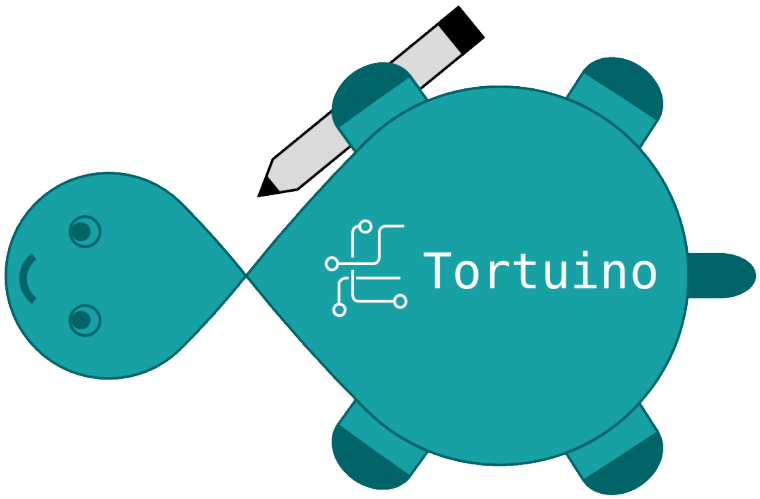


Documentation Tortuino

Généré par Doxygen 1.8.13

Table des matières

Chapitre 1

Index des fichiers

1.1 Liste des fichiers

Liste de tous les fichiers avec une brève description :

Chapitre 2

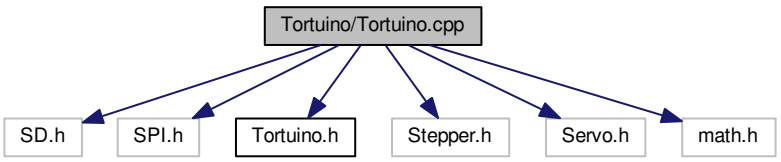
Documentation des fichiers

2.1 Référence du fichier Tortuino/Tortuino.cpp

Ce fichier décrit les instructions de base pour contrôler le robot.

```
#include <SD.h>
#include <SPI.h>
#include <Tortuino.h>
#include <Stepper.h>
#include <Servo.h>
#include <math.h>
```

Graphe des dépendances par inclusion de Tortuino.cpp :



Fonctions

- int distanceToStep (float distance)
  Réalise la conversion d'une distance que le robot peut parcourir en un certain nombre de pas que chacun des deux moteurs pas à pas doit effectuer pour que le robot puisse avancer de la distance donnée.
- void initialiser ()
  Initialise la configuration du robot pour qu'il puisse correctement communiquer avec ses différents composants qui le constituent : le servomoteur, les moteurs pas à pas et le bouton de démarrage différé.
- void initialiser (char couleur)
  Cette version de l'opération d'initialisation met en place une méthode pour calibrer chaque robot à souhait pour que les erreurs systématiques au moment de la rotation puissent être compensées.
- void initialiser (float braquage)
  Cette version de l'opération d'initialisation est utile à la calibration d'un robot car affecte directement au rayon de braquage la moitié de la valeur fournie en entrée.
- void attendreBouton ()
  Réalise l'attente nécessaire à la fonctionnalité du démarrage différé.

- void stopper ()
  Permet de mettre à l'arrêt l'exécution en cours que réalise l'Arduino.
- void vitesse (int v)
  Règle la vitesse de rotation des moteurs pas à pas.
- void avancer (float distance)
  Fait avancer le robot Tortuino d'une distance donnée.
- void reculer (float distance)
  Fait reculer le robot Tortuino d'une distance donnée.
- void tournerGauche (float angle)
  Fait tourner sur place le robot Tortuino d'un angle fourni vers sa gauche.
- void tournerDroite (float angle)
  Fait tourner sur place le robot Tortuino d'un angle fourni vers sa droite.
- void monterFeutre ()
  Place le feutre en position haute de telle manière qu'il ne touche pas la feuille en-dessous du robot, en supposant que le collier le tenant et permettant ce déplacement soit correctement ajusté.
- void descendreFeutre ()
  Place le feutre en position basse de telle manière qu'il touche la feuille en-dessous du robot, en supposant que le collier le tenant et permettant ce déplacement soit correctement ajusté.

