporaires *c* et *item* sont déclarées entre deux barres verticales `| |`. Il n'y pas de type, les variables sont toujours des références vers des objets.[4]
- Les objets sont ajoutés un à un à la figure en envoyant un message approprié à celle-ci. Ici le message à mot clé `#point:`, avec comme argument deux coordonnées, crée un point libre.

  Le résultat est une capsule[5] sur un objet géométrique "point" de Dr.Geo qu'il est possible de modifier par l'envoi de messages, ici `#nommer:` pour le renommer 'A'.

---

[1] `https://fr.wikipedia.org/wiki/Interface_de_programmation`

[2] Le raccourci `Alt-k` fonctionne aussi lorsqu'aucune fenêtre est sélectionnée.

[3] Alternativement, c'est l'entrée à choisir dans le menu contextuel de l'espace de travail.

[4] Des instances de classes qui représentent des objets géométriques.

[5] Pour être précis, la capsule est un objet `DrGWrappedPoint` dont l'objectif est de simplifier la manipulation des objets géométriques de type point, pour aussi bien obtenir ses attributs ou modifier son style. Il est toujours possible d'accéder à l'objet point sous-jacent en envoyant le message `#mathItem` à la capsule. Il est alors possible d'utiliser les méthodes décrites dans le chapitre sur les scripts.

Poursuivons avec un troisième exemple :

```
| c triangle hasard m n p |
triangle := [:p1 :p2 :p3 |
c segmentDe: p1 a: p2.
c segmentDe: p2 a: p3.
c segmentDe: p3 a: p1].
hasard := [5 - 10 auHasard].
c := DrGeoFigure nouveau.
m := c point: hasard valeur @ 0.
n := c point: 5 @ 0.
p := c point: hasard valeur @ 3.
triangle valeur: m valeur: n valeur: p.
```

Cet exemple est particulièrement intéressant, il nous montre trois choses importantes :

1. L'introduction d'une construction de plus haut niveau, non prévue au départ par Dr.Geo. Ici nous avons défini le bloc de code *triangle* qui, à partir de trois points, construit le triangle passant par ces trois points. Nous pouvons comparer ceci avec les macro-constructions mais avec une approche différente, orientée programmation.
2. La définition d'un bloc de code associé, ici nous avons défini *hasard* qui retourne un nombre entier compris entre -5 et 5. Nous utilisons ce bloc pour placer au hasard certains points de notre figure, ainsi à chaque exécution la figure est légèrement différente.
3. L'affectation du résultat d'une construction à une variable n'est pas obligatoire, nous l'utilisons lorsque nous souhaitons garder une référence de l'objet créé. Par exemple dans le bloc de code *triangle*, nous ne gardons pas de référence des segments créés, en revanche lorsque nous définissons nos trois points nous avons besoin de garder une référence dans des variables temporaires. Ainsi, lors de l'utilisation du bloc de code *triangle*, nous passons en paramètre les variables *m*, *n* et *p*.

Pour clore cette section, voici un dernier exemple :

```
| c a b d |
c := DrGeoFigure nouveau.
a := c point: 1@0.
b := c point: 5@0.
d := c line: a to: b.
a couleur: Color yellow;
   rond;
   large.
b cacher.
d tiret.
```

Deux points et une droite sont créés. Ensuite des commandes sont utilisées pour modifier l'aspect des objets, voire pour en cacher.

Nous avons terminé notre petite visite guidée des *Figures Pharo*. Dans les sections suivantes nous exposons l'ensemble des commandes disponibles.

7.2 Méthodes de référence pour les Figures Pharo

Pour construire un objet vous envoyez un message à la figure, l'objet construit est édité en lui envoyant des messages spécifiques.

Cependant, avant tout ajout d'un objet à une figure, cette dernière doit être créée avec la commande `DrGeoFigure nouveau`.

Dans la description de l'API des sections suivantes, pour plus de concision, l'objet capsule `DrGWrappedPoint` est noté `WrpPoint`, de même pour les autres capsules.

7.2.1 Commandes générales

`<une DrGeoFigure>` *nouveau* [Méthode sur `DrGeoFigure`]
⇒ Figure et affiche celle-ci dans une fenêtre. Le résultat est nécessaire pour créer des objets dans cette figure, il est donc important de la placer dans une variable.

```
| figure |
figure := DrGeoFigure nouveau.
```

`<une DrGeoFigure>` *minimal* [Méthode sur `DrGeoFigure`]
⇒ Figure et affiche celle-ci dans une fenêtre **sans décoration**.

C'est une alternative à la méthode `nouveau` pour créer une figure.

`supprimer` [Méthode sur `DrGeoFigure`]
Supprime la figure et ferme sa fenêtre

```
| figure |
figure := DrGeoFigure nouveau.
figure supprimer
```

`faire:` *bloc* [Méthode sur `DrGeoFigure`]
bloc, bloc de code Pharo contenant des instructions de construction et/ou d'animation de la figure interactive.

Exécute le bloc de code dans un processus en tâche de fond. A utiliser lorsque la construction doit se faire sous les yeux de l'utilisateur ou bien lorsque la figure est animée.

```
| figure point |
figure := DrGeoFigure nouveau.
point := figure point: 0@0.
figure do: [
   -5 a: 5 par: 0.1 faire: [:x |
      point deplacerA: x@(x cos * 3).
      (Delay forMilliseconds: 100) wait.
      figure actualiser]
]
```

`actualiser` [Méthode sur `DrGeoFigure`]
Mise à jour de la figure après modification des attributs de quelques items. La plupart du temps ce n'est pas nécessaire.

`pleinEcran` [Méthode sur `DrGeoFigure`]
La fenêtre de la figure est étendue pour couvrir tout l'écran.

`afficherGrille` [Méthode sur `DrGeoFigure`]
Affiche la grille de la figure.

`centrerVueEn:` *unPoint* [Méthode sur `DrGeoFigure`]
unPoint, coordonnées d'un point.

La figure est décalée afin d'afficher le point donné en argument au centre de la fenêtre.

```
figure centrerVueEn: 5@0
```

`echelle:` *unEntier* [Méthode sur `DrGeoFigure`]
unEntier, échelle de la figure. Une unité représente approximativement 1 pixel.

Modifie l'échelle de la figure.

```
figure echelle: 10
```

7.2.2 Point

`<WrpPoint> point:` *pointOuBloc* [Méthode sur `DrGeoFigure`]

pointOuBloc, un couple de coordonnées (x,y) ou un bloc de code retournant un couple de coordonnées. Dans le deuxième cas, le bloc est exécuté à chaque fois que les coordonnées du point sont demandées.

⇒ référence d'un point libre du plan de coordonnées *pointOuBloc*.

```
| fig |
fig := DrGeoFigure nouveau.
fig point: 5@2.
fig point: [ 5 auHasard @ 5 auHasard ].
```

`<WrpPoint> pointX:Y:` *v1 v2* [Méthode sur `DrGeoFigure`]

v1, un objet valeur

v2, une objet valeur

⇒ référence d'un point contraint par ses coordonnées

```
figure
   pointX: (figure valeurLibre: 2) cacher
   Y: (fiugre valeurLibre: 5) cacher.
```

`<WrpPoint> pointSurLigne:a:` *curve a* [Méthode sur `DrGeoFigure`]

curve, référence d'une ligne (droite, demi-droite, segment, etc.)

a, abscisse curviligne du point libre, la valeur est normalisée sur [0 ; 1].

⇒ référence d'un point libre sur une ligne

```
myPoint := figure pointSurLigne: s1 at: 0.5.
```

`<WrpPoint> milieuDe:et:` *p1 p2* [Méthode sur `DrGeoFigure`]

p1, référence d'un point ou d'un couple de coordonnées

p2, référence d'un point ou d'un couple de coordonnées

⇒ référence du milieu des deux points

```
| a i |
a := figure point: 1@1.
i := figure milieuDe: a et: 4@4.
```

`<WrpPoint> milieuDe:` *s* [Méthode sur `DrGeoFigure`]

s, référence d'un segment

⇒ référence du milieu du segment

```
figure milieuDe: s.
```

`<WrpPoint> intersectionDe:et:` *l1 l2* [Méthode sur `DrGeoFigure`]

l1, référence d'une ligne

l2, référence d'une ligne

⇒ référence du point d'intersection des deux lignes

```
figure intersectionDe: droite et: segment
```

`<WrpPoint> autreIntersectionDe:et:` *l1 l2* [Méthode sur `DrGeoFigure`]

l1, référence d'une ligne

l2, référence d'une ligne

⇒ référence de l'autre point d'intersection des deux lignes, lorsqu'il existe.

```
figure autreIntersectionDe: droite et: circle.
```

`<WrpPoint> point:parent:` *bloc item* [Méthode sur `DrGeoFigure`]

bloc, bloc de code retournant un point

item, référence d'un item géométrique

⇒ référence d'un point dont les coordonnées sont calculées avec le bloc de code ayant comme argument `item`.

```
| figure s mobile c block |
figure := DrGeoFigure nouveau.
s := figure segment: -5@0 to: 5@0.
mobile := figure pointSurLigne: s a: 0.1.
block := [:mathItem | |x|
   x := mathItem point x.
   x @ (x * x * x / 25 - x)].
c := figure point: block parent: mobile.
figure lieuDe: c lorsqueBouge: mobile.
```

`<WrpPoint> point:parents` *bloc liste* [Méthode sur `DrGeoFigure`]

bloc, bloc de code retournant un point

liste, une collection d'items géométriques

⇒ référence d'un point dont les coordonnées sont calculées avec le bloc de code ayant comme argument *liste*.

```
| figure d m p |
figure := DrGeoFigure nouveau.
d := figure droitePassantPar: -2 @ 1 et: 3 @ 3.
d couleur: Color blue.
m := figure point: 1 @ -1.
p := figure
   point: [:parents |
      parents first closestPointTo: parents second point]
   parents: {d . m}.
```

7.2.3 Droite

`<WrpCurve> droitePassantPar:et:` *p1 p2* [Méthode sur `DrGeoFigure`]

p1, référence d'un point ou d'un couple de coordonnées

p2, référence d'un point ou d'un couple de coordonnées

⇒ référence d'une droite passant par deux points

```
| p1 |
p1 := figure point: 0@0.
figure droitePassantPar: p1 et: 1@2.
```

`<WrpCurve> paralleleA:passantPar:` *d p* [Méthode sur `DrGeoFigure`]

d, référence d'une direction (droite, segment, vecteur,...)

p, référence d'un point ou d'un couple de coordonnées

⇒ référence d'une droite parallèle à la direction de *d* et passant par *p*

```
| a |
a := figure point: 1@5.
figure paralleleA: d passantPar: a.
```

`<WrpCurve> perpendiculaireA:passantPar:` *d p* [Méthode sur `DrGeoFigure`]

d, référence d'une direction (droite, segment, vecteur,...)

p, référence d'un point ou d'un couple de coordonnées

⇒ référence d'une droite perpendiculaire à la direction de *d* et passant par *p*

```
figure perpendiculaireA: d passantPar: 1@5.
```

