Manuel de KTurtle

Cies Breijs
Anne-Marie Mahfouf
Mauricio Piacentini
Traduction française : Anne-Marie Mahfouf
Traduction française : Ludovic Grossard
Traduction française : Stanislas Zeller

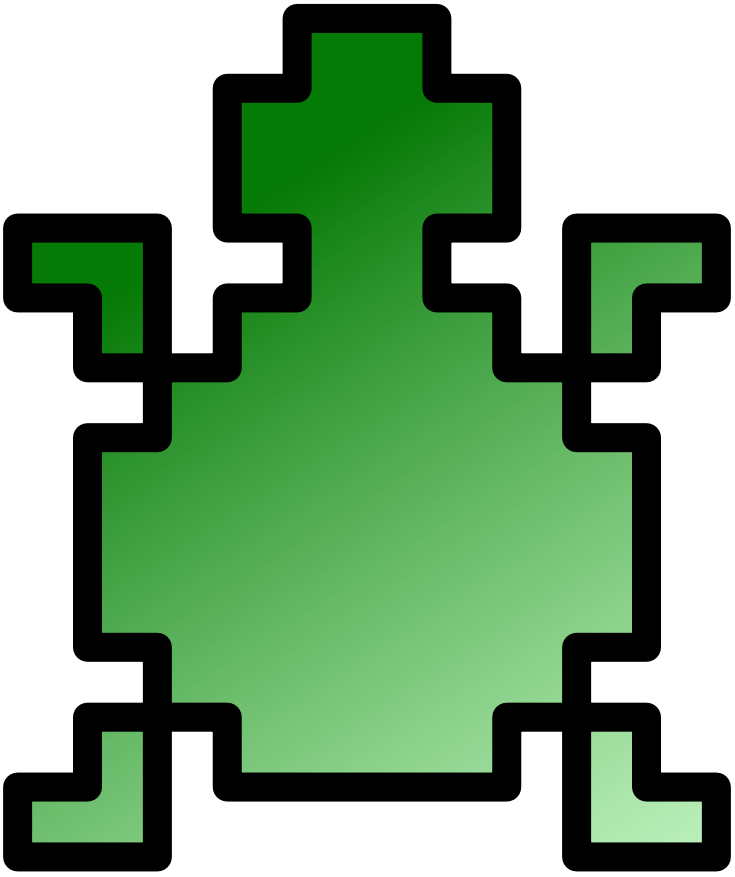


Table des matières

Liste des tableaux

Résumé

KTurtle est un environnement de programmation éducatif dont l'objectif est de rendre l'apprentissage de la programmation aussi facile que possible. Pour ceci, KTurtle rend tous les outils de programmation disponibles depuis l'interface graphique. Le langage de programmation utilisé est TurtleScript, qui permet aux commandes d'être traduites.

Chapitre 1

Introduction

KTurtle est un environnement de programmation à but éducatif utilisant le langage de programmation TurtleScript, un langage de programmation inspiré de Logo. Le but de KTurtle est de rendre la programmation aussi facile et abordable que possible. Cela rend KTurtle facilement utilisable pour enseigner aux enfants les bases des mathématiques, de la géométrie et... de la programmation. Une caractéristique unique de TurtleScript est que les commandes sont souvent traduites dans la langue parlée par le programmeur.

KTurtle tire son nom de "la tortue" qui joue un rôle central dans l'environnement de programmation. L'étudiant va habituellement donner des instructions à la tortue, en utilisant les commandes TurtleScript pour dessiner une figure sur le canevas.

1.1 Qu'est ce que TurtleScript ?

TurtleScript, le langage de programmation utilisé par KTurtle, est inspiré par la famille des langages de programmation Logo. La première version de Logo a été créée par Seymour Papert du Laboratoire d'Intelligence Artificielle du MIT en 1967 comme un dérivé du langage de programmation LISP. Depuis, beaucoup de versions de Logo sont sorties. En 1980, le Logo a pris de l'ampleur, avec des versions existantes pour les systèmes MSX, Commodore, Atari, Apple II et IBM PC. Ces versions existèrent principalement pour des motifs éducatifs. Le MIT tient toujours à jour un site Web sur le Logo contenant une liste de plusieurs implémentations populaires du langage.

TurtleScript partage une caractéristique que l'on trouve dans de nombreuses autres implémentations de Logo : la possibilité de traduire les commandes afin de s'adapter à la langue de l'étudiant. Cette caractéristique facilite l'apprentissage pour les étudiants qui comprennent mal ou pas du tout l'anglais. En plus de ceci, KTurtle dispose de nombreuses autres caractéristiques afin de faciliter les premiers pas de l'étudiant en programmation.

1.2 Caractéristiques de KTurtle

KTurtle possède quelques caractéristiques qui facilitent l'apprentissage de la programmation. En voici quelques-unes :

— Un environnement intégré avec un interpréteur TurtleScript, un éditeur, un canevas et d'autres outils, le tout dans une application unique (dans autres dépendances).

— La possibilité de traduire les commandes TurtleScript en utilisant l'architecture de traduction de KDE.

— TurtleScript gère les fonctions définies par l'utilisateur, la récursion et le changement de type dynamique.

— L'exécution peut être ralentie, mise en pause ou arrêtée à n'importe quel moment.

— un éditeur puissant avec une coloration syntaxique intuitive, la numérotation des lignes, le marquage des erreurs, l'exécution visuelle, et plus encore.

— Le canevas, dans lequel la tortue dessine, peut être imprimé ou enregistré comme une image (PNG) ou comme dessin (SVG) ;

— L'aide contextuelle : de l'aide lorsque vous en avez besoin. Appuyez simplement sur **F2** (ou voyez **Aide** → **Aide sur :**) pour obtenir de l'aide le morceau de code qui se trouve sous le curseur.

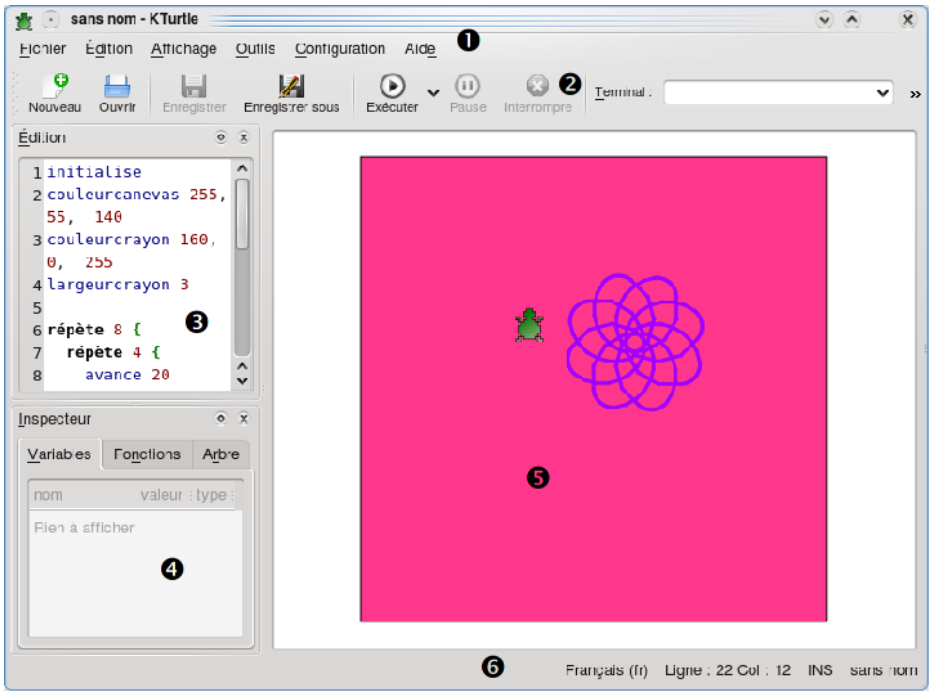
— une boîte de dialogue d'erreur qui fait un lien entre les messages d'erreur et les erreurs dans le programme, et les marque en rouge.

— une terminologie de programmation simplifiée ;

— Des programmes d'exemples intégrés pour faciliter le démarrage. Ces exemples sont traduits en utilisant l'architecture de traduction de KDE.