Variables

- const int stepsPerRevolution = 64 ∗ 64 / 2
  Le nombre de pas par tour que réalise un moteur pas à pas ; c'est une donnée constructeur.
- float PERIMETER = M_PI ∗ 9.2
  Le périmètre des roues du robot tel que mesuré avec le pneu.
- float BRAQUAGE = 11.3 / 2
  Le rayon de braquage du robot.
- const int FEUTRE_HAUT = 50
  L'angle de la position haute du servomoteur.
- const int FEUTRE_BAS = 10
  L'angle de la position basse du servomoteur.
- const int portBouton = 7
  Le numéro de la broche qui sert de port pour le bouton permettant le démarrage différé : 7.
- const int portServo = 9
  Le numéro de la broche pour le port du servomoteur : 9.
- const int delaiEntreBouton = 10
  Le délai en ms entre chaque test du bouton.
- const int delaiApresBouton = 500
  Le délai en ms effectué après que le bouton ait été pressé.
- const int delaiMonterDescendre = 200
  Le délai en ms d'attente après l'envoi d'une commande au feutre.
- Stepper stepperLeft = Stepper(stepsPerRevolution, 10, 12, 11, 13)
  L'objet qui sert à contrôler le moteur pas à pas de gauche et qui est relié aux ports 10 à 13.
- Stepper stepperRight = Stepper(stepsPerRevolution, 2, 4, 3, 5)
  L'objet qui sert à contrôler le moteur pas à pas de droite et qui est relié aux ports 2 à 5.
- Servo servo
  L'objet qui sert à contrôler le servomoteur soulevant et abaissant le feutre du robot.

2.1.1 Description détaillée

Ce fichier décrit les instructions de base pour contrôler le robot.

Auteur

Alexandre Comte
Paul Mabileau paulmabileau@hotmail.fr
Florian Bescher

Version

1.3

Le fichier Tortuino.cpp rassemble les fonctionnalités essentielles au bon fonctionnement de la communication avec un robot Tortuino. Ces fonctionnalités sont principalement réalisées et mises à disposition au travers de fonctions qui les implémentent. Il y a aussi des paramètres en début de fichiers qui peuvent être modifiés à souhait pour adapter au mieux les programmes au robot qui sera manipulé au final : ils se révèlent particulièrement utiles pour le calibrer. Voir la section **Variables** pour plus de détails.

Pour comprendre comment peut s'utiliser cette bibliothèque de manière plus pratique, le fichier de test Test↩ Tortuino.ino résume assez bien l'ensemble des choses réalisables grâce à elle. Pour expliquer avec un autre exemple, les instructions suivantes permettent de faire dessiner au robot un carré de côté 10cm :

```
void setup() {                      // setup() n'est exécutée qu'une seule fois.
    initialiser();                  // Règle l'Arduino sur les bons ports de communication.

    for (int i = 0; i < 4; i++) {   // Pour chacun des côtés,
        avancer(10);                // on avance de 10cm,
        tournerGauche(90);          // et on tourne vers la gauche de 90°.
    }
}

void loop() {                       // loop() est exécutée en boucle, après setup().
                                    // Aucune instruction de plus à ajouter ici.
}
```

Notez bien que dans ce code source, les fonctions ainsi déclarées `setup()` et `loop()` sont spécifiques à l'Arduino, le tout se doit d'être utilisé avec l'éditeur Arduino IDE fourni par le constructeur. Consultez la `documentation Arduino` pour plus de détails à ce sujet et en particulier la référence précise de l'utilisation des fonctions `setup()` et `loop()`.