`<WrpCurve> mediatriceDe:` *s* [Méthode sur `DrGeoFigure`]

s, référence d'un segment

⇒ référence de la médiatrice du segment *s*

```
c mediatriceDe: (c segmentDe: 0@0 a: 4@4)
```

`<WrpCurve> mediatriceDe:a:` *a b* [Méthode sur `DrGeoFigure`]

a, référence d'un point ou d'un couple de coordonnées

b, référence d'un point ou d'un couple de coordonnées

⇒ référence de la médiatrice du segment *ab*

```
figure mediatriceDe: 0@0 a: 4@4
```

`<WrpCurve> bissectriceDe:` *a* [Méthode sur `DrGeoFigure`]

a, référence d'un angle géométrique défini par **trois points**

⇒ référence de la bissectrice de l'angle *a*

```
figure bissectrice: angle
```

`<WrpCurve> bissectriceSommet:cote1:cote2` *a b c* [Méthode sur `DrGeoFigure`]

a,*b*,*c*, points définissant l'angle géométrique bac

⇒ référence de la bissectrice de l'angle bac

```
figure bissectriceSommet: 0@0 cote1: 1@0 cote2: 0@1
```

7.2.4 Demi-droite

`<WrpCurve> demiDroiteOrigine:passantPar:` *o p* [Méthode sur `DrGeoFigure`]

o, référence d'un point ou d'un couple de coordonnées, origine de la demi-droite

p, référence d'un point ou d'un couple de coordonnées, point de la demi-droite

⇒ référence d'une demi-droite définie par son origine et un point

```
| a |
a := figure point: 1@5.
figure demiDroiteOrigine: 0@0 passantPar: a.
```

7.2.5 Segment

`<WrpSegment> segmentDe:a:` *p1 p2* [Méthode sur `DrGeoFigure`]

p1, référence d'un point ou d'un couple de coordonnées

p2, référence d'un point ou d'un couple de coordonnées

⇒ référence d'un segment défini par ses extrémités

```
| a |
a := figure point: 5@5.
figure segmentDe: 10@10 a: a.
```

7.2.6 Cercle

`<WrpFilledCircle> cercleCentre:passantPar:` *c p* [Méthode sur `DrGeoFigure`]

c, référence d'un point ou d'un couple de coordonnées, centre du cercle

p, référence d'un point ou d'un couple de coordonnées, point du cercle

⇒ référence d'un cercle défini par son centre et un point

```
| a |
a := figure point: 1@5.
figure cercleCentre: a passantPar: 10@4.
```

`<WrpFilledCircle> cercleCentre:rayon:` *c r* [Méthode sur `DrGeoFigure`]

c, référence d'un point ou d'un couple de coordonnées, centre du cercle

r, référence d'un item numérique ou d'une valeur numérique, rayon du cercle

⇒ référence d'un cercle défini par son centre et son rayon

```
| a r |
a := figure point: 1@5.
r := figure valeurLibre: 4.
figure cercleCentre: a rayon: r.
figure cercleCentre: 4@4 rayon: 5
```

`<WrpFilledCircle> cercleCentre:segment:` *c s* [Méthode sur `DrGeoFigure`]

c, référence d'un point ou d'un couple de coordonnées, centre du cercle

r, référence d'un segment, rayon du cercle

⇒ référence d'un cercle défini par son centre et de rayon la longueur du segment

```
| a s |
a := figure point: 1@5.
s := figure segmentDe: 0@0 a: 3@0.
figure cercleCentre: a segment: s.
```

7.2.7 Arc de cercle

`<WrpFinitCurve> arcDe:a:passantPar:` *p1 p2 p3* [Méthode sur `DrGeoFigure`]

p1, référence d'un point ou d'un couple de coordonnées, 1ère extrémité de l'arc

p2, référence d'un point ou d'un couple de coordonnées, 2ème extrémité de l'arc

p3, référence d'un point ou d'un couple de coordonnées de l'arc

⇒ référence d'un arc de cercle défini par ses extrémités et un point

```
| a b |
a := figure point: 1@5.
b := figure point: 0@5.
figure arcDe: a a: -1 @ -2 passantPar: b.
```

`<WrpFinitCurve> arcCentre:de:a:` *O A B* [Méthode sur `DrGeoFigure`]

O, référence d'un point ou d'un couple de coordonnées, centre de l'arc

A, référence d'un point ou d'un couple de coordonnées, origine de l'arc

B, référence d'un point ou d'un couple de coordonnées, tel que l'angle de l'arc est AOB

⇒ référence d'un arc de cercle défini par son centre et l'angle AOB

```
| a b |
a := figure point: 1@5.
b := figure point: 0@5.
figure arcCentre: a de: b a: -1 @ -2.
```

7.2.8 Polygone

`<WrpFilledCurve> polygone:` *collection* [Méthode sur `DrGeoFigure`]

collection, une liste de références de points ou de couples de coordonnées ; sommets du polygone

⇒ référence d'un polygone défini par ses sommets

```
| b |
b := figure point: 1@3.
figure polygone: {1@2. b. 0@0. d}
```

`<WrpFilledCurve>` [Méthode sur `DrGeoFigure`]
`polygoneRegulierCentre:sommet:cotes:` *c s n*

c, référence d'un point ou d'un couple de coordonnées, centre du polygone

s, référence d'un point ou d'un couple de coordonnées, sommet du polygone

n, référence d'une valeur ou valeur, nombre de sommets du polygone

⇒ référence d'un polygone régulier défini par son centre, un de ses sommets, et son nombre de sommets

```
| b |
b := figure point: 1@3.
figure polygoneRegulierCentre: b sommet: 1@1 cotes: 7.
```

7.2.9 Les transformations géométriques

Les transformations géométriques permettent la construction des tranformés d'objets. Elles s'appliquent à des références d'objets de type point, segment, droite, demi-droite, vecteur, cercle, arc de cercle, polygone et lieu de point.

`<WrpCurve> rotationDe:parCentre:etAangle:` *i* [Méthode sur `DrGeoFigure`]
c a

i, référence de l'objet à transformer (point, segment, droite, demi-droite, vecteur, cercle, arc-cercle, polygone)

c, référence d'un point ou d'un couple de coordonnées, centre de la rotation

a, référence d'un item valeur ou d'une valeur, angle de la rotation

⇒ référence de l'objet transformé

```
| c k l |
c := figure point: 5@5.
k := 3.1415.
l := figure droitePassantPar: 0@0 et: 5@5.
figure rotationDe: l parCentre: c etAngle: k.
figure rotationDe: l parCentre: 0@0 etAngle: Float pi / 3.
```

`<WrpCurve>` [Méthode sur `DrGeoFigure`]
`homothetieDe:parCentre:etFacteur:` *i c k*

i, référence de l'objet à transformer (point, segment, droite, demi-droite, vecteur, cercle, arc-cercle, polygone)

c, référence d'un point ou d'un couple de coordonnées, centre de l'homothétie

k, référence d'un item valeur ou d'une valeur, facteur de l'homothétie

⇒ référence de l'objet transformé

```
| c k l |
c := figure point: 5@5.
k := -3.
```

```
l := figure droitePassantPar: 0@0 et: 5@5.
figure homothetieDe: l parCentre: c etFacteur: k.
figure homothetieDe: l parCentre: 0@0 etFacteur: 5.
```

`<WrpCurve> symetriqueDe:selonCentre` *i c* [Méthode sur `DrGeoFigure`]

i, référence de l'objet à transformer (point, segment, droite, demi-droite, vecteur, cercle, arc-cercle, polygone)

c, référence d'un point ou d'un couple de coordonnées, centre de la symétrie

⇒ référence de l'objet transformé

```
| a |
a := figure point: 4@2.
figure symetriqueDe: a selonCentre: 0@0
```

`<WrpCurve> symetriqueDe:selonAxe:` *i axe* [Méthode sur `DrGeoFigure`]

i, référence de l'objet à transformer (point, segment, droite, demi-droite, vecteur, cercle, arc-cercle, polygone)

axe, référence d'une droite, axe de la réflexion

⇒ référence de l'objet transformé

```
symetriqueDe: polygone selonAxe: d1
```

`<WrpCurve> translationDe:parVecteur:` *i v* [Méthode sur `DrGeoFigure`]

i, référence de l'objet à transformer (point, segment, droite, demi-droite, vecteur, cercle, arc-cercle, polygone)

v, référence d'un item vecteur ou d'un couple de coordonnées

⇒ référence de l'objet transformé

```
| u a|
u := figure vecteurOrigine: 1@1 extremite: 3@2.
a := figure translationDe: (figure point: 2@1) parVecteur: u
```

```
| u a|
a := figure translationDe: (figure point: 2@1) parVecteur: 2@1
```

7.2.10 Lieu d'un point

`<WrpCurve> lieuDe:lorsqueBouge:` *c m* [Méthode sur `DrGeoFigure`]

m, référence d'un point mobile sur une ligne

c, référence d'un point fixe dépendant du point *m*

⇒ référence d'un lieu

```
figure lieuDe: p lorsqueBouge: mobile
```

7.2.11 Vecteur

`<WrpCurve> vecteurOrigine:extremite:` *o e* [Méthode sur `DrGeoFigure`]

o, référence d'un point ou d'un couple de coordonnées, origine du vecteur

e, référence d'un point ou d'un couple de coordonnées, extrémité du vecteur

⇒ référence d'un vecteur

```
| b |
b := figure point: 0@5.
figure vecteurOrigine: b extremite: -1 @ -2
```

`<WrpCurve> vecteur:` *p* [Méthode sur `DrGeoFigure`]
p, référence d'un point ou d'un couple de coordonnées, coordonnées du vecteur

⇒ référence d'un vecteur

```
| p |
p := figure point: 5@5.
figure vecteur: p.
figure vecteur: -5 @ -5
```

7.2.12 Valeur numérique

`<Point> coordonnees` [Méthode sur `WrpPoint`]
⇒ coordonnées (statiques) d'un point

```
| c p |
p := pointSurLigne: segment a: 0.5.
c := p coordonnees.
c x
```

`<WrpValue> abscisseDe:` *item* [Méthode sur `DrGeoFigure`]
item, un point ou un vecteur

⇒ abscisse (dynamique) de *item*

```
| m x |
m := figure milieuDe: 10@5 et: 7@8.
x := figure abscisseDe: m
```

`<WrpValue> ordonneeDe:` *item* [Méthode sur `DrGeoFigure`]
item, un point ou un vecteur

⇒ ordonnée (dynamique) de *item*

```
| m x |
m := figure milieuDe: 10@5 et: 7@8.
x := figure ordonneesDe: m
```

`<WrpValue> valeurLibre:` *v* [Méthode sur `DrGeoFigure`]
v, la valeur initiale du nombre

⇒ référence d'un nombre libre

```
v := figure valeurLibre: (-1 arcCos)
```

`valeur:` *unNumbre* [Méthode sur `WrpValue`]
aNumber, un nombre

Modifie la valeur d'un objet nombre libre

```
v := figure valeurLibre: 3.
v valeur: Float pi
```

`<WrpValue> longueurDe:` *item* [Méthode sur `DrGeoFigure`]
item, référence d'un segment, cercle, arc ou vecteur