Chapitre 2

Utiliser KTurtle



La fenêtre principale de KTurtle comprend trois parties principales : l'éditeur (1) sur la gauche où vous saisissez les commandes TurtleScript, le canevas (2) sur la droite où la tortue fait votre dessin, et l'inspecteur (3) qui vous donne des informations lorsque votre programme s'exécute. De plus, vous trouverez la barre de menus (4) de laquelle toutes les actions peuvent être lancées, la barre d'outils (4) qui vous permet de sélectionner rapidement les actions les plus utilisées, le **terminal**, que vous pouvez utiliser pour saisie une commande en une seule ligne pour la tester, et la barre d'état (en bas de la fenêtre) qui vous fournit des informations sur l'état de KTurtle.

2.1 L'éditeur

Dans l'éditeur, vous saisissez les commandes TurtleScript. La plupart de ses fonctionnalités de l'éditeur se trouve dans les menus Fichier et Édition. L'éditeur peut être accroché de chaque côté de la fenêtre principale ou il peut se détacher pour être placé n'importe où sur votre bureau.

Il existe plusieurs façons d'introduire du code dans l'éditeur. Le moyen le plus facile est d'utiliser un exemple. Pour cela, allez dans **Fichier** → **Exemples** et cliquez sur un fichier. Le nom du fichier

va vous indiquer ce qu'illustre l'exemple (par exemple `carré.logo` va dessiner un carré). Le fichier que vous avez choisi va être ouvert dans l'éditeur, vous pouvez ensuite exécuter le code avec **Fichier** → **Exécuter** depuis la barre de menus, ou **Exécuter** depuis la barre d'outils si vous préférez..

Vous pouvez ouvrir des fichiers TurtleScript en choisissant **Fichier** → **Ouvrir...**.

Le troisième moyen est de saisir directement votre propre code dans l'éditeur ou encore de le copier / coller.

2.2 Le canevas

Le canevas est le domaine de la tortue, ici la tortue dessine selon les commandes qu'on lui fournit. Après avoir mis du code dans l'éditeur, et après l'avoir exécuté, deux choses peuvent se produire : soit le code s'exécute correctement et vous verrez sûrement un changement sur le canevas, soit il y a une erreur dans votre code. Dans ce cas l'onglet d'erreurs apparaîtra, vous expliquant quelle erreur vous avez commise.

Vous pouvez zoomer en avant et en arrière sur le canevas avec la molette de la souris.

2.3 L'inspecteur

L'inspecteur vous donne des informations sur les variables, les fonctions que vous avez créées et affiche l'arborescence du code pendant que le programme fonctionne.

L'inspecteur peut être attaché sur chaque bord de la fenêtre principale ou il peut être détaché et placé n'importe où sur votre bureau.

2.4 La barre d'outils

Vous pouvez ici atteindre rapidement les actions les plus utilisées. La barre d'outils contient également le **terminal**, dans lequel vous pouvez appeler rapidement des commandes. Ce peut être utile si vous souhaitez tester une commande sans modifier le contenu de l'éditeur.

Vous pouvez configurer la barre d'outils à votre guise en utilisant **Configuration** → **Configurer la barre d'outils...**

2.5 La barre de menus

Dans la barre de menus, vous trouverez toutes les actions de KTurtle. Elles se trouvent dans les menus suivants : **Fichier**, **Édition**, **Canevas**, **Exécuter**, **Outils**, **Configuration** et **Aide**. Ce chapitre les décrit tous.

2.5.1 Le menu Fichier

Fichier → **Nouveau (Ctrl-N)**
: Crée un nouveau fichier TurtleScript vide.

Fichier → **Ouvrir... (Ctrl-O)**
: Ouvre un fichier TurtleScript.

Fichier → Récemment ouvert(s)
Ouvre un fichier TurtleScript qui a récemment été ouvert.

Fichier → Exemples
Ouvre les programmes d'exemple de TurtleScript. Les exemples sont dans votre langue natale que vous pouvez choisir dans **Configuration → Langue du code**.

Fichier → Obtenir d'autres exemples...
Ouvre l'**Installateur d'extensions** pour télécharger d'autres fichiers TurtleScript depuis Internet.

Fichier → Enregistrer (Ctrl-S)
Enregistre le fichier TurtleScript actuellement ouvert.

Fichier → Enregistrer sous...
Enregistre le fichier TurtleScript actuellement ouvert vers un emplacement spécifique.

Fichier → Exporter en HTML...
Exporte le contenu actuel de l'éditeur dans un fichier HTML, en y incluant la coloration syntaxique.

Fichier → Imprimer... (Ctrl-P)
Imprime le code courant dans l'éditeur

Fichier → Quitter (Ctrl-Q)
Quitte KTurtle.

2.5.2 Le menu Édition

Édition → Annuler (Ctrl-Z)
Annule la dernière modification dans le code. KTurtle a un nombre de possibilités d'annulation infini !

Édition → Refaire (Ctrl-Maj-Z)
Refait une modification annulée dans le code.

Édition → Couper (Ctrl-X)
Coupe le texte sélectionné de l'éditeur dans le presse-papiers.

Édition → Copier (Ctrl-C)
Copie le texte sélectionné depuis l'éditeur dans le presse-papiers.

Édition → Coller (Ctrl-V)
Colle le texte du presse-papiers dans l'éditeur.

Édition → Tout sélectionner (Ctrl-A)
Sélectionne tout le texte depuis l'éditeur.

Édition → Chercher... (Ctrl-F)
Grâce à cette action, vous pouvez trouver des mots dans le code.

Édition → Poursuivre la recherche (F3)
Utiliser ceci pour trouver la prochaine occurrence du mot que vous cherchez.

Édition → Poursuivre la recherche (Maj-F3)
Utiliser ceci pour trouver la prochaine occurrence de la phrase que vous cherchez.

Édition → Écraser / Insérer (Inser)
Bascule entre les modes « Insérer » et « Écraser ».

2.5.3 Le menu Canevas

Canevas → Exporter sous forme d'image (PNG)...
Exporte le contenu actuel du canevas comme image matricielle au format PNG (*Portable Network Graphics*).

Canevas → Exporter en dessin (SVG)...
Exporte le contenu actuel du canevas comme dessin vectoriel au format SVG (*Scalable Vector Graphics*).

Canevas → Imprimer le canevas
Imprime le contenu actuel du canevas.

2.5.4 Le menu Exécuter

Exécuter → Exécuter (F5)
Démarre l'exécution des commandes dans l'éditeur.

Exécuter → Pause (F6)
Suspend l'exécution. Cette action n'est disponible que lorsque les commandes s'exécutent.

Exécuter → Interrompre (F7)
Arrête l'exécution. Cette action n'est disponible que lorsque les commandes s'exécutent.

Exécuter → Pleine vitesse
Présente une liste des vitesses d'exécution possibles : **Pleine vitesse (pas de mise en évidence ni d'inspecteur)**, **Pleine vitesse**, **Lente**, **Plus lente**, **La plus lente** et **Pas à pas**. Lorsque la vitesse d'exécution est réglée sur **Pleine vitesse** (valeur par défaut), vous pouvez difficilement vous rendre compte de ce qu'il se passe. Ce comportement est parfois voulu, mais il arrive que vous souhaitiez suivre l'exécution du programme. Dans ce cas, utilisez une vitesse d'exécution **Lente**, **Plus lente**, **La plus lente**. Lorsqu'un des modes lents est choisi, la position actuelle dans le programme est affichée dans l'éditeur. **Pas à pas** exécutera une commande à la fois.

2.5.5 Le menu Outils

Outils → Choisir la direction...
Cette action ouvre une boîte de dialogue de choix de direction.

Outils → Sélecteur de couleur...
Cette action ouvre une boîte de dialogue de choix de couleur.

2.5.6 Le menu Configuration

Configuration → Langue du code
Choisit la langue pour le code.

Configuration → Afficher l'éditeur (Ctrl-E)
Affiche ou masque l'éditeur.

Configuration → Afficher l'inspecteur (Ctrl-I)
Affiche ou masque l'inspecteur.