La présente bibliothèque fourni aussi une fonction stopper() qui une fois appelée bloquera l'exécution de tout programme. Cela permet alors de réaliser le programme précédent de manière équivalente, mais en séparant cette fois-ci l'initialisation de l'exécution :

```
void setup() {                      // setup() n'est exécutée qu'une seule fois.
    initialiser();                  // Règle l'Arduino sur les bons ports de communication.
}

void loop() {                       // loop() est exécutée en boucle, après setup().
    for (int i = 0; i < 4; i++) {   // Pour chacun des côtés,
        avancer(10);                // on avance de 10cm,
        tournerGauche(90);          // et on tourne vers la gauche de 90°.
    }

    stopper();                      // Enfin, on empêche l'Arduino de boucler à l'infini.
}
```

Les deux programmes n'ont, au fond, aucune différence : le robot tracera le même carré. Ceci peut être intéressant pour améliorer la compréhension de la syntaxe Arduino, car `setup()` ne gardera ainsi que ce qui est effectivement lié à l'initialisation du robot, tandis que `loop()` se chargera du principal du programme. Cependant, l'instruction stopper() présente ci-dessus à la fin de `loop()` est très importante car sinon, le robot exécutera en boucle les instructions de `loop()` et le dessin qu'elles décrivent sera tracé indéfiniment. Un choix pédagogique est donc à faire ici.

De plus, la bibliothèque fourni une fonctionnalité supplémentaire : un moyen pour le robot de contrôler son servomoteur pour faire monter ou descendre son feutre. On peut donc par exemple reprendre le programme précédant et le modifier un peu pour tracer un carré en pointillés de la sorte :

```
void setup() {                      // setup() n'est exécutée qu'une seule fois.
    initialiser();                  // Règle l'Arduino sur les bons ports de communication.
}

void loop() {                       // loop() est exécutée en boucle, après setup().
    for (int i = 0; i < 4; i++) {   // Pour chacun des côtés,
        descendreFeutre();          // on met le feutre en position basse,
        avancer(3.33);              // on avance du premier tiers du côté en cours : trait tracé,
        monterFeutre();             // on monte le feutre,
        avancer(3.33);              // on avance du deuxième tiers du côté en cours : trait non
       tracé,
        descendreFeutre();          // on fait descendre le feutre,
        avancer(3.33);              // on avance du dernier tiers : trait tracé,
        tournerGauche(90);          // et on tourne à gauche pour le côté suivant.
    }

    stopper();                      // Enfin, on empêche l'Arduino de boucler à l'infini.
}
```

Remarquez bien qu'ici les appels à la fonction avancer() sont réalisés avec un argument effectivement de type flottant (`float`), alors que précédemment, seulement fournir un entier (`int`) était suffisant. En réalité, il y avait déjà avant une conversion des entiers fournis en nombre décimaux pour que l'appel se réalise correctement.

Voir également

initialiser()
avancer(float distance)
tournerGauche(float angle)
monterFeutre()

2.1.2 Documentation des fonctions

2.1.2.1 attendreBouton()

```
void attendreBouton ( )
```

Réalise l'attente nécessaire à la fonctionnalité du démarrage différé.

L'exécution de cette fonction bloquera tant que le bouton en question n'a pas été appuyé. Cette fonction est toujours appelée par initialiser(), car cela permet d'éviter que l'utilisateur du robot ne soit surpris par son démarrage alors que le câble de programmation est encore branché et que le robot n'est pas en place. Une petite attente après l'appui du bouton est aussi effectué pour ne pas que l'utilisateur ne trouve son doigt coincé dans les câblages après le départ du robot.

Définition à la ligne 243 du fichier Tortuino.cpp.

2.1.2.2 avancer()

```
void avancer (
            float distance )
```

Fait avancer le robot Tortuino d'une distance donnée.

Paramètres

distance	La distance en centimètres à parcourir.

Voir également

reculer(float distance)

Définition à la ligne 294 du fichier Tortuino.cpp.

2.1.2.3 descendreFeutre()

```
void descendreFeutre ( )
```

Place le feutre en position basse de telle manière qu'il touche la feuille en-dessous du robot, en supposant que le collier le tenant et permettant ce déplacement soit correctement ajusté.

Voir également

monterFeutre()

Définition à la ligne 378 du fichier Tortuino.cpp.