⇒ référence d'un nombre, longueur de l'item

```
figure longueurDe: v1
```

`<WrpValue> distanceDe:a:` *item point* [Méthode sur `DrGeoFigure`]
item, référence d'une droite ou d'un point

point, référence d'un point

⇒ référence d'un nombre, distance entre deux points ou un point et une droite

```
d := figure droitePassantPar: 0@0 et: 1@1.
figure distanceDe: d a: 12@2.
figure distanceDe: 0@0 a: 12@2
```

`<WrpValue> penteDe:` *droite* [Méthode sur `DrGeoFigure`]

droite, référence d'une droite

⇒ référence d'un nombre, pente de la droite

```
| p |
d := figure droitePassantPar: 0@0 et: 3@8.
p := figure pendteDe: d
```

7.2.13 Angle

`<WrpValue> angleCentre:de:a` *a b c* [Méthode sur `DrGeoFigure`]

a, référence d'un point, sommet de l'angle

b, référence d'un point

c, référence d'un point

⇒ référence d'un angle orienté bac dont la mesure appartient à [0 ; 360[.

```
figure angleCentre: 0@0 de: 3@1 a: -2@3
```

`<WrpValue> angleVecteur:et:` *v1 v2* [Méthode sur `DrGeoFigure`]

v1, référence d'un vecteur

v2, référence d'un vecteur

⇒ référence d'un angle orienté, formé par les deux vecteurs, dont la mesure appartient à [-180 ; 180[.

```
| v1 v2 a |
v1 := figure vecteurOrigine: a extremite: b.
v2 := figure vecteurOrigine: a extremite: c.
a := figure angleVecteur: v1 et: v2
```

`<WrpValue> angleGeometriqueCentre:de:a` *a b c* [Méthode sur `DrGeoFigure`]

a, référence d'un point, sommet de l'angle

b, référence d'un point

c, référence d'un point

⇒ référence d'un angle géométrique bac dont la mesure appartient à [0 ; 180[.

```
figure angleGeometriqueCentre: a de: b a: c
```

7.2.14 Équation

`<WrpValue> equationDe:` *item* [Méthode sur `DrGeoFigure`]

item, référence d'une droite ou d'un cercle

⇒ référence d'une équation de la droite ou du cercle

```
| e d |
d := figure droitePassantPar: 0@0 et: 15@13.
e := figure equationDe: e
```

7.2.15 Texte

`<WrpText> texte:` *string* [Méthode sur `DrGeoFigure`]
string, une chaîne de caractères

⇒ référence d'un objet texte positionné arbitrairement

```
figure texte: 'Hello'
```

`<WrpText> texte:a` *string point* [Méthode sur `DrGeoFigure`]
string, une chaîne de caractères

point, un couple de coordonnées

⇒ référence d'un objet texte positionné à aPoint

```
figure texte: 'Hello,
  je suis heureux !' a: 0@0
```

`texte:` *string* [Méthode sur `WrpText`]
string, une chaîne de caractères

Modifie le texte d'un objet texte

```
monTexte := figure texte: 'Hello'.
monTexte texte: 'Au revoir'
```

7.2.16 Style et attribut

Lecture des attributs

`<Point> coordonnees` [Méthode sur `WrpPoint`]
⇒ instance de `Point`, coordonnées du point

`<Number> x` [Méthode sur `WrpPoint`]
⇒ instance de `Number`, abscisse du point

`<Number> y` [Méthode sur `WrpPoint`]
⇒ instance de `Number`, ordonnée du point

```
a := figure point: 0@10.
figure assert: [a y = 10].
figure assert: [a coordonnees = (0@10)]
```

Modification des attributs

Pour modifier les attributs d'un objet déjà créé, nous lui envoyons le message approprié. La modification des attributs se fait donc toujours à posteriori.

`couleur:` *uneCouleur* [Méthode sur `WrpItem`]
uneCouleur, une instance de `Color`, voir ses méthodes de classe pour des définitions existantes : `Color black, Color red, Color blue, Color orange, Color yellow,...`

Modifie la couleur d'un item

```
pointA couleur: Color green
```

`couleurFond:` *uneCouleur* [Méthode sur `WrpText`]
unCouleur, une instance de `Color`

Modifie la couleur d'arrière plan d'un texte

```
monTexte couleurFond: Color green
```

`nommer:` *string* [Méthode sur `WrpItem`]

string, une chaîne de caractères

Renomme un item

```
segment nommer: '[AB]'
```

`textPositionDelta:` *vecteur* [Méthode sur `MathItemCostume`]

vecteur, une instance de `Point`

Modifie la position de l'étiquette d' un item relativement à sa position de référence.

```
pointA nommer: 'A'.
point costume textPositionDelta: -20 @ -20.
```

`cacher` [Méthode sur `WrpItem`]

Masque un item.

`montrer` [Méthode sur `WrpItem`]

Montre un item.

`fin` [Méthode sur `WrpCurve`]

Donne une épaisseur fine à une ligne (droite, demi-droite, cercle, lieu, etc.).

```
circle fin
```

`normal` [Méthode sur `WrpCurve`]

Donne une épaisseur normale à une ligne (droite, demi-droite, cercle, lieu, etc.). C'est l'épaisseur par défaut.

```
arc normal
```

`epais` [Méthode sur `WrpCurve`]

Donne une épaisseur large à une ligne (droite, demi-droite, cercle, lieu, etc.).

```
polygon epais
```

`plein` [Méthode sur `WrpCurve`]

Donne un style de trait continue, plein, à une ligne (droite, demi-droite, cercle, lieu, etc.).

```
polygon plein
```

`tiret` [Méthode sur `WrpCurve`]

Donne un style de trait en tirets à une ligne (droite, demi-droite, cercle, lieu, etc.).

```
polygon tiret
```

`pointille` [Méthode sur `WrpCurve`]

Donne un style de trait en pointillés à une ligne (droite, demi-droite, cercle, lieu, etc.).

```
arc pointille
```

`flecheDebut` [Méthode sur `wrpFinitCurve`]

Ajoute à un **arc** ou un **segment** une flèche en début de ligne.

```
| figure segment |
figure := DrGeoFigure nouveau.
segment := figure segmentDe: 0@0 a: 5@1.
segment flecheDebut
```

`flecheFin` [Méthode sur `wrpFinitCurve`]

Ajoute à un **arc** ou un **segment** une flèche en fin ligne.

```
segment flechefin
```

`fleches` [Méthode sur `wrpFinitCurve`]
Ajoute à un **arc** ou un **segment** des flèches en début et en fin de ligne.

```
| figure arc |
figure := DrGeoFigure nouveau.
arc := figure arcDe: 0@0 a: 5@3 passantPar: 2@1.
arc fleches
```

`marquerAvecCercle` [Méthode sur `wrpSegment`]
Marque – codage – un segment avec un cercle.

```
segment marquerAvecCercle
```

`marquerAvecDisque` [Méthode sur `wrpSegment`]
Marque – codage – un segment avec un Disque.

```
segment marquerAvecDisque
```

`marquerAvecSimpleTrait` [Méthode sur `wrpSegment`]
Marque – codage – un segment avec un trait.

```
segment marquerAvecSimpleTrait
```

`marquerAvecDoubleTrait` [Méthode sur `wrpSegment`]
Marque – codage – un segment avec un double trait.

```
segment marquerAvecDoubleTrait
```

`marquerAvecTripleTrait` [Méthode sur `wrpSegment`]
Marque – codage – un segment avec un triple trait.

```
segment marquerAvecTripleTrait
```

`marquerAucun` [Méthode sur `wrpSegment`]
Supprime toute marque d'un segment. Cette fonctionnalité sera rarement nécessaire car les segments nouvellement créés ne sont pas marqués.

```
segment marquerAucun
```

`croix` [Méthode sur `WrpPoint`]
Donne une forme en croix à un point.

```
a := figure point: 0@0.
a croix
```

`rond` [Méthode sur `WrpPoint`]
Donne une forme en rond à un point.

```
a rond
```

`carre` [Méthode sur `WrpPoint`]
Donne une forme carrée à un point.

```
a carre
```

`small` [Méthode sur `WrpPoint`]
Donne une petite taille à un point.

```
a small
```

`large` [Méthode sur `WrpPoint`]
Donne une taille large à un point.

```
a large
```

`bloquer` [Méthode sur `WrpItem`]
Bloque un item à sa position actuelle, pour peu que cela ait un sens.

```
| figure cercle |
figure := DrGeoFigure nouveau.
cercle := figure cercleCentre: 0@0 passantPar: 5@0.
figure := segmentDe: 0@0 a: (figure pointSurLigne: cercle a: 0.2).
(figure point: 0@0) bloquer
```

`debloquer` [Méthode sur `WrpItem`]
Débloque un item de sa position actuelle, pour peu que cela ait un sens. Cette fonctionnalité est rarement nécessaire car les items nouvellement créés sont débloqués par défaut.

```
| figure  |
(figure point: 0@0) debloquer
```

`deplacerA:` *point* [Méthode sur `WrpItem`]
point, couple de coordonnées
Déplace un point ou une valeur à la position donnée, pour peu que cela ait un sens.

```
| a |
a := figure point: 0@0.
a deplacerA: 5@5.
figure actualiser
```

7.2.17 Autres méthodes

La classe `DrGeoFigure` propose dans la catégorie `helpers` des méthodes supplémentaires pour faciliter la réalisation de figures interactives complexes.

`courbeDe:de:a:` *block x0 x1* [Méthode sur `DrGeoFigure`]
block, un bloc de code à un argument décrivant une fonction
x0, nombre, abscisse inférieure de la courbe
x1, nombre, abscisse supérieure de la courbe
Affiche la courbe représentative de la fonction décrite par le bloc de code de x0 à x1.

```
| figure |
figure courbeDe: [:x| x * x] de: -3 a: 3
```

`<BlockClosure> decimal:a:min:max` *f1 p f2 f3* [Méthode sur `DrGeoFigure`]

`<BlockClosure> decimal:a:min:max:nom:` *f1 p f2 f3 s* [Méthode sur `DrGeoFigure`]

`<BlockClosure> decimal:a:min:max:nom:affciherValeur:` *v1 p v2 v3 s b* [Méthode sur `DrGeoFigure`]

`<BlockClosure> entier:a:min:max` *v1 p v2 v3* [Méthode sur `DrGeoFigure`]

`<BlockClosure> eniter:a:min:max:nom:` *v1 p v2 v3 s* [Méthode sur `DrGeoFigure`]

`<BlockClosure> entier:a:min:max:nom:affciherValeur:` *v1 p v2 v3 s b* [Méthode sur `DrGeoFigure`]
v1, valeur initiale
p, position du bord gauche de la réglette
v2, valeur minimum
v3, valeur maximum

s, nom de la valeur

b, booléen (true|false), affiche ou non la valeur numérique avec le nom sous la forme 'a = 1.2'

⇒ un bloc de code retournant la valeur courante, décimale ou entière, de la réglette

Construis une réglette à la position indiquée avec une plage de valeur dans [f2 ; f3].

```
A := figure decimal: 1 a: -10@4 min: 0 max: 10 nom: 'A'.
F := figure entier: 3 a: -10@3 min: 0 max: 10 nom: 'F' afficherValeur: true.
A value + F value
```

Il existe d'autres variantes, dont certaines pour des nombres entiers.