Configuration → Afficher les erreurs...
Affiche ou cache l'onglet **Erreur** avec la liste des erreurs qui provient du code en cours. Si cette option est activée, appuyez sur le **Canevas** pour afficher à nouveau la tortue.

Configuration → Afficher les numéros de ligne (F11)
Cette action vous permet d'afficher les numéros de lignes dans l'éditeur. Ceci peut être très pratique pour trouver des erreurs.

Configuration → Afficher la barre d'outils...
Affiche ou cache la barre d'outils

Configuration → Afficher la barre d'état
Affiche ou cache la barre d'état

Configuration → Configurer les raccourcis clavier...
Affiche une boîte de dialogue standard de KDE pour configurer les raccourcis clavier.

Configuration → Configurer les barre d'outils...
La boîte de dialogue standard de KDE pour configurer les barre d'outils.

2.5.7 Le menu Aide

Aide → Manuel utilisateur de KTurtle (F1)
Exécute le système d'aide de KDE en commençant par les pages d'aide de KTurtle (ce document).

Aide → Qu'est-ce que c'est ? (Maj+F1)
Change le curseur de la souris en une combinaison flèche et point d'interrogation (?). Cliquer sur des éléments dans KTurtle ouvrira une fenêtre d'aide (s'il en existe une pour cet élément particulier), expliquant la fonction de l'élément en question.

Aide → Rapport de bogue...
Ouvre une boîte de dialogue de rapport de bogue où vous pouvez signaler un bogue ou effectuer une requête de fonctionnalité sous la forme d'une liste de souhaits ("wishlist").

Aide → À propos de KTurtle
Affiche des informations sur la version et sur l'auteur.

Aide → À propos de KDE
Affiche la version de KDE et d'autres informations de base.

Aide → Aide sur : (F2)
C'est une fonctionnalité particulièrement utile. Elle vous fournit de l'aide sur le code qui se trouve sous le curseur dans l'éditeur. Ainsi, par exemple, vous avez utilisé la commande **`écris`** dans votre code, et vous voulez savoir ce que dit le manuel sur cette commande. Déplacez simplement le curseur sur la commande **`écris`** et appuyez sur **F2**. Le manuel affichera toutes les informations sur la commande **`écris`**.
Cette fonction peut être très utile lors de l'apprentissage de TurtleScript.

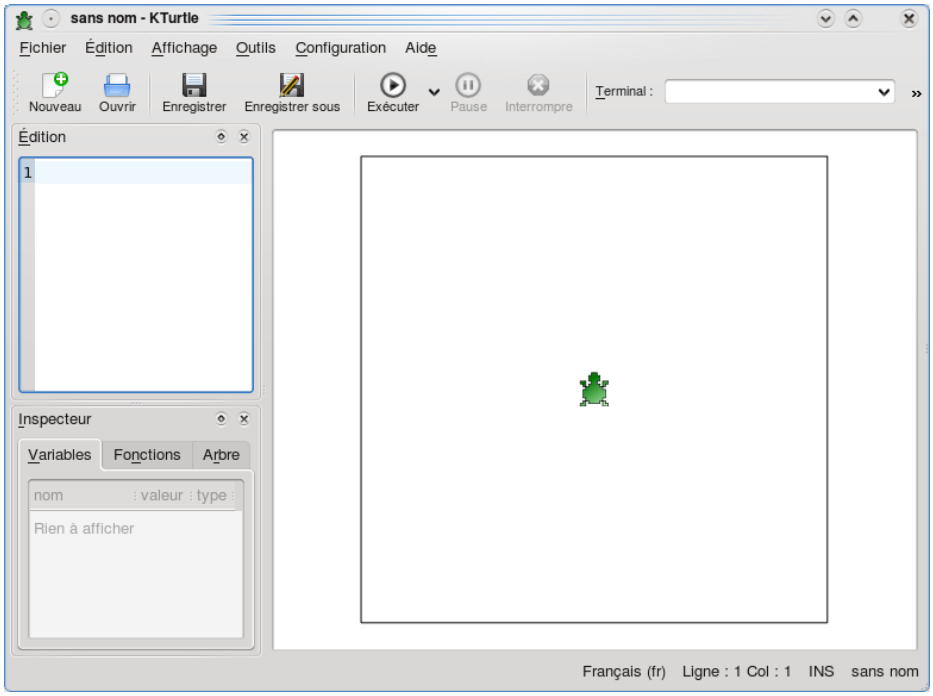
2.6 La barre d'état

Dans la barre d'état, vous trouverez des informations sur l'état de KTurtle. Sur la partie gauche, l'état de la dernière action est affiché. Sur la partie droite vous trouverez l'endroit localisé du curseur (numéros de ligne et de colonne). Au milieu de la barre d'état est indiquée la langue actuellement utilisée pour les commandes.

Chapitre 3

Démarrage rapide

Lorsque vous démarrez KTurtle vous allez voir quelque chose comme cela :



Dans ce démarrage rapide, nous supposons que la langue des commandes TurtleScript est le français. Vous pouvez modifier cette langue dans **Configuration** → **Langue du code** dans la section **Langue**. Soyez conscient que la langue que vous fixez ici pour KTurtle est celle que vous utiliserez pour saisir les commandes TurtleScript, non la langue utilisé par KDE sur l'ordinateur et utilisée pour afficher l'interface et les menus de KTurtle.

3.1 Premières étapes avec le TurtleScript : faites connaissance avec la tortue !

Vous avez remarqué que la tortue est au milieu du canevas : vous allez apprendre à la contrôler en utilisant des commandes dans l'éditeur de code.

3.1.1 La tortue bouge

Commençons par faire bouger la tortue. Notre tortue peut faire trois sortes de mouvements, (1) elle peut avancer et reculer, (2) elle peut tourner à gauche et à droite et (3) elle peut aller directement (sauter) à une position donnée sur l'écran. Vous pouvez essayer ceci par exemple :

```
avance 90
tournegauche 90
```

Saisissez ou copier / coller ce code dans l'éditeur puis exécutez-le (en utilisant **Exécuter** → **Exécuter**) pour voir le résultat.

Pendant que vous saisissez et exécutez les commandes comme celles ci-dessus dans l'éditeur, vous pouvez remarquer les choses suivantes :

1. après avoir exécuté les commandes, la tortue a avancé, a dessiné une ligne puis s'est tournée d'un quart de tour vers la gauche. Ceci est le résultat des commandes **avance** et **tournegauche** ;
2. la couleur du code change pendant que vous le saisissez : cette caractéristique est appelée *coloration syntaxique intuitive*. Différentes sortes de commandes sont colorées différemment. Ceci rend la lecture de grandes parties de code plus facile. Vous pouvez aussi détecter facilement des erreurs typographiques dans le cas où un mot n'est pas coloré ;
3. la tortue dessine une fine ligne noire ;
4. peut-être avez-vous eu un message d'erreur. Cela peut vouloir dire deux choses : vous pouvez avoir fait une erreur en copiant les commandes ou vous n'avez pas fixé la langue qui correspond aux commandes de TurtleScript (vous pouvez faire cela en choisissant **Configuration** → **Langue du code**).

Vous avez certainement compris que **avance 100** dit à la tortue d'avancer en dessinant une ligne et que **tournegauche 90** demande à la tortue de tourner vers la gauche de 90 degrés.

Veuillez suivre les liens suivants vers le manuel de référence pour une explication complète des nouvelles commandes : **avance**, **recule**, **tournegauche**, et **tournedroite**.

3.1.2 Plus d'exemples

Le premier exemple était très simple, nous allons avancer un peu plus loin !

```
initialise

taillecanevas 200, 200
couleurcanevas 0, 0, 0
couleurcrayon 255, 0, 0
largeurcrayon 5
nettoietout

va 20, 20
direction 135

avance 200
tournegauche 135
avance 100
tournegauche 135
avance 141
tournegauche 135
avance 100
tournegauche 45

va 40,100
```

Encore une fois, vous pouvez saisir ou copier / coller le code dans l'éditeur ou aussi ouvrir le fichier exemple `flèche` dans le menu **Exemples** et l'exécuter (en utilisant **Exécuter → Exécuter**) pour voir le résultat. Pour l'exemple suivant, je supposerai que vous connaissez la procédure.