2.1.2.4 distanceToStep()

```
int distanceToStep (
            float distance )
```

Réalise la conversion d'une distance que le robot peut parcourir en un certain nombre de pas que chacun des deux moteurs pas à pas doit effectuer pour que le robot puisse avancer de la distance donnée.

Cette conversion prend en compte les paramètres décrivant la géométrie du robot.

Paramètres

distance	La distance linéaire en centimètres correspondant à un déplacement.

Renvoie

Le nombre de pas permettant de réaliser le déplacement de la distance donnée.

Définition à la ligne 145 du fichier Tortuino.cpp.

2.1.2.5 initialiser() [1/3]

```
void initialiser ( )
```

Initialise la configuration du robot pour qu'il puisse correctement communiquer avec ses différents composants qui le constituent : le servomoteur, les moteurs pas à pas et le bouton de démarrage différé.

Au cours de cette configuration, elle met le robot dans un état standard qui sera ainsi toujours le même au début de l'exécution de chaque essai : la vitesse de rotation des moteurs pas à pas est par défaut de 10 et le feutre est en position basse. Cette fonction à sa fin fait appel à attendreBouton() qui bloquera tant que le bouton de démarrage différé n'est pas appuyé.
L'initialisation est une étape absolument nécessaire au bon fonctionnement du robot ; sans cela, la carte Arduino que contrôle ces fonctions n'est pas en mesure de connaître les différents composants du robot, ni sur quels ports ils se trouvent et ni comment les utiliser. Les fonctions initialiser(char couleur) et initialiser(float braquage) ajoute à la présente fonction quelques détails en plus, mais finissent toujours par l'appeler. Il n'y a donc pas besoin d'appeler par exemple initialiser(float braquage) si cela se révèle utile pour vous et juste ensuite initialiser(), car cela sera redondant et même plutôt contre-productif. Un seul appel est suffisant.

Définition à la ligne 164 du fichier Tortuino.cpp.

2.1.2.6 initialiser() [2/3]

```
void initialiser (
            char couleur )
```

Cette version de l'opération d'initialisation met en place une méthode pour calibrer chaque robot à souhait pour que les erreurs systématiques au moment de la rotation puissent être compensées.

Le choix qui a été fait ici est de donner à chaque robot une couleur (par exemple de la plaquette d'expérimentation électrique) représentée par la première lettre de son écriture et qui l'identifie de manière unique. Ensuite, grâce à une correspondance établie au préalable, la valeur du rayon de braquage est affectée par cette fonction, retrouvant ainsi le calibrage effectué. Voir le code source pour bien comprendre comment ces associations sont réalisées.
Le reste de l'initialisation est bien entendu aussi réalisé.

Paramètres

couleur	La première lettre de la couleur identifiant le robot utilisé.

Voir également

initialiser()
initialiser(float braquage)

Définition à la ligne 186 du fichier Tortuino.cpp.

2.1.2.7 initialiser() [3/3]

```
void initialiser (
            float braquage )
```

Cette version de l'opération d'initialisation est utile à la calibration d'un robot car affecte directement au rayon de braquage la moitié de la valeur fournie en entrée.

Le reste de l'initialisation est bien entendu aussi réalisé.
Notez bien qu'un paramètre de type `float` est attendu en entrée. Or, d'autres fonctions existent mais ont des signatures différentes. En particulier, initialiser(char couleur) attend un caractère, ce qui revient à un nombre entier. Ainsi, si vous fournissez un nombre entier à la présente fonction, par exemple avec `initialiser(10);`, le compilateur C++ terminera sur une erreur car il y a précisément une ambiguïté entre initialiser(char couleur) et initialiser(float braquage). En effet, ce n'est pas au compilateur d'effectuer le choix implicite entre ces deux fonctions. Pour tout de même parvenir à ce que l'on souhaite dans cet exemple, il est possible de forcer le nombre entier `10` à être converti en `float` grâce à un `f` à la fin. Cela donne donc : `avancer(10f);`, et dans ce cas là, cela fonctionnera comme souhaité car cela appelera effectivement initialiser(float braquage). Si maintenant vous souhaitez appeler `initialiser(11.3);`, pour la même raison, il faut en fait faire `initialiser