`exporterVersImage:` *nomFichier* [Méthode sur `DrGeoFigure`]

nomFichier, une chaîne de caractère représentant le chemin et le nom du fichier où exporter la figure

Exporte la figure dans un fichier bitmap au format PNG.

```
| figure |
figure := DrGeoFigure minimal.
figure point: 0@0.
figure afficherAxes.
figure exporterVersImage: '/tmp/Toto.png'.
figure supprimer
```

7.3 Galerie d'exemples de figures Pharo

Pour illustrer l'utilisation des Figures Pharo Dr.Geo, nous vous proposons une petite série d'exemples. Ceux-ci vous montrent leurs importantes possibilités et nous espérons qu'ils seront également une source d'inspiration. Pour chacun de ces exemples, nous donnons le code source Pharo de la figure puis son résultat. Le code source doit être copié dans un espace de travail (...Clic arrière-plan → Outils → Espace de travail...) puis exécuté.

Animer une figure

Ces exemples s'appuient sur la gestion du temps et celle des processus programmés en Pharo.

Un premier exemple simple pour comprendre le principe :

```
| figure p pause |
figure:=DrGeoFigure nouveau.
p := figure point: 0@0.
pause := Delay forSeconds: 0.2.
figure faire: [
   100 foisRepete: [
      p deplacerA: (p coordonnees + (0.1@0)).
      figure actualiser.
      pause wait]]
```

Un deuxième exemple avec une figure plus élaborée :

```
| figure s r u pause |
figure := DrGeoFigure nouveau pleinEcran.
s := figure segmentDe: 0@ -1 a: 4@ -1.
r := figure pointSurLigne: s a: 0.8.
s := figure segmentDe: 0@0 a: 0@1.
u := figure pointSurLigne: s a: 0.7.
u rond petit; couleurr: Color blue.
```

```
1 a: 100 faire: [:n|
   u := figure
      point: [:parents| |y t|
         y := parents first y.
         t := parents second x.
         (n / 5) @ t * y * (1 - y)]
      parents:  {u . r}.
   u rond petit; couleur: Color blue].
pause := Delay forSeconds: 0.1.
figure faire: [
   0 a: 1 par: 0.05 faire: [:x |
      r mathItem setCurveAbscissa: x.
      figure actualiser.
      pause wait]]
```

Triangle de Sierpinski

Cet exemple s'appuie largement sur un bloc de code récursif.

```
| triangle c |
c := DrGeoFigure nouveau.
triangle := [].

triangle := [:s1 :s2 :s3 :n |
    c segmentDe: s1 a: s2;
        segmentDe: s2 a: s3;
        segmentDe: s3 a: s1.
    n >0 ifTrue:
        [triangle
            valeur: s1
            valeur: (c milieuDe: s1 et: s2) cacher
            valeur: (c milieuDe: s1 et: s3) cacher
            valeur: n-1.
        triangle
            valeur: (c milieuDe: s1 et: s2) cacher
            valeur: s2
            valeur: (c milieuDe: s2 et: s3) cacher
            valeur: n-1.
        triangle
            value: (c milieuDe: s1 et: s3) cacher
            value: (c milieuDe: s2 et: s3) cacher
            value: s3
            value: n-1.]].
triangle valeur: 0@3 valeur:  4@ -3 valeur: -4@ -3 valeur: 3.
(c point: 0@3) montrer
```

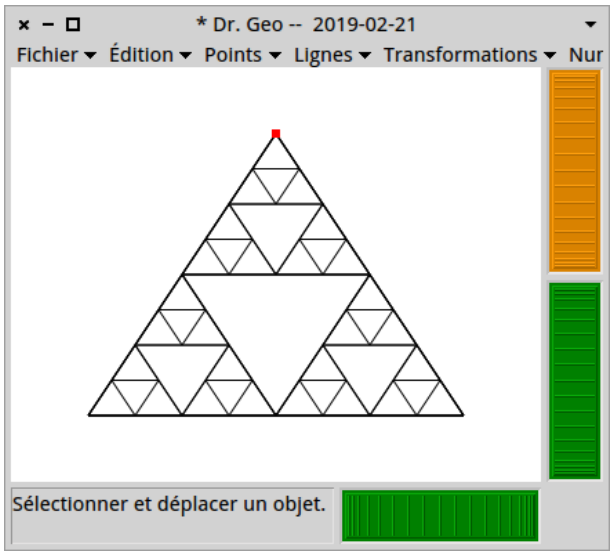

Figure 7.1: Triangle de Sierpinski

Partie III
Programmation

8 Exemples didactiques

Ce chapitre est une aide pour étudier Dr.Geo à partir d'exemples. Contrairement aux chapitres précédents, l'approche est plus concrète par rapport à des situations précises. Le contenu de ce chapitre s'est constitué à partir de contributions diverses.

8.1 Aire et périmètre

Une des possibilités d'utilisation didactique de Dr.Geo consiste dans l'utilisation des scripts Pharo pour résoudre des exercices de géométrie.

Comme exemple, nous allons montrer la solution d'un problème classique mettant en oeuvre le théorème de Pythagore dont le texte est le suivant :

> Soit un trapèze rectangle ABCD où sont connues les bases et la hauteur. Calculer le périmètre et l'aire du trapèze.

Solution :

Commençons par construire la figure dans Dr.Geo qui doit être comme ci-dessous :

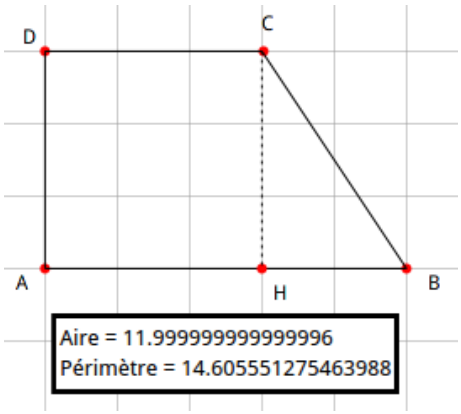

Figure 8.1: Trapèze rectangle

Nous écrirons un script qui, étant donnés les bases et la hauteur, calculera son aire et son périmètre. Le script aura donc comme argument. trois segments, les bases1, base2 et la hauteur. Dans un premier temps, nous définissons le script avec arguments les deux bases et la hauteur du trapèze.

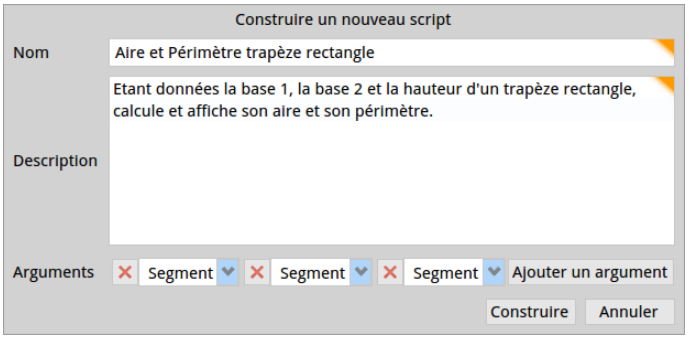

Figure 8.2: Définition du script

Dans ce script, nous écrivons quelques méthodes pour accéder facilement aux longueurs caractéristiques du trapèze rectangle :

base1

```
    ^ self arg1 length
```

base2

```
    ^ self arg2 length
```

hauteur

```
    ^ self arg3 length
```

Puis une méthode calculant son aire :

aire

```
"Calcule l'aire du trapèze"
    ^ (self base1 + self base2) * self hauteur / 2
```

Et une pour calculer le périmètre, nous calculons la longueur BC à l'aide du théorème de Pythagore :

perimetre

```
"Calculer la longueur manquante,
puis le périmètre du trapèze rectangle"
| hb bc |
    hb := (self base1  -self base2 ) abs.
    bc := (hb squared + self hauteur squared) sqrt.
    ^ self base1 + self base2 + self hauteur + bc
```

Enfin la méthode `compute` assemble les deux calculs pour affichage:

```
compute
    ^ 'Aire = ', self aire printString, '
Périmètre = ', self perimetre printString
```

Il n'est pas difficile, si vous suivez le même modèle, de développer d'autres exemples similaires.

8.2 Théorème et conjectures

Les scripts Pharo permettent de résoudre des exercices mais aussi de comprendre de façon plus approfondie l'énonciation des théorèmes et de vérifier des conjectures. Dans cette section nous commençons par analyser le théorème de Tolomeo :

> Étant donné un quadrilatère inscrit dans un cercle la somme du produit des côtés opposés est égale au produit des diagonales.

Nous pouvons construire la figure avec Dr.Geo comme ci-dessous où nous avons implémenté deux scripts qui calculent respectivement la somme du produit des côtés opposés et le produit des diagonales. Nous ne détaillons plus toutes les étapes de création du script, nous donnons uniquement le corps de la méthode `compute`.

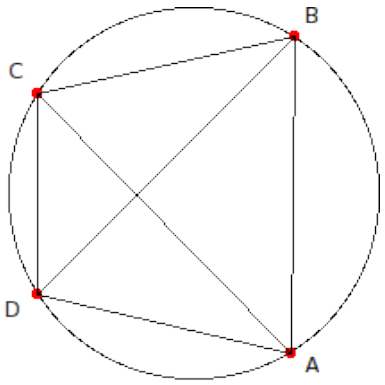

Figure 8.3: Théorème de Tolomeo : quadrilatère convexe

La méthode `compute` du premier script est la suivante :

```
compute
"Choisir quatre côtés consécutifs du quadrilatère"
   | ab bc cd ad |
   ab := self arg1 length.
   bc := self arg2 length.
   cd := self arg3 length.
   ad := self arg4 length.
   ^ (ad * bc) + (ab * cd)
```

La méthode `compute` du second script est :

```
compute
"Choisir les deux diagonales du quadrilatère"
   | ac bd |
   ac := self arg1 length
   bd := self arg2 length
   ^ ac * bd
```

Comme nous pouvons le voir les valeurs retournées par les deux scripts, en accord avec le théorème de Tolomeo, sont les mêmes[1]. Lorsque nous modifions dynamiquement la figure, les valeurs des scripts sont toujours identiques, sauf dans la situation suivante où le quadrilatère perd sa convexité :

[1] Il ne s'agit que d'une vérification numérique.

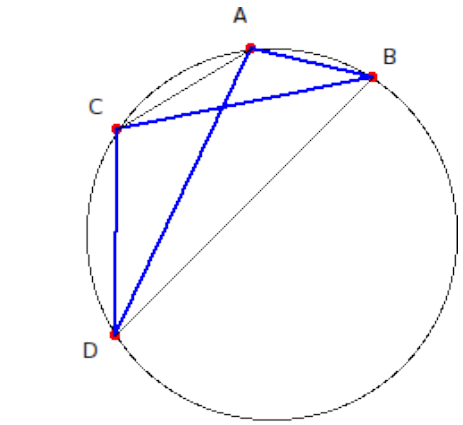

Figure 8.4: Théorème de Tolomeo : quadrilatère non convexe

Dans ce cas le théorème n'est pas vrai et l'énoncé précédent n'est pas bien énoncé, il doit donc être reformulé comme ci-dessous :

> Étant donné un quadrilatère CONVEXE inscrit dans un cercle la somme du produit des côtés opposés est égale au produit des diagonales.