Vous pouvez avoir remarqué que cet exemple utilise beaucoup plus de code. Vous y trouvez aussi de nouvelles commandes. Voici une courte explication des nouvelles commandes :

Après avoir lancé la commande **`initialise`**, tout s'affiche comme si KTurtle venait juste de démarrer.

`taillecanevas 200, 200` fixe la largeur et la hauteur du canevas à 200 pixels chacune. La largeur et la hauteur sont ici égales en taille ce qui signifie que le canevas est un maintenant un carré.

`couleurcanevas 0, 0, 0` fixe la couleur du canevas à noire. **`0, 0, 0`** est une combinaison RVB où toutes les valeurs sont mises à **`0`** ici ce qui donne un résultat de noir.

`couleurcrayon 255, 0, 0` fixe la couleur du crayon à rouge. **`255, 0, 0`** est une combinaison RVB où ici seule la valeur pour rouge est fixée au maximum à **`255`** pendant que les autres valeurs (vert et bleu) sont mises à **`0`**, ce qui résulte en un rouge brillant.

Si vous ne comprenez pas les valeurs des couleurs, lisez le glossaire sur les combinaisons RVB

`largeurcrayon 5` fixe la largeur (la taille) du crayon à **`5`** pixels. À partir de maintenant, chaque ligne que va dessiner la tortue aura une épaisseur de **`5`**, jusqu'à ce que nous changions à nouveau la valeur de **`largeurcrayon`** pour une autre valeur.

`va 20, 20` commande à la tortue d'aller à un certain endroit sur le canevas. Ceci est compté depuis le coin en haut à gauche, cet endroit est donc à 20 pixels depuis la gauche et 20 pixels depuis le haut du canevas. Veuillez noter que l'utilisation de la commande **`va`** ne dessinera pas de ligne.

`direction 135` fixe la direction de la tortue. Les commandes **`tournegauche`** et **`tournedroite`** changent l'angle de direction de la tortue en partant de sa direction actuelle. La commande **`direction`** change l'angle de direction de la tortue en partant toujours de zéro et donc ne dépend pas de la direction précédente de la tortue.

Après la commande **`direction`** suivent beaucoup de commandes **`avance`** et **`tournegauche`**. Ces commandes font le dessin que vous voyez.

Enfin, une autre commande **`va`** est utilisée pour bouger la tortue sur le côté.

Veuillez suivre les liens vers le manuel de référence. Celui-ci explique chaque commande plus en détails.

Chapitre 4

Référence de programmation TurtleScript

Ceci est le chapitre de référence TurtleScript de KTurtle. La première section de ce chapitre donne quelques aspects de la grammaire des programmes en TurtleScript. La deuxième section traite exclusivement des opérateurs mathématiques, des opérateurs booléens (vrai / faux) et des opérateurs de comparaison. La troisième partie est une énorme liste de toutes les commandes, expliquées une par une. La section quatre explique comment assigner des valeurs aux variables. Enfin, nous expliquons comment arranger l'exécution des commandes avec les instructions de contrôle d'exécution dans la section cinq, et comment créer vos propres commandes avec apprends dans la section six.

4.1 La grammaire de TurtleScript

Comme tout langage, TurtleScript dispose de différents types de mots et de symboles. En français, nous distinguons les verbes (comme « marcher » ou « chanter »), et les noms (comme « sœur » ou « maison »), ils sont utilisés pour différents objectifs. TurtleScript est un langage de programmation, il est utilisé pour indiquer à KTurtle ce qu'il faut faire.

Dans cette section, certains types de mots et symboles de TurtleScript sont expliqués brièvement. Nous expliquons les commentaires, les commandes et les trois différents types de littéraux : les nombres, les chaînes et les booléens (vrai / faux).

4.1.1 Commentaires

Un programme est constitué d'instructions qui sont exécutées lorsque le programme est lancé, et de commentaires. Les commentaires ne sont pas exécutés. KTurtle les ignore simplement lors que votre programme est exécuté. Ils sont là pour aider les autres programmeurs à mieux comprendre votre programme. Tout ce qui suit le symbole **#** est considéré comme un commentaire en TurtleScript. Par exemple, ce petit programme ne fait rien :

```
# ce petit programme ne fait rien, c'est juste un commentaire !
```

C'est un peu inutile, mais ça explique bien les choses.

Les commentaires sont très utiles lorsque le programme devient un peu plus complexe. Il peut aider à donner des conseils aux autres programmeurs. Dans le programme suivant, vous pouvez voir l'utilisation des commentaires avec la commande print.

```
# Ce programme a été écrit par Cies Breijs.
écris "ce texte sera affiché sur le canevas"
# la ligne précédente n'est pas un commentaire, mais la ligne suivante en ↩
    est un :
# écris "ce texte ne sera pas affiché : "
```

La première ligne décrit le programme. La deuxième est exécutée par KTurtle et écrit **`Ce texte sera affiché sur le canevas`** sur le canevas. La troisième ligne est un commentaire. Et la quatrième ligne est un commentaire qui contient un morceau de TurtleScript. Si le symbole **`#`** était enlevé dans la quatrième ligne, l'instruction écris serait exécutée par KTurtle. Les programmeurs disent que l'instruction écris dans la quatrième ligne est « commentée ».

Les lignes commentées sont surlignées en gris clair dans l'éditeur de code.

4.1.2 Commandes

À l'aide des commandes, vous dites à la tortue ou à KTurtle de faire quelque chose. Certaines commandent nécessitent un paramètre, d'autres donnent une sortie.

```
# avance est une commande qui nécessite un paramètre, dans ce cas le nombre ↩
    100 :
avance 100
```

La première ligne est un commentaire. La seconde ligne contient la commande **`avance`** et le nombre **`100`**. Le nombre ne fait pas partie de la commande, mais est considéré comme un « paramètre » de la commande.

Certaines commandes comme par exemple **`va`** nécessitent plus d'un paramètre en entrée. Plusieurs valeurs doivent être séparées un utilisant le caractère **`,`** (virgule).

Pour une étude détaillée de toutes les commandes gérées par KTurtle, allez ici. Les commandes internes sont surlignées en bleu foncé.

4.1.3 Nombres

Vous savez probablement certaines choses sur les nombres. La façon dont ils sont utilisés dans KTurtle n'est pas très différente de la façon dont on les utilise dans le langage parlé, ou avec les mathématiques.

Nous avons les nombres appelés nombre naturels : **`0`**, **`1`**, **`2`**, **`3`**, **`4`**, **`5`**, etc. Les nombres négatifs : **`-1`**, **`-2`**, **`-3`**, etc. Enfin les nombres décimaux, où nombres à virgule, par exemple : **`0.1`**, **`3.14`**, **`33.3333`**, **`-5.05`**, **`-1.0`**. Le caractère **`.`** (point) est utilisé comme séparateur décimal.

Les nombres peuvent être utilisés dans des opérateurs mathématiques et dans des opérateurs de comparaison. Ils peuvent également être placés dans des variables. Les nombres sont surlignés en rouge foncé.

4.1.4 Chaînes

Tout d'abord un exemple :

```
écris "Bonjour, je suis une chaîne."
```

Dans cet exemple, **`écris`** est une commande et **`Bonjour, je suis une chaîne.`** est une chaîne. Les chaînes commencent et se terminent par un symbole ". Grâce à ces symboles, KTurtle sait qu'il s'agit d'une chaîne.

Les chaînes peuvent être placées dans des variables, tout comme les nombres. Cependant, contrairement aux nombres, les chaînes ne peuvent pas être utilisées avec les opérateurs mathématiques ou avec les opérateurs de comparaison. Les chaînes sont surlignées en rouge.

4.1.5 Les valeurs booléennes (vrai / faux)

Il y a deux types de valeurs booléennes : **vrai** et **faux**. Parfois on les appelle également on et off, oui et non, un et zéro. Mais en TurtleScript, on les appelle toujours **vrai** et **faux**. Regardez cet extrait de TurtleScript :

```
$a = vrai
```

Si vous regardez dans l'inspecteur, vous verrez que la variable $a est réglée sur **vrai**, et a le type booléen.

Souvent, les valeurs booléennes sont le résultat d'un opérateur de comparaison, comme dans le morceau de TurtleScript suivant :

```
$reponse = 10 > 3
```

La variable **$reponse** est placée à **vrai** car **10** est plus grand que **3**.

Les valeurs booléenne **vrai**et **faux** sont surlignées en rouge foncé.

4.2 Les opérateurs mathématiques, booléens et de comparaison

Le titre de cette section peut sembler compliqué, mais ça ne l'est pas autant que ça en a l'air.

4.2.1 Les opérateurs mathématiques

Il y a les symboles mathématiques de base connus comme : l'addition (**+**), la soustraction (**−**), la multiplication (*****), division (**/**) et la puissance (^).

Voici un petit exemple sur les opérateurs mathématiques que vous pouvez utiliser dans TurtleScript :

```
$somme = 1 + 1
$difference = 20 - 5
$produit = 15 * 2
$rapport = 30 / 30
$puissance = 2 ^ 2
```

Les valeurs résultant de ces opérateurs mathématiques sont assignées aux différentes variables. En utilisant l'inspecteur, vous pouvez voir ces valeurs.

Si vous voulez faire simplement un calcul, vous pouvez le faire de la façon suivante :

```
écris 2010-12
```

Voici maintenant un exemple avec des parenthèses :

```
écris ( ( 20 - 5 ) * 2 / 30 ) + 1
```

Les expressions dans les parenthèses vont être calculées les premières. Dans cet exemple, 20-5 sera calculé, puis multiplié par 2, divisé par 30, puis 1 est ajouté (ce qui donne 2). Les parenthèses peuvent également être utilisées dans d'autres cas.

KTurtle dispose également de caractéristiques mathématiques plus avancées sous la forme de commandes. Jetez un œil aux commandes suivantes, mais gardez à l'esprit qu'il s'agit d'opérations avancées : arrondi, hasard, racine, pi, sin, cos, tan, arcsin, arccos, arctan.

4.2.2 Opérateurs booléens (vrai / faux)

Tandis que les opérateurs mathématiques travaillent avec des nombres, les opérateurs booléens travaillent avec des valeurs booléennes (**vrai** et **faux**). Il y a seulement trois opérateurs booléens : **et**, **ou** et **non**. Le morceau de TurtleScript suivant montre comment les utiliser :

```
$et_1_1 = vrai et vrai    # -> vrai
$et_1_0 = vrai et faux    # -> faux
$et_0_1 = faux et vrai    # -> faux
$et_0_0 = faux et faux    # -> faux

$ou_1_1 = vrai ou vrai    # -> vrai
$ou_1_0 = vrai ou faux    # -> vrai
$ou_0_1 = faux ou vrai    # -> vrai
$ou_0_0 = faux ou faux    # -> faux

$non_1 = non vrai    # -> faux
$non_0 = non vrai    # -> vrai
```

En utilisant l'inspecteur, vous pouvez voir les valeurs, bien que nous fournissons ces résultats en petits commentaires à la fin de chaque ligne. **et** ne vaut **vrai** seulement si les deux côtés sont **vrais**. **ou** vaut **vrai** si l'un des deux côtés est **vrai**. Et **non** passe **vrai**en **faux**, et **faux** en **vrai**.
Les opérateurs booléens sont surlignés en rose.

4.2.2.1 Quelques exemples plus complexes

Considérons l'exemple suivant avec **et** :

```
$a = 1
$b = 5
si (($a < 10) et ($b == 5)) et ($a < $b) {
  écris "bonjour"
}
```

Dans ce morceau de TurtleScript, le résultat de trois opérateurs de comparaison sont fusionnés en utilisant des opérateurs **et**. Ceci signifie que les trois opérateurs doivent donner vrai pour que le texte « bonjour » soit affiché.

Un exemple avec **ou** :

```
$n = 1
si ($n < 10) ou ($n == 2) {
  écris "bonjour"
}
```

Dans ce morceau de TurtleScript, la partie gauche du **ou** donne « vrai », la partie droite donne « faux ». Comme un des deux côtés de l'opérateur **ou** est « vrai », l'opérateur **ou** vaut « vrai ». Ceci signifie que « bonjour » sera affiché.

Et pour terminer un exemple avec **non**, qui change « vrai » en « faux » et « faux » en « vrai ». Regardez :

```
$n = 1
si non ($n == 3) {
  écris "bonjour"
} sinon {
  écris "pas bonjour ;-)"
}
```

4.2.3 Opérateurs de comparaison

Considérons cette comparaison simple :

```
$reponse = 10 >
```

Ici, **10** est comparé à **trois** avec l'opérateur « plus grand que ». le résultat de cette comparaison, la valeur booléenne **vrai** est stockée dans la variable **$reponse**.

Tous les nombres et variables (qui contiennent des nombres) peuvent être comparés entre eux avec les opérateurs de comparaison.

Voici tous les opérateurs de comparaison possibles :

$A == $B	égal	la réponse est "vrai" si **$A** est égal à **$B**
$A != $B	non égal	la réponse est "vrai" si **$A** n'est pas égal à **$B**
$A > **$B**	plus grand que	la réponse est "vrai" si **$A** est plus grand que **$B**
$A < **$B**	plus petit que	la réponse est "vrai" si **$A** est plus petit que **$B**
$A >=$ B	supérieur ou égal	la réponse est "vrai" si **$A** est plus grand ou égal à **$B**
$A <= $B	inférieur ou égal	la réponse est "vrai" si **$A** est plus petit ou égal à **$B**

TABLE 4.1: Type de question

Notes que $A et $B doivent être des nombres ou des variables qui contiennent ces nombres.

4.3 Commandes

En utilisant les commandes, vous dites à la tortue ou à KTurtle de faire quelque chose. Certaines commandes ont besoin d'une entrée, d'autres donnent une sortie. Dans cette section, nous expliquons toutes les commandes internes de KTurtle. Vous pouvez également créer vos propres commandes avec apprends. Les commandes internes discutées ici sont surlignées en bleu foncé.

4.3.1 Déplacer la tortue

Il y a plusieurs commandes qui déplacent la tortue sur l'écran.

avance (av)

```
avance X
```

avance fait avancer la tortue sur l'écran de X pixels. Lorsque le crayon est baissé, la tortue laisse une trace (dessine une ligne). **avance** peut se contracter en **av**

recule (re)

```
recule X
```

recule fait reculer la tortue sur l'écran de X pixels. Lorsque le crayon est baissé, la tortue laisse une trace (dessine une ligne). **recule** peut se contracter en **re**.

tournegauche (tg)

```
tournegauche X
```

tournegauche dit à la tortue de se tourner vers la gauche de X degrés. **tournegauche** peut se contracter en **tg**.

tournedroite (td)

```
tournedroite X
```

tournedroite dit à la tortue de se tourner vers la droite de X degrés. **tournedroite** peut se contracter en **td**.

direction (dir)

```
direction X
```

direction fixe la direction de la tortue de X degrés en comptant de zéro, et donc ne dépend pas de la position et de la direction précédentes de la tortue. **direction** peut se contracter en **dir**.

obtenirdirection

```
obtenirdirection
```

obtenirdirection renvoie la direction de la tortue en degrés en partant de zéro, où zéro correspond à la tortue pointant vers le haut.

centre

```
centre
```

centre déplace la tortue au centre du canevas.

va

```
va X,Y
```

va commande à la tortue d'aller à un certain endroit sur le canevas. Cet endroit est à X pixels depuis la gauche du canevas et à Y pixels depuis le haut du canevas.

vax

```
vax X
```

vax : en utilisant cette commande, la tortue va se déplacer de X pixels depuis la gauche du canevas tandis qu'elle restera à la même hauteur. **vax** peut être abrégé en **vx**.

vay

```
vay Y
```

vay : en utilisant cette commande, la tortue va se déplacer de Y pixels depuis le haut du canevas tandis qu'elle restera à la même distance de la bordure gauche du canevas. **vay** peut être abrégé en **vy**.

NOTE

En utilisant les commandes **vas**, **vax**, **vay** et **centre** la tortue ne dessinera pas une ligne, peut importe si le crayon est levé ou baissé.