À ce stade les conjectures apparaîssent naturellement : est-ce que le théorème de Tolomeo est valide pour un quadrilatère convexe **non inscrit dans un cercle** ?

Avec Dr.Geo nous pouvons vérifier que cette conjecture est fausse comme le montre la figure suivante, où il nous a suffit de détacher du cercle le point B en l'attrapant avec la touche `Shift` enfoncée.

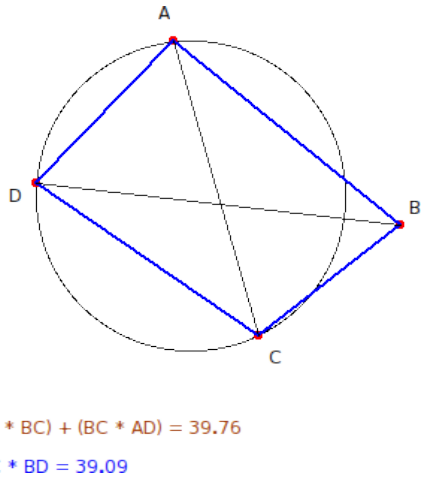

Figure 8.5: Réfutation de la conjecture

Le lecteur n'aura pas de difficulté à utiliser Dr.Geo dans la construction d'exemples didactiques, probablement plus connus, relativement aux théorèmes de Pythagore et d'Euclide.

8.3 Nombre irrationnel

Une construction classique, relative au nombre irrationnel, connue sous le nom de spirale de Teodoro, permet de construire géométriquement la racine carrée de nombres entiers à partir d'un triangle rectangle isocèle.

Considérons le triangle OAB où OA = 1 :

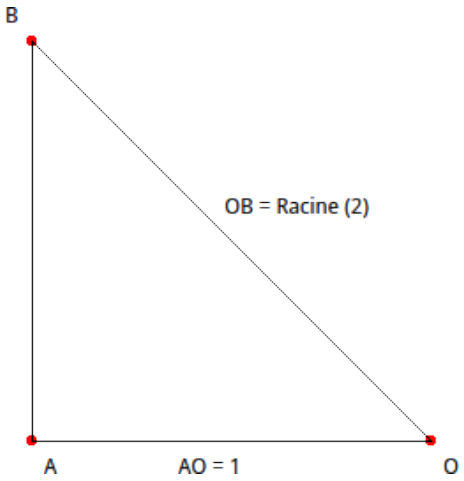

Figure 8.6: Construction de la racine de 2

Par le théorème de Pythagore nous avons OB égale à la racine carrée de 2. Si maintenant, avec la figure, nous construisons un nouveau triangle rectangle en B, avec les côtés OB et BC tel que BC = 1.

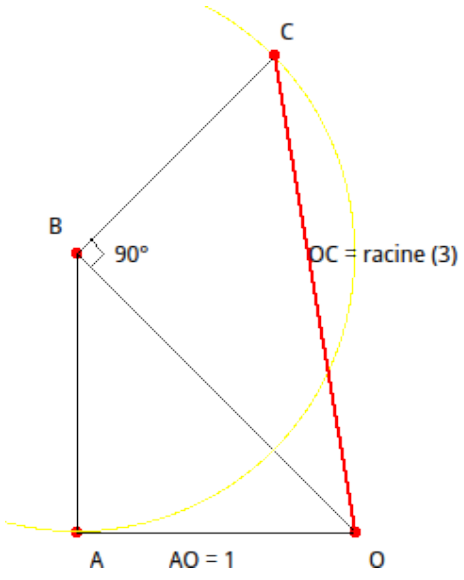

Figure 8.7: Construction de la racine 3

Toujours par le théorème de Pythagore, il est clair que l'hypoténuse OC de OBC a pour longueur la racine carrée de 3. En itérant le processus précédent à l'infini nous obtenons toutes les racines carrées des nombres naturels.

La nature itérative de la construction s'adapte parfaitement à l'utilisation des figures Pharo.

Considérons alors le code suivant :

```
| figure triangle |
figure := DrGeoSketch new fullscreen.
triangle := [:p1 :p2 :p3 :n |
   |s1 s2 s3 perp cercle p4 |
   s1 := figure segment: p1 to: p2.
   s2 := (figure segment: p2 to: p3) color: Color red.
   s3 := figure segment: p3 to: p1.
   perp := (figure perpendicular: s3 at: p3) hide.
   cercle := (figure circleCenter: p3 to: p2) hide.
   p4 := (figure altIntersectionOf: cercle and:  perp) hide.
   n > 0 ifTrue:
      [triangle value: p1 value: p3 value: p4 value: n -1]
].
triangle
   value: 0@0
   value: -1@0
   value: -1@1
   value: 50
```

Le triangle du début est défini à travers les coordonnées seulement par commodité. Le code est la transcription littérale de la procédure itérative décrite précédement. Une fois exécuté par Dr.Geo le code donne la figure ci-dessous.

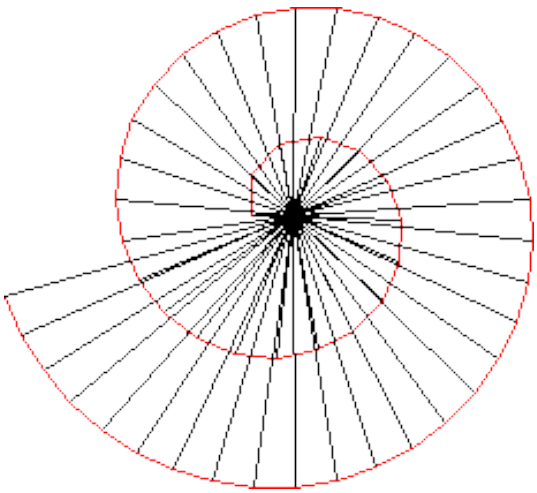

Figure 8.8: Spirale de Teodoro

Les hypoténuses de chaque triangle ont pour longueur les racines carrées des nombres entiers naturels compris entre 2 et 52.

La même spirale avec les éléments intermédiaires de construction montre combien il serait difficile de construire *à la main* une telle figure :

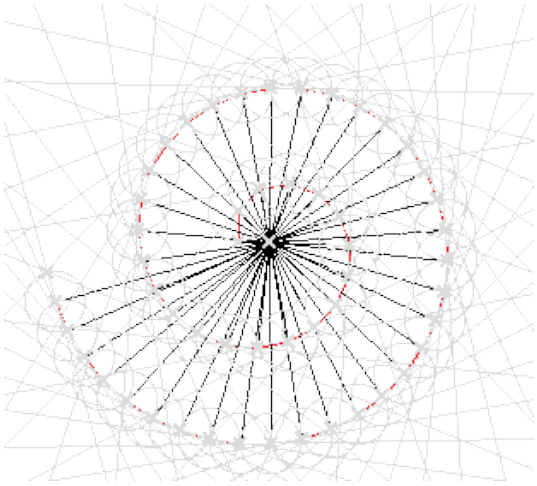

Figure 8.9: Spirale de Teodoro avec les éléments cachés révélés

Comme bonus, nous vous proposons ci-dessous une version animée de la construction de la spirale. Pour cela nous envoyons au canevas les messages `#do:` et `#update`. Ils sont documentés, Voir [SmalltalkSketchMethods], page 51. Noter l'utilisation de la classe `Delay` pour ralentir la construction :

```
| figure triangle delay|
figure := DrGeoSketch new fullscreen.
triangle := [:p1 :p2 :p3  :n |
   |s1 s2 s3 perp cercle p4 |
   s1 := figure segment: p1 to: p2.
   s2 := (figure segment: p2 to: p3) color: Color red.
   s3 := figure segment: p3 to: p1.
   perp := figure perpendicular: s3 at: p3.
   cercle := figure circleCenter: p3 to: p2.
   p4 := figure altIntersectionOf: cercle and:  perp.
   figure update.
   (Delay forMilliseconds: 200) wait.
   perp hide. cercle hide. p4 hide.
   n > 0 ifTrue: [triangle value: p1 value: p3 value: p4 value: n -1]].
figure do: [triangle value: 0@0 value: -1@0 value: -1@1 value: 50]
```

8.4 Spirale de Baravelle

Comme vu précédement, à l'aide de figure Pharo il est possible de construire de façon intuitive et simple des figures pour *visionner* des situations qui en programmation sont récursives – ou cycliques.

Approfondissons cet aspect, en modifiant le code Pharo utilisé pour la construction des nombres irrationnels, afin d'obtenir une figure fameuse de la littérature des mathématiques, à savoir la spirale de Baravelle.

Le code définissant la spirale est le suivant :

```
| figure triangle |
figure := DrGeoSketch new fullscreen.
triangle := [:p1 :p2 :p3  :n |
```

```
    |s1 s2 s3 m  perp cercle p4 |
    s1 := figure segment: p1 to: p2.
    s2 := figure segment: p2 to: p3.
    s3 := figure segment: p3 to: p1.
    m := (figure middleOf: p1 and: p3) hide.
    perp := (figure perpendicular: s3 at: p3) hide.
    cercle := (figure
       circleCenter: p3
       radius: (figure distance: m to: p3) hide) hide.
    p4 := (figure altIntersectionOf: cercle and: perp) hide.
    n > 0 ifTrue:
       [triangle value: m value: p3 value: p4 value: n - 1]].
triangle
    value: (figure point: 0 @ 5)
    value: (figure point: 5 @ 5)
    value: (figure point: 5 @ 0)
    value: 9.
triangle
    value: (figure point: 0 @ -5)
    value: (figure point: -5 @ -5)
    value: (figure point: -5 @ 0)
    value: 9
```

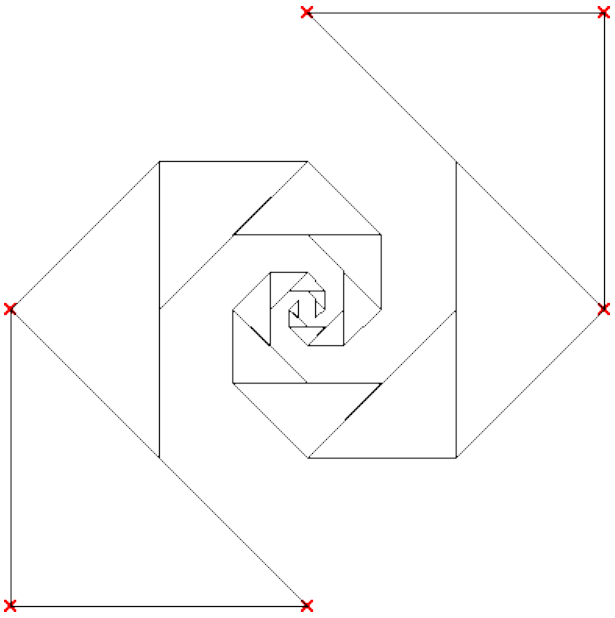

Figure 8.10: La spirale de Baravelle suite à l'exécution du code Pharo

À partir de la figure et du code Pharo correspondant nous percevons bien la nature itérative du mécanisme de construction de la figure. Un problème intéressant que nous laissons au lecteur, consiste à établir à quel moment les deux rameaux de la spirale convergent.