4.3.2 Où est la tortue ?

Il y a deux commandes qui donnent la position de la tortue sur l'écran.

positionx

positionx retourne le nombre de pixels séparant la gauche du canevas de la position courante de la tortue.

positiony

positiony retourne le nombre de pixels séparant le haut du canevas de la position courante de la tortue.

4.3.3 La tortue a un crayon

La tortue a un crayon qui trace une ligne lorsqu'elle se déplace. Il y a peu de commandes pour contrôler le crayon. Nous expliquons ces commandes dans ce paragraphe.

lèvecrayon (lc)

```
lèvecrayon
```

lèvecrayon relève le crayon du canevas. Lorsque le crayon est "levé", aucune ligne n'est tracée lorsque la tortue se déplace. Voir aussi **baissecrayon**. **lèvecrayon** peut se contracter en **lc**.

baissecrayon (bc)

```
baissecrayon
```

baissecrayon abaisse le crayon sur le canevas. Lorsque le crayon est "baissé" sur le canevas, une ligne est tracée lorsque la tortue se déplace. Voir aussi **lèvecrayon**. **baissecrayon** peut se contracter en **bc**.

largeurcrayon (lac)

```
largeurcrayon X
```

largeurcrayon fixe l'épaisseur (la largeur du trait) du crayon à X pixels. **largeurcrayon** peut se contracter en **lac**.

couleurcrayon (cc)

```
couleurcrayon R, G, B
```

couleurcrayon fixe la couleur du crayon. **couleurcrayon** demande une combinaison RVB comme entrée. **couleurcrayon** peut se contracter en **cc**.

4.3.4 Commandes pour contrôler le canevas

Voici les différentes commandes pour contrôler le canevas.

taillecanevas (tc)

```
taillecanevas X, Y
```

Avec la commande **taillecanevas**, vous pouvez fixer la taille du canevas. Elle reçoit X et Y comme entrée, où X est la nouvelle largeur du canevas en pixels, et Y est la nouvelle hauteur du canevas en pixels. **taillecanevas** peut se contracter en **tc**.

couleurcanevas (cca)

```
couleurcanevas R, G, B
```

couleurcanevas fixe la couleur du canevas. **couleurcanevas** reçoit une combinaison RVB comme entrée. **couleurcanevas** peut se contracter en **cca**.

4.3.5 Commandes pour nettoyer

Il existe deux commandes pour nettoyer le canevas après avoir mis le désordre.

nettoietout (ntt)

```
nettoietout
```

Avec **nettoietout**, vous pouvez nettoyer tous les dessins sur le canevas. Toutes les autres choses restent : la position et l'angle de la tortue, la couleur du canevas, la visibilité de la tortue et la taille du canevas.

initialise

```
initialise
```

initialise nettoie beaucoup plus profondément que la commande **nettoietout**. Après la commande **initialise**, tout redevient comme lorsque vous avez lancé KTurtle. La tortue se place au milieu de l'écran, la couleur du canevas est blanche, la tortue trace une ligne noire sur le canevas et taillecanevas est défini à 400 x 400 pixels.

4.3.6 La tortue est un lutin (*sprite*)

La plupart des personnes ne savent pas ce qu'un lutin (*sprite*) est, voici donc une courte explication : les lutins sont de petites images qui peuvent se déplacer sur l'écran. (pour plus d'information, voir le glossaire sur lutins). Donc la tortue est un lutin !

Ce qui suit est un aperçu des commandes relatives aux lutins.

[La version actuelle de KTurtle ne gère pas encore l'utilisation des lutins autres que la tortue. Dans des versions futures vous pourrez remplacer la tortue par quelque chose de votre propre conception !]

montre (mo)

```
montre
```

montre rend de nouveau la tortue visible après avoir été cachée.**montre** peut être abrégé par **mo**.

cache (ca)

```
cache
```

cache cache la tortue. Ceci peut être utilisé si la tortue ne s'adapte pas à votre dessin. **cache** peut se contracter en **ca**.

4.3.7 La tortue peut-elle écrire du texte ?

La réponse est : "oui". La tortue peut écrire, elle écrit tout ce que vous lui commander d'écrire.

écris

```
écris X
```

La commande **écris** est utilisée pour commander à la tortue d'écrire quelque chose sur le canevas. **écris** reçoit des nombres et des chaînes de caractères comme entrée. Vous pouvez utiliser **écris** pour écrire plusieurs nombres et chaînes en utilisant le symbole "+". Voici un petit exemple :

```
$année = 2003
$auteur = "Cies"
écris $auteur + " a commencé de projet KTurtle en" + $année + "et prend
 toujours du plaisir à travailler dessus ! "
```

taillepolice

```
taillepolice X
```

taillepolice fixe la taille de la police qui est utilisée par la commande **écris**. **taille police** reçoit une entrée qui doit être un nombre. La taille est fixée en pixels.

4.3.8 Commandes mathématiques

Les commandes suivantes sont des commandes mathématiques de KTurtle plus avancées.

arrondi

```
arrondi(x)
```

arrondi le nombre donné à l'entier le plus proche.

```
écris arrondi(10.8)
avance 20
écris arrondi(10.3)
```

Avec ce code, la tortue écrira les nombre 11 et 10.

hasard (hsd)

```
hasard X, Y
```

hasard est une commande qui demande une entrée et qui vous donne une sortie. Comme entrée sont requis deux nombres, le premier (X) donne la sortie minimale, et le second (Y) fixe le maximum. La sortie est un nombre choisi au hasard qui est égal ou plus grand que le minimum et égal ou plus petit que le maximum. Voici un petit exemple :

```
 répète 500 [
  $x = hasard 1, 20
  avance $x
  tournedroite 10 - $x
]
```

En utilisant la commande **hasard**, vous pouvez ajouter un peu de chaos dans votre programme.

mod

```
mod X,Y
```

La commande **mod** renvoie le reste de la division entière du premier nombre par le second.

racine

```
racine X
```

La commande **racine** est utilisée pour trouver la racine carrée d'un nombre X.

pi

```
pi
```

Cette commande renvoie la constante PI, **3,14159**.

sin, cos, tan

```
sin X
cos X
tan X
```

Ces trois commandes représentent les fameuses fonctions trigonométriques **sin**, **cos** et **tan**. L'argument en entrée pour ces trois commandes, X, est un nombre.

arcsin, arccos, arctan

```
arcsin X
arccos X
arctan X
```

Ces commandes sont les fonctions inverse des fonctions sin, cos et tan. L'argument en entrée pour ces trois commandes, X, est un nombre.

4.3.9 Entrée et retour avec les boîtes de dialogue

Une boîte de dialogue est une petite fenêtre contextuelle qui fournit du retour ou demande des choses en entrée. KTurtle possède deux commandes pour les boîtes de dialogue : **message** et **demande**

message

```
message X
```

La commande **message** prend une chaîne en entrée. Elle affiche une boîte de dialogue contenant le texte de la chaîne.

```
message "Cies a commencé à travailler sur KTurtle en 2003 et s'amuse ↩
    toujours à travailler dessus ! "
```

demande

```
demande X
```

demande prend une chaîne en entrée. Elle affiche cette chaîne une boîte de dialogue (similaire à message), ainsi qu'un champ d'entrée. Après que l'utilisateur a saisi un nombre ou une chaîne, le résultat peut être stocké dans une variable ou passé comme argument à une commande. Par exemple :

```
 $entree = demande "quelle est votre année de naissance ? "
$sortie = 2010 - $in
écris "En 2010, vous avez eu " + $sortie + " ans à un moment donné."
```

Si l'utilisateur annule la boîte de dialogue, ou ne met rien du tous, la variable est vide.

4.4 Affectations des variables

Nous devons d'abord regarder les variables, ensuite, nous verrons comment affecter des valeurs à ces variables.

Les variables sont des mots qui commencent par un "$", dans l'éditeur, elles sont surlignées en violet.

Les variables contiennent soit des nombres, des chaînes ou des valeurs booléennes (vrai / faux). En utilisant une affectation, **=**, une variable reçoit son contenu. Elle le conservera jusqu'à ce que le programme se termine ou qu'on assigne quelque chose d'autre à la variable.

Vous pouvez utiliser les variables, une fois affectées, tout comme s'il s'agissait de leur contenu. Par exemple, dans le morceau de TurtleScript :

```
 $x = 10
$x = $x / 3
écris $x
```

Tout d'abord la variable **$x** se voit affecter la valeur **10**. Ensuite, **$x** se voit de nouveau affecter sa propre valeur divisée par **3** — ceci signifie en fait que **$x** est affectée du résultat de **10 + 3**. Enfin, **$x** est affichée. Dans les lignes deux et trois, vous voyez que **$x** est utilisé comme s'il s'agissait de son contenu.

Les variables doivent être affectées avant de pouvoir être utilisées. Par exemple :

```
écris $n
```

donnera un message d'erreur.

Regardez le morceau de TurtleScript suivant :

```
$a = 2004
$b = 25

# la commande suivante affiche « 2029 »
écris $a + $b
recule 30
# la commande suivante écrit "2004 plus 25 égal 2009"
écris $a + " plus " + $b + " égal " + ($a + $b)
```

Dans les deux premières lignes, les variables **$a** et **$b** sont initialisées à 2004 et 25. Ensuite viennent les deux commandes **écris** et un **recule 30** entre. Les commentaires avant les commandes **écris** expliquent ce que ces commandes font. La commande **recule 30** est ici pour s'assurer que chaque sortie se trouve sur une nouvelle ligne. Comme vous le voyez, les variables peuvent être utilisée comme s'il s'agissait de leur contenu, vous pouvez les utiliser avec n'importe quel type d'opérateurs ou les passer en entrée lorsque vous appelez des commandes.

Un autre exemple :

```
$nom = demande "Quel est votre nom ? "
écris "Salut " + $nom + " ! Bonne chance dans l'apprentissage de la ↩
    programmation..."
```

Plutôt direct. Encore une fois, vous voyez que la variable **$nom** est traitée comme une chaîne.

Lorsque vous utilisez des variables, l'inspecteur est très utile. Il vous montre le contenu de toutes les variables actuellement en cours d'utilisation.

4.5 Contrôler l'exécution

Les contrôleurs d'exécution vous permettent, — comme leur nom l'indique — de contrôler l'exécution.

Les commandes de contrôle d'exécution sont surlignées en vert foncé en utilisant une police en gras. Les accolades sont principalement utilisées avec les contrôleurs d'exécution et sont surlignées en noir.

4.5.1 Faite attendre la tortue

Si vous avez exécuté quelques programmes dans KTurtle vous pouvez avoir remarqué que la tortue se déplace très rapidement pour dessiner. La commande suivante ralentit la tortue.

attends

```
attends X
```

attends fait attendre la tortue pendant X secondes.

```
répète 36 {
  avance 5
  tournedroite 10
  attends 0.5
}
```

Ce code dessine un cercle, mais la tortue va attendre une demi-seconde après chaque étape. Cela donne l'impression d'une tortue qui va moins vite.

4.5.2 Exécute « si »

si

```
si booléen { ... }
```

Le code qui est placé entre les accolades sera exécuté seulement **si** la valeur booléenne vaut "vrai".

```
$x = 6
si $x > 5 {
  écris "x est plus grand que cinq ! "
}
```

À la première ligne, **$x** est fixé à 6. À la seconde ligne, un opérateur de comparaison est utilisé pour évaluer **$x** > **5**. Comme le résultat est "vrai", 6 est plus grand que 5, le contrôleur d'exécution **si** va permettre au code entre accolades d'être exécuté.

4.5.3 Si non, en d'autres termes : « sinon »

sinon

```
si booléen { ... } sinon { ... }
```

sinon peut être utilisé en plus du contrôleur d'exécution **si**. Le code entre accolades après **sinon** n'est exécuté que si le booléen vaut "faux".

```
initialise
$x = 4
si $x > 5 {
  écris "x est plus grand que cinq ! "
} sinon {
  écris "x est plus petit que six ! "
}
```

L'opérateur de comparaison évalue l'expression **$x** > **5**. Comme 4 n'est pas plus grand que 5, l'expression vaut "faux". Cela signifie que le code entre accolades après **sinon** est exécuté.

4.5.4 La boucle « tantque »

tantque

```
tantque booléen { ... }
```

Le contrôleur d'exécution **tantque** ressemble beaucoup à **si**. La différence est que **tantque** répète le code entre accolades jusqu'à ce que le booléen vaut "faux"

```
$x = 1
tantque $x < 5 {
  avance 10
  attends 1
  $x = $x + 1
}
```

À la première ligne, **$x** est fixé à 1. À la seconde ligne, **$x** < **5** est évalué. Comme la réponse à cette question est "vrai", le contrôleur d'exécution **tantque** commence à exécuter le code entre accolades jusqu'à ce que **$x** < **5** soit "faux". Dans ce cas, le code entre accolades sera exécuté quatre fois, à chaque fois que la cinquième ligne est exécutée, **$x** augmente de 1.

4.5.5 La boucle « répète »

répète

```
répète nombre { ... }
```

Le contrôleur d'exécution **répète** est similaire à **tantque**. La différence est que **répète** laisse en boucle le code entre accolades autant de fois que le nombre donné.

4.5.6 La boucle « pour », une boucle de comptage

pour

```
pour variable = nombre à nombre { ... }
```

La boucle **pour** est une "boucle de comptage", c'est-à-dire qu'elle compte à votre place. Le premier nombre règle la variable pour la première boucle. À chaque boucle, le nombre est incrémenté jusqu'à ce que la seconde valeur soit atteinte.

```
pour $x = 1 à 10 {
  écris $x * 7
  avance 15
}
```

Chaque fois que le code entre accolades est exécuté, la valeur de **$x** augmente de 1, jusqu'à ce que **$x** atteigne la valeur 10. Le code entre les accolades écrit **$x** multiplié par 7. Après que ce programme a fini son exécution, vous allez voir la table de multiplication de 7 imprimée sur le canevas.
La taille du pas par défaut de la boucle est 1, vous pouvez utiliser une autre valeur avec :

```
pour variable = nombre à nombre pas nombre { ... }
```

4.5.7 Quitter la boucle

coupure

```
coupure
```

Termine immédiatement la boucle courante et transfert le contrôle à la déclaration qui suit la boucle.

4.5.8 Arrête l'exécution du programme

sortie

```
sortie
```

Termine l'exécution de votre programme.

4.5.9 Vérifier une assertion lors de l'exécution

assertion

```
assertion booléen
```

Peut être utilisé pour déterminer la conformité d'un programme ou d'une entrée.

```
$in = demande "quel est votre année de naissance ? "
# l'année doit être positive
assertion $in
> 0
```

4.6 Créez vos propres commandes avec "apprends"

apprends est une commande très spéciale, car elle est utilisée pour créer vos propres commandes. La commande que vous créez peut recevoir des entrées et retourner des sorties. Regardons maintenant comment une nouvelle commande est créée :

```
apprends cercle $x {
  répète 36 {
    avance $x
    tournegauche 10
  }
}
```

La nouvelle commande est appelée **cercle**. **cercle** reçoit un argument en entrée, qui fixe la taille du cercle. **cercle** ne retourne aucune sortie. La commande **cercle** peut maintenant être utilisée comme une commande normale dans la suite du programme. Voyez cet exemple :

```
apprends cercle $X {
  répète 36 {
    avance $X
    tournegauche 10
  }
}

va 30,30
cercle 20

va 40,40
cercle 50
```

Dans l'exemple suivant, une commande avec une valeur de retour est créée.

```
apprends factorielle $x {
  $r = 1
  pour $i = 1 à $x {
    $r = $r * $i
  }
  retourne $r
}

écris factorielle 5
```

Dans cet exemple, une nouvelle commande appelée **factorielle** est créée. Si l'entrée de la commande est **5**, alors la sortie sera **5*4*3*2*1**. En utilisant **retourne**, la valeur de sortie est spécifiée et le résultat est renvoyé.

Les commandes peuvent avoir plus d'une entrée. Dans l'exemple suivant, une commande qui dessine un rectangle est créé.

```
apprends boîte $x, $y {
  avance $y
  tournedroite 90
  avance $x
  tournedroite 90
  avance $y
  tournedroite 90
  avance $x
  tournedroite 90
}
```

Maintenant vous pouvez lancer **boîte 50, 100** et la tortue dessinera un rectangle sur le canevas.