8.5 Catena di Pappo

Une utilisation de base de Figure Pharo de Dr.Geo consiste en la reproduction de figure dont nous connaissons les caractéristiques analytiques. L'exemple de construction que nous proposons est représenté par la fameuse "Catena di Pappo".

Les centres et rayons successifs des cercles qui la constituent ont une expression analytique connue, il est donc aisé de reproduire la figure par programmation.

```
| figure circle a o m|
figure := DrGeoSketch new fullscreen.
circle := [].
circle := [:n |
   |r c p |
   r := (figure freeValue: 15 / (n squared + 6)) hide.
   c := figure point:
      (15 / (n squared + 6) * 5)
      @ (15 / (n squared + 6) * n * 2).
   c small; round.
   p := figure circleCenter: c radius: r.
   n > 0 ifTrue: [circle value: n - 1]].
circle value: 10 .
a := (figure point: 5@0) name: 'A'.
o := (figure point: 0@0) name: 'O'.
m := figure
   middleOf: o
   and: ((figure point: 15@0) name: 'B').
m name: 'M'.
figure
   circleCenter: m to: o;
   circleCenter: a to: o;
   line: a to: o.
```

La figure s'appuie sur un bloc de code récursif pour construire les cercles. Un exercice non trivial, que nous laissons au lecteur, consiste à déterminer une construction à la règle et au compas conduisant à une implémentation itérative.

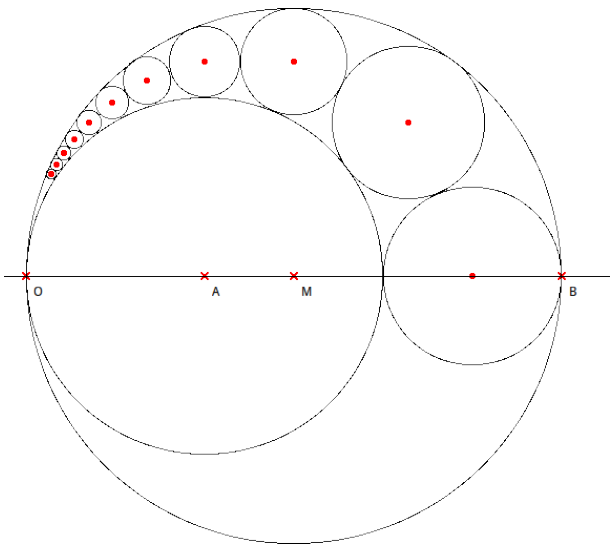

Figure 8.11: Catena di Pappo

8.6 Calcul de PI

Le calcul approximatif de PI a joué un rôle important dans l'histoire des Mathématiques. Les méthodes pour ce type de calcul sont diverses et contiennent souvent des améliorations d'une méthode à l'autre. Nous vous proposons une approche du problème très simplifiée. Cette approche a toutefois l'avantage de montrer l'essence même de la méthodologie.

Nous commençons avec la construction d'un hexagone régulier inscrit dans un cercle.

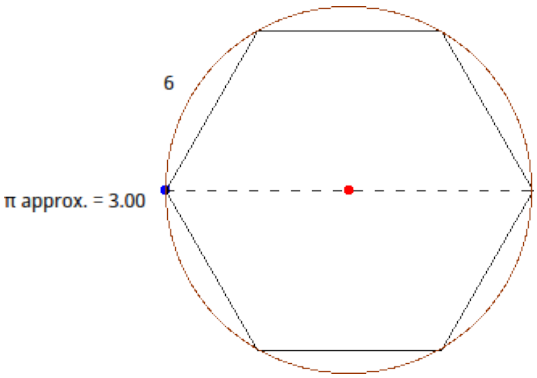

Figure 8.12: Hexagone régulier inscrit

L'idée de la méthode consiste dans un premier temps à approximer la longueur du cercle avec le périmètre P0 de l'hexagone et de calculer une approximation de PI quotient de P0 par le diamètre du cercle. Clairement l'approximation de PI obtenue sera de 3.

Lors d'une deuxième étape nous pouvons, en utilisant Dr.Geo, construire un côté du dodécagone inscrit dans le cercle. Nous calculons son périmètre P1 ainsi qu'une approximation successive de PI comme quotient de P1 par le diamètre.

Un petit script est écrit pour calculer ce quotient à partir du polygone et d'un diamètre du cercle :

```
compute
"Approximation de PI à partir d'un polygone régulier
dans un cercle.
Sélectionner: le polygone et un diamètre du cercle"
   ^ self arg1 length / self arg2 length
```

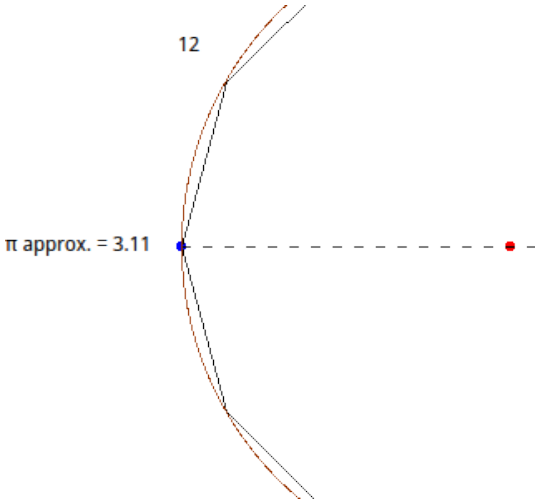

Figure 8.13: Approximation de PI

En augmentant le nombre de côtés du polygone régulier inscrit nous obtenons de meilleures approximations.

9 Outils du développeur

Du fait de son intégration dans l'environnement Pharo, Dr.Geo recèle quelques génies, nombres de ceux-ci sont cachés à la vue de l'utilisateur. Ce n'est pas tant le souhait d'en restreindre l'accès mais plutôt le souci d'alléger la charge cognitive avec une interface simple. Comme nous allons le voir dans les sections suivantes, ces génies s'invoquent par raccourcis clavier, commandes de menu ou codes Pharo.

Dans cette section nous présentons quelques outils à utiliser lors de l'écriture de scripts ou de figures programmées : l'espace de travail, le débogueur, l'inspecteur, etc.

9.1 Espace de travail

Exécuter le code d'une figure

Pour l'afficher ...Clic arrière-plan → Outils → Espace de travail... ou cliquer sur l'arrière-plan et faire *Alt-k*[1].

Un espace de travail – *Playground* – ressemble à s'y méprendre à un éditeur de texte. Mais c'est en fait une console d'édition de code Pharo : pour écrire, compiler et exécuter du code source, il est bien sûr possible d'y coller un code copié ailleurs. Après l'invocation d'un espace de travail[2], y coller le code source de la figure programmée ci-dessous[3] :

```
| figure fonction p integrale sommets |
fonction := [:x | x * x ].
sommets := OrderedCollection new.
figure := DrGeoSketch new.
p := figure point: -1 @ 0.
p hide.
sommets add: p.
-1 to: 1 by: 0.1 do: [:x |
   p := figure point: x @ (fonction value: x).
   sommets add: p hide].
p := figure point: 1 @ 0.
sommets add: p hide.
integrale := figure polygon: sommets.
integrale color: Color blue.
```

[1] Selon votre système d'exploitation, remplacer *Alt* par *Ctrl*.

[2] Contrairement aux génies, vous pouvez invoquer plusieurs espaces de travail.

[3] Pour coller un texte, essayer avec le raccourci clavier *Ctrl-v* ou depuis le menu contextuel de l'espace de travail (clic droit de souris).

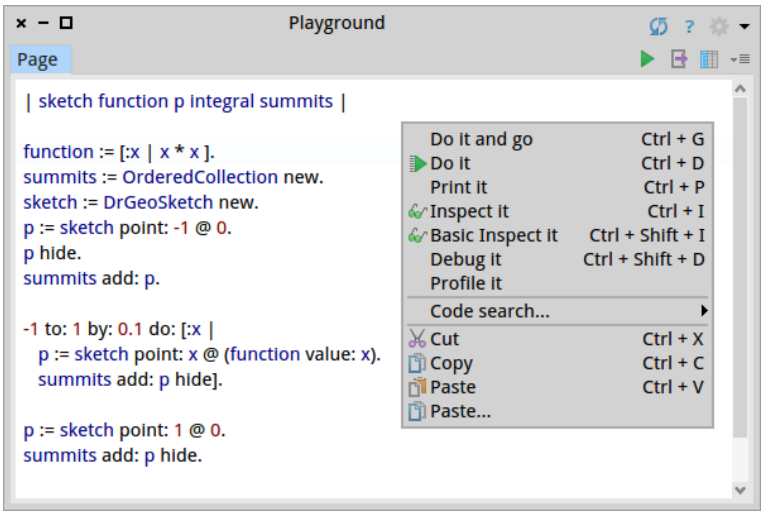

Figure 9.1: Votre espace de travail avec le code source collé et son menu contextuel

Pour compiler et exécuter ce code source, il suffit de le sélectionner à la souris et d'invoquer *Do it(d)* dans le menu contextuel. Ces deux opérations se font également au clavier par un `Ctrl-a` suivi d'un `Ctrl-d`. Vous obtenez alors immédiatement le résultat de cette figure programmée, sous la forme d'une figure interactive dans un canevas Dr.Geo.

Figure 9.2: Résultat de l'exécution du code source : intégrale de la fonction sur [-1 ; 1]

Exécuter et inspecter une figure

Depuis l'espace de travail, il existe une autre façon d'exécuter le code d'une figure et de compléter celle-ci pas à pas. Dans un espace de travail vide saisir ce code :

```
| figure |
figure := DrGeoSketch new.
figure point: 0@0.
figure
```

Cette fois-ci, pour l'exécuter utiliser le bouton vert de lecture, en haut à droite, ou son raccourci `Ctrl-Shift-g`. Cette commande s'appelle *Do it and go*, pour aller où ? Elle exécute le code de la figure et en plus ouvre un inspecteur sur le dernier objet du code, ici `figure`. Un inspecteur est un outil pour scruter les attributs d'un objet – ses variables – et naviguer dans ceux-ci ; l'inspecteur comprend un mini éditeur de code, en bas, pour exécuter des instructions sur l'objet inspecter.

Dans l'inspecteur à droite, en bas, il est alors possible de saisir et d'exécuter du code pour compléter la figure. Dans ce code, la figure est désignée par `self`, littéralement elle-même. Par exemple, pour afficher la grille, créer une droite et un point supplémentaire, saisir[4] :

```
"a DrGeoSketch"
self axesOn.
self line: 0@0 to: 1@0.
self point: 1@1
```

Ce code s'exécute par la combinaison de touches *`Ctrl-a`* puis *`Ctrl-d`* ou avec la souris comme expliqué précédemment.