Chapitre 5

Glossaire

Dans ce chapitre, vous trouverez une explication de la plupart des mots “difficiles” qui sont utilisés dans ce guide de l’utilisateur.

degrés

Les degrés sont les unités de mesure des angles ou des tournants. Un tour complet est 360 degrés, un demi tour est 180 degrés et un quart de tour est 90 degrés. Les commandes **tournegauche**, **tournedroite** et **direction** demandent une entrée en degrés.

entrée et sortie des commandes

Certaines commandes reçoivent une entrée, d’autres commandes donnent une sortie, certaines commandes reçoivent une entrée *et* donnent une sortie et d’autres commandes ne reçoivent pas d’entrée ni ne donnent de sortie.

Voici des exemples de commandes qui ne reçoivent que des entrées :

```
avance 50
couleurcrayon 255,0,0
écris "Bonjour ! "
```

La commande **avance** reçoit **50** comme entrée. **avance** a besoin de cette entrée pour savoir de combien de pixels la tortue doit avancer. **couleurcrayon** prend une couleur comme entrée et **écris** prend une chaîne de caractères (un morceau de texte) comme entrée. Remarquez que l’entrée peut aussi être un conteneur. L’exemple suivant illustre cela :

```
$x = 50
écris $x
$str = "Bonjour ! "
écris $str
```

Maintenant quelques exemples de commandes qui donnent une sortie :

```
$x = demande "Veuillez saisir quelque chose et faire Entrée... Merci ! ↩
    "
$r = hasard 1, 100
```

La commande **demande** reçoit une chaîne de caractères comme entrée et en sortie le nombre ou la chaîne qui est entrée. Comme vous pouvez le voir, la sortie de **demande** est stockée dans le conteneur **x**. La commande **hasard** donne aussi une sortie. Dans ce cas, la sortie est un nombre entre 1 et 100. La sortie de hasard est ensuite stockée dans un conteneur appelé **r**. Veuillez noter que les conteneurs **x** et **r** ne sont pas utilisés dans l’exemple de code ci-dessus.

Il existe aussi des commandes qui ne reçoivent pas d’entrée et qui ne donnent pas de sortie. Voici quelques exemples :

```
nettoietout
lèvecrayon
```

coloration syntaxique

Ceci est une caractéristique de KTurtle qui rend la programmation vraiment plus facile. Avec la coloration syntaxique, le code que vous écrivez prend une couleur qui indique de quel type de code il s'agit. Dans la liste suivante, vous trouverez les différents types de code ainsi que la couleur qu'ils auront dans l'éditeur.

commandes habituelles	bleu foncé	Les commandes habituelles sont décrites ici.
Commandes contrôlant l'exécution	noir (gras)	Ces commandes spéciales contrôlent l'exécution. Pour en savoir plus, veuillez voir ici.
commentaires	gris	Les lignes qui sont commentées débutent par le caractère de commentaire (#). Ces lignes sont ignorées lorsque le code est exécuté. Les commentaires permettent au programmeur d'expliquer son code ou ils peuvent aussi être utilisés pour empêcher temporairement une partie du code d'être exécutée.
accolades {, }	vert foncé (gras)	Les accolades sont utilisées pour grouper des portions de code. Les crochets sont souvent utilisés en conjonction avec les contrôleurs d'exécution.
la commande apprends	vert clair (gras)	La commande apprends est utilisée pour créer de nouvelles commandes.
chaînes	rouge	Il n'y a pas non plus grand-chose à dire sur les chaînes de caractère (texte) sauf qu'elles commencent et finissent toujours avec des guillemets doubles (″).
nombres	rouge foncé	Les nombres, il n'y a rien à dire sur eux.
valeurs booléennes	rouge foncé	Il y a exactement deux valeurs booléennes : vrai et faux.

variables	violet	démarre par un « $ » et peut contenir des nombres, des chaînes ou des valeurs booléennes.
opérateurs mathématiques	gris	Voici les opérateurs mathématiques : **+**, **−**, *****, **/** et **^**.
opérateurs de comparaison	bleu clair (gras)	Voici les opérateurs de comparaison : **==**, **!=**, **<**, **>**, **<=** et **>=**.
opérateurs booléens	rose (gras)	Voici les opérateurs booléens : **et**, **ou** et **non**.
texte normal	noir	

TABLE 5.1: Les différents types de code et leur coloration syntaxique

pixels

Vous verrez que l'écran de votre moniteur utilise des pixels. Toutes les images sur l'écran sont construites avec ces pixels. Un pixel est la plus petite chose qui puisse être dessinée sur l'écran.
Beaucoup de commandes demandent un nombre de pixels comme entrée. Ces commandes sont : **avance**, **recule**, **va**, **vax**, **vay**, **taillecanevas** et **largeurcrayon**.
Dans les versions précédentes de KTurtle, le canevas était essentiellement une image matricielle. Dans les dernières versions, le canevas est un dessin vectoriel. Ceci signifie que l'on peut zoomer le canevas, et ainsi un pixel ne se traduit pas nécessairement par un point sur l'écran.

Les combinaisons RVB (codes de couleur)

Les combinaisons RVB sont utilisées pour décrire les couleurs. Le "R" veut dire "rouge", le "V" veut dire "vert" et le "B" signifie "bleu". Un exemple d'une combinaison RVB est **255, 0, 0** : la première valeur ("rouge") est 255 et les autres sont fixées à zéro, cela représente donc un rouge vif. Chaque valeur dans une combinaison RVB doit être comprise entre 0 et 255. Voici une courte liste des combinaisons RVB les plus souvent utilisées :

0, 0, 0	noir
255, 255, 255	blanc
255, 0, 0	rouge
150, 0, 0	rouge foncé
0, 255, 0	vert
0, 0, 255	bleu
0, 255, 255	bleu clair
255, 0, 255	rose
255, 255, 0	jaune

TABLE 5.2: Les combinaisons RVB les plus souvent utilisées

Deux commandes ont besoin d'une combinaison RVB comme entrée : ces commandes sont **couleurcanevas** et **couleurcrayon**.

sprite

Un lutin est une petite image qui peut bouger sur l'écran. Notre chère tortue, par exemple, est un lutin.

NOTE
Dans cette version de KTurtle le lutin ne peut changer d'une tortue en quelque chose d'autre. Dans les versions futures de KTurtle, ce sera possible.

Chapitre 6

Guide pour la traduction des fichiers de KTurtle

Comme vous le savez déjà probablement, le langage de programmation de KTurtle, TurtleScript, peut être traduit. Ceci permet de faire tomber une barrière, en particulier pour les étudiants les plus jeunes, dans leur effort pour comprendre les bases de la programmation.

Lorsque vous traduisez KTurtle dans une nouvelle langue, vous trouverez, en plus des chaînes de l'interface graphique, des commandes de programmation, des exemples et des messages d'erreurs inclus dans les fichiers standards au format `.pot`, de la même façon que ceux utilisés pour la traduction de KDE. Tout est traduit en utilisant la méthode habituelle de traduction de KDE. Il vous est ainsi fortement recommandé de vous documenter un peu sur la façon de traduire (comme vous le liriez probablement dans les commentaires de traducteurs).

Veuillez regarder la page http://edu.kde.org/kturtle/translator.php pour plus d'informations à propos du processus de traduction. Merci beaucoup pour votre travail! KTurtle dépend beaucoup de ses traductions.

Chapitre 7

Remerciements et licence

KTurtle

Annexe A

Installation

A.1 Comment se procurer KTurtle

KTurtle fait partie du projet KDE http://www.kde.org/ .

KTurtle se trouve dans le paquet kdeedu à l'adresse ftp://ftp.kde.org/pub/kde/ , le site FTP principal du projet KDE.

A.2 Compilation et installation

Pour des informations détaillées sur comment compiler et installer les applications KDE, consultez la page Construire et Lancer KDE à partir des sources

Étant donné que KDE utilise **cmake**, vous ne devriez pas rencontrer de problèmes pour le compiler. Si c'est le cas, veuillez les signaler aux listes de discussions de KDE.

Annexe B

Index