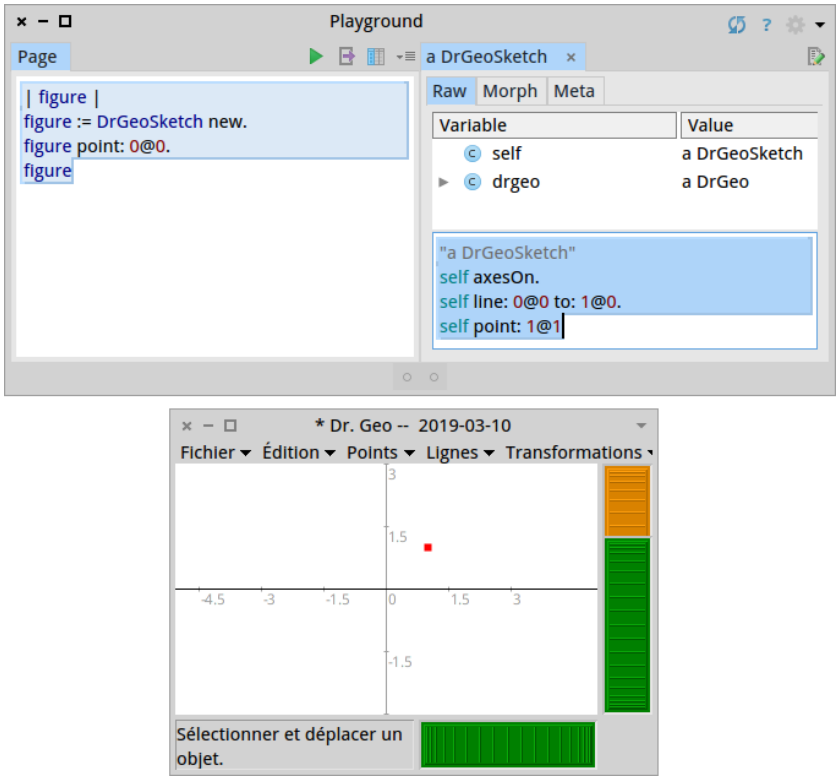

Figure 9.3: Exécution du code, inspection de l'objet `figure` et exécution d'instructions supplémentaires depuis l'inspecteur

Sauver et charger

L'espace de travail sauvegarde automatiquement le code du script édité. Il se recharge avec le bouton *Play pages* à droite de la fenêtre. Pour information, le code est sauvé dans un sous dossier de `DrGeo/Contents/Resources`.

Il est possible de nommer un script de l'espace de travail : double cliquer sur l'onglet *Page* à gauche en haut de l'espace de travail, éditer le nom du script puis valider par *`Enter`*. Pour retrouver ultérieurement le script, écrire son nom dans l'outil *Spotter* : *`Shift-Enter`* pour invoquer *Spotter*, écrire le nom du script, il apparaît alors, le sélectionner et valider par *`Enter`*.

Le code d'une figure se partage également sur Internet, de façon confidentielle. Sven Van Caekenberghe, un développeur de la communauté Pharo, propose un service de partage

[4] `"a DrGeoSketch"` est un commentaire, inutile de le saisir ; il est dans l'exemple car l'inspecteur l'affiche dans son panneau en bas, cela permet de vous repérer.

sur le *cloud*. Depuis l'espace de travail cliquer sur le bouton *Remote publish* à droite, une URL unique est alors copiée dans l'espace tampon du système d'exploitation hôte. Coller celle-ci dans tout document pour partager le code.

Pour charger dans Dr.Geo un code publié sur le *cloud*, le plus simple est d'utiliser l'outil de recherche *Spotter* : ***Shift-Enter*** puis ***Ctrl-v*** pour coller l'URL, puis ***Enter*** ouvrira un espace de travail avec ce code. L'essayer avec cette ressource `http://ws.stfx.eu/8I4GUN1B5W7K`.

9.2 Profileur

Lors de l'exécution d'un code source complexe, exécuter celui-ci par l'intermédiaire du *Profileur* aide à trouver les portions de code consommatrice en temps de calcul. Pour ce faire, dans le menu contextuel de l'espace de travail exécuter le code en invoquant *Profile it*. Le code source est exécuté, le canevas Dr.Geo affiché et en plus la fenêtre du profileur informe l'utilisateur sur le temps d'exécution du code et des méthodes invoquées. C'est un outil remarquable pour naviguer dans l'arbre d'exécution du code et afficher les méthodes consommatrice en temps machine.

Testé avec la code du début de section, le profileur montre que Dr.Geo prend plus de temps à construire la fenêtre que la figure :

```
86.3% {109ms} UndefinedObject>>DoIt
  71.0% {89ms} DrGeoSketch class(Behavior)>>new
  6.5% {8ms} DrGWrappedPoint(DrGWrappedItem)>>hide
  5.6% {7ms} DrGeoSketch>>point:
  1.6% {2ms} DrGeoSketch>>polygon:
  1.6% {2ms} primitives
```

En revanche sur une figure plus complexe, récursive par exemple, le profileur montre que le code consomme davantage de temps lors de la construction elle-même.

Copier-coller dans un espace de travail ce code qui construit la courbe de Sierpinski :

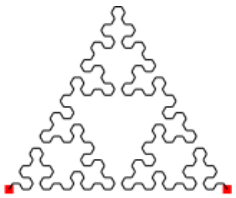

Figure 9.4: Courbe interactive de Sierpinski

```
| canvas dragon |
canvas := DrGeoSketch new.
dragon := [  ].
dragon := [ :a :b :k | | c m n |
   k > 0 ifTrue: [
      c := canvas
         altIntersectionOf: (canvas circleCenter: a to: b) hide
         and: (canvas circleCenter: b to: a) hide.
      c hide.
      m := (canvas middleOf: a and: c) hide.
      n := (canvas middleOf: b and: c) hide.
      dragon value: m value: a value: k - 1.
      dragon value: m value: n value: k - 1.
      dragon value: b value: n value: k - 1].
   k = 0 ifTrue: [ canvas segment: a to: b ].
].
canvas text: 'Sierpinski Dragon' at: 0@0.
```

```
dragon
   value: (canvas point: -4@1.5)
   value: (canvas point: 4@1.5)
   value: 5
```

Puis le sélectionner afin de l'exécuter avec le profileur. Davantage de temps est utilisé pour la construction des côtés de la courbe – méthode `segment:to:` – mais aussi d'autres éléments non visibles comme des intersections entre lignes.

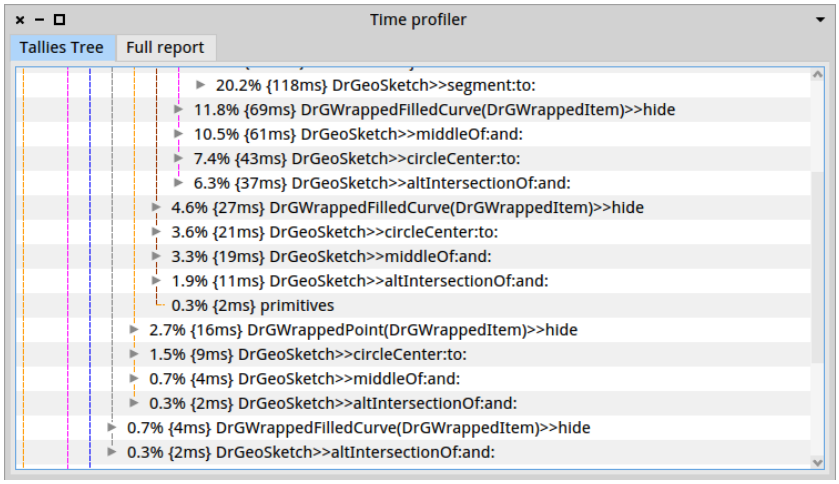

Figure 9.5: Le profileur Dr.Geo à l'issue de la construction de la courbe de Sierpinski

9.3 Débogueur

Dernier raffinement : l'exécution en mode pas à pas du code source. Cela se fait en l'exécutant avec le débogueur, dans le menu contextuel choisir la commande *Debug it*. Le débogueur est invoqué sur la première ligne, le code s'exécute pas à pas avec le bouton *Over*. Dans la partie basse à droite, les variables locales et leur contenu est consultable et modifiable.

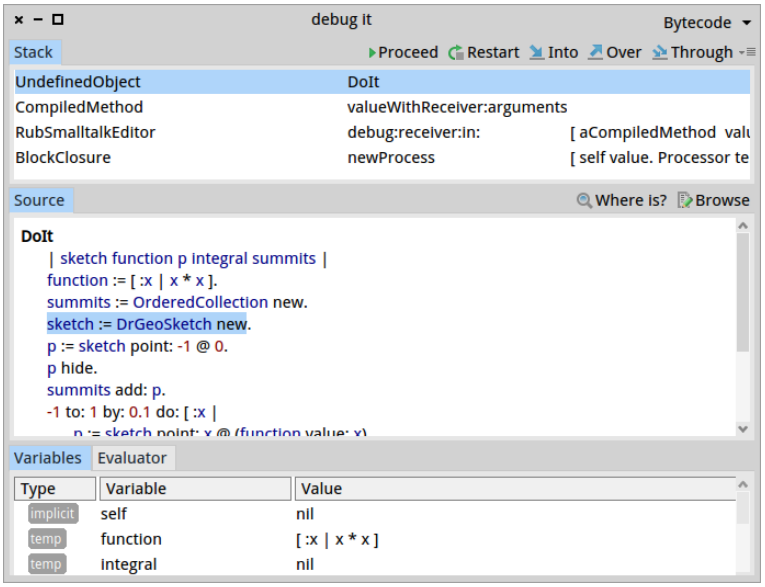

Figure 9.6: Le débogueur Dr.Geo

Quelques précisions concernant l'interface du débogueur s'imposent :

- **Stack.** Ce panneau indique la pile d'exécution des objets invoqués. Ce n'est pas utile lors de l'écriture du code source d'une figure depuis un espace de travail.
- **Proceed.** Ce bouton permet de reprendre l'exécution normale du code.
- **Restart.** Pour reprendre l'exécution pas à pas du programme depuis le début de la méthode, c'est à dire le début du code source de la figure.
- **Into.** Exécuter l'instruction mise en évidence, et suivre le message envoyé dans la méthode invoquée.
- **Over.** Exécuter l'instruction mise en évidence, et reprendre le contrôle après le message envoyé, c'est à dire à l'instruction suivante.
- **Through.** Exécuter l'instruction suivante, ne pas suivre le message envoyé mais exécuter pas à pas les éventuels blocs de code en arguments du message. Utile par suivre l'exécution dans un bloc de code.
- **Source.** Ce panneau contient le code source exécuté avec mise en évidence de l'instruction suivante à exécuter. Il offre également un menu contextuel avec quelques fonctions intéressantes comme "Run to here" pour exécuter le programme jusqu'à la ligne où aura été placé le curseur d'insertion textuelle.

Comme montré dans une section précédente, le débogueur permet l'exécution en mode pas à pas. Il s'invoque aussi à n'importe quel moment avec le raccourci clavier `Alt-.` (Alt + point).

En outre, le débogueur s'enclenche également par programmation, directement dans le code source en ajoutant une ligne `self halt.` Dans notre exemple précédent, nous pouvons modifier le code source comme suit :

```
...
p:=figure point: -1 @ 0.
p hide.
sommets add: p.
self halt.
-1 to: 1 by: 0.1 do: [ :x |
   p:=figure point: x @ (fonction value: x).
   sommets add: p hide].
...
```

Lorsque ce code est compilé et exécuté, le programme est stoppé. Dans la fenêtre qui s'affiche alors, il est demandé de faire un choix :

- **Proceed.** Reprendre l'exécution du programme à partir de l'instruction suivante
- **Abandon.** Stopper l'exécution du programme
- **Debug.** Invoquer le débogueur, une fenêtre identique à la figure précédente s'affiche alors pour exécuter pas à pas le programme et examiner les objets du programme.

9.4 Inspecteur

Avec l'inspecteur, l'utilisateur consulte les attributs d'une instance ou bien le contenu d'une variable.

Dans notre exemple, supposons que nous souhaitions voir le contenu de la collection *sommets*. Dans ce cas rien de plus simple, nous ajoutons une ligne de code où nous envoyons le message `inspect` à *sommets*, l'emplacement où se fait cette invocation n'est pas très important car nous n'avons ni point d'arrêt, ni exécution en mode pas à pas :

```
...
```

```
p:=figure point: -1 @ 0.
p hide.
sommets add: p.
sommets inspect.
-1 to: 1 by: 0.1 do: [ :x |
   p:=figure point: x @ (fonction value: x).
   sommets add: p hide].
...
```

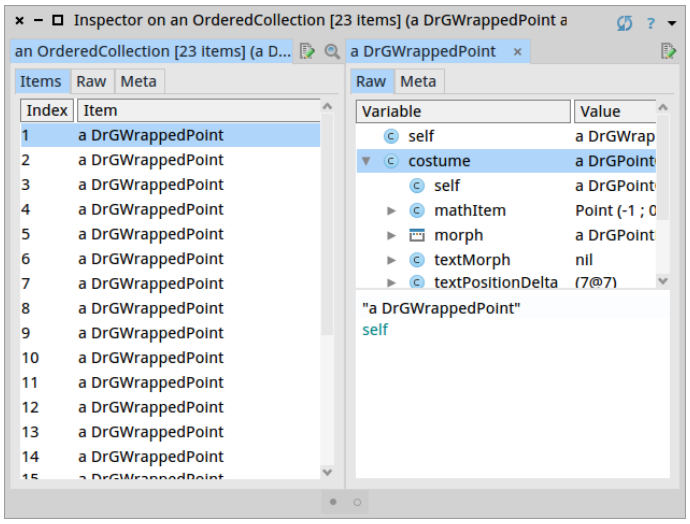

Figure 9.7: L'inspecteur sur la variable *sommets*

Un inspecteur est capable d'examiner autre chose que des attributs d'un objet, par exemple le contenu d'un dossier. Dr.Geo utilise cette fonctionnalité pour parcourir, exécuter, modifier les codes des figures du dossier `DrGeo/SmalltalkSketches`. Faire ...Clic arrière-plan → Outils → Travailler sur un script... un inspecteur s'affiche alors avec une liste de figures programmées. En choisissant une figure, son code source s'affiche dans un panneau à droite.

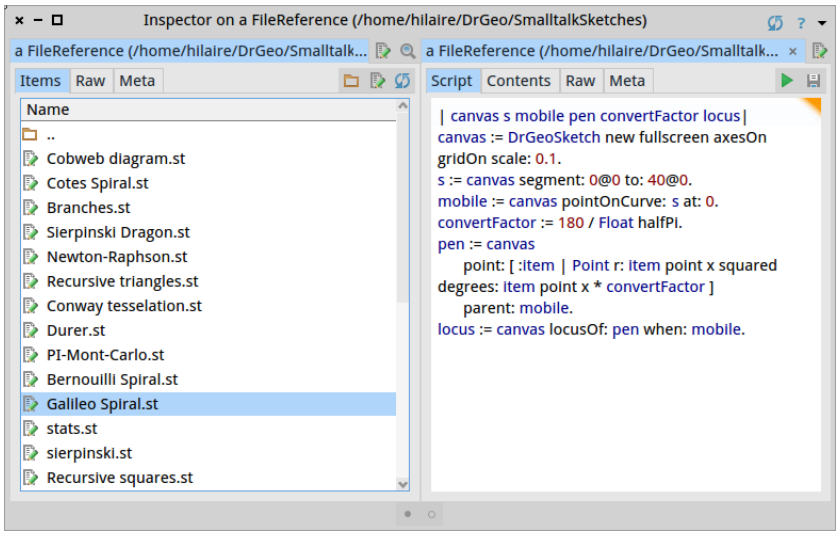

Figure 9.8: Inspecteur et codes source des figures

Choisir *la spirale de Galilée*, puis l'exécuter avec les raccourcis clavier ou les boutons, à l'identique de l'espace de travail.

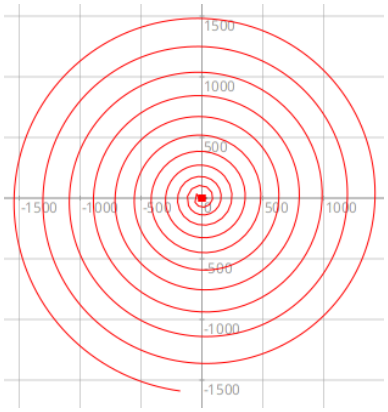

Figure 9.9: Spirale de Galilée

Lorsque le script est modifié dans le panneau à droite, cliquer sur le bouton "Save" à droite ou utiliser le raccourci clavier `Alt-s`. Pour créer un nouveau script, dans le panneau à gauche utiliser le bouton "Create File". Par convention les codes sources Pharo de figure portent l'extension `.st`.

9.5 Chercheur

Le Chercheur est un outil de recherche de sélecteurs (nom d'une méthode), de classes, d'expressions dans le code source de Dr.Geo et de Pharo plus généralement. Il s'appuie sur les capacités réflexives de l'environnement Pharo. Pour ouvrir l'outil faire ...Clic arrière-plan → Outils → Chercheur de méthode...

La recherche se fait par un nom partiel ou encore plus intéressant par un exemple de calcul et de résultat souhaité. L'outil vous affiche alors l'ensemble des réponses correspondantes avec la possibilité de parcourir le code source de chacune d'elles in situ ; en effet il existe souvent plusieurs réponses valides, il convient de choisir celle souhaitée.

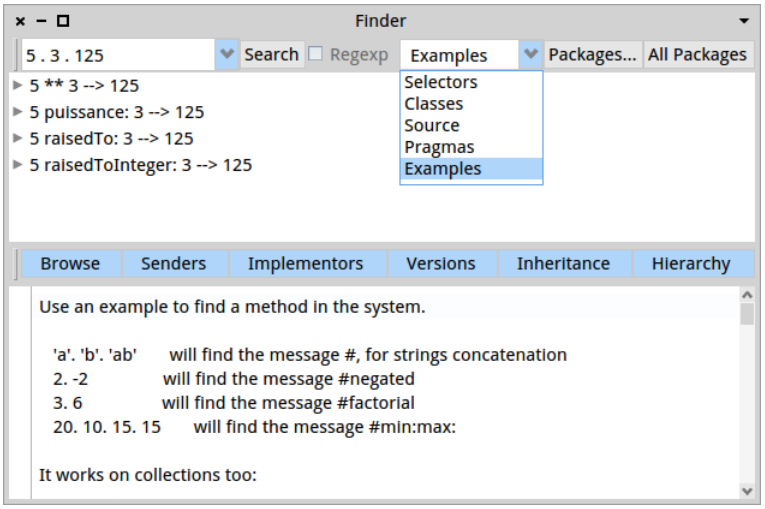

Figure 9.10: Réponse du Chercheur sur un motif de calcul et sa réponse souhaitée

Quelques exemples de recherches sur des éléments de natures différentes :

- **Sélecteur.** Saisir `^square` comme nom de sélecteur, cocher "Regexp" et choisir "Selectors" dans la liste. Ces réponses sont retournées (en cliquant sur l'une d'elle son code source est affiché) :

```
square
squareNorm: (DrGLocus2ptsItem)
squared
squaredDistanceTo: (Point)
```

- **Classe.** Pour trouver les classes modélisant les demi-droites, saisir `DrGRay(.*)Item` comme nom de classe avec "Regex" coché et choisir "Class" dans la liste, le Chercheur retourne cette liste de classe :

```
DrGRay2Item
DrGRayHomothetyItem
DrGRayItem
DrGRayReflexionItem
DrGRayRotationItem
DrGRaySymmetryItem
DrGRayTranslationItem
```

- **Motif de calcul.** Pour trouver comment calculer la puissance 3 d'un nombre, saisir la séquence `5 . 3 . 125`. Elle indique qu'à partir de 5 et 3 nous souhaitons obtenir 125. Choisir "Examples" dans la liste, voici les possibilités trouvées :

```
5 ** 3 --> 125
5 puissance: 3 --> 125
5 raisedTo: 3 --> 125
5 raisedToInteger: 3 --> 125
```

  Noter le sélecteur `puissance:` en *Français* propre à Dr.Geo.

  Un autre exemple amusant sur l'addition des couleurs, saisir le motif `Color red . Color green . Color yellow`, le Chercheur retourne :

```
Color green + Color red --> Color yellow
Color red + Color green --> Color yellow
```

  Il suffit d'utiliser le sélecteur `+` pour additionner des couleurs !

9.6 Spotter

Spotter est un outil de recherche de contenu dans l'environnement Dr.Geo. Il s'active par le raccourci clavier *`Shift-Enter`* et il suffit alors de saisir un mot de recherche. Il est capable de rechercher dans les menus, le code source, la documentation intégrée, voire des scripts de figure Pharo sur internet comme montré précédemment.

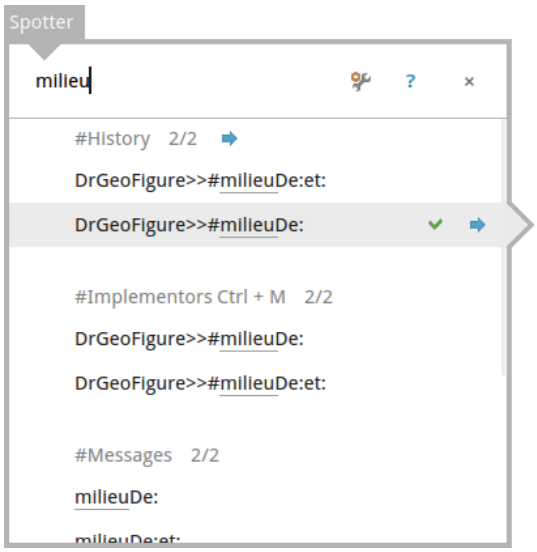

Figure 9.11: Spotter l'outil de recherche

Partie IV
Annexes

Annexe A GNU Free Documentation License

GNU Free Documentation License

Version 1.2, November 2002

59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD

and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the

Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Annexe B Liste des figures

Annexe C Index conceptuel

Deux index sont proposés : un index sur les concepts ci-dessous et, plus loin, un deuxième sur les methodes pour la programmation des scripts et des figures.

É

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

V

Z

Annexe D Index des méthodes

A

B

C

D

E

F

H

I

L

M

N

O

P

R

S

T

V

X

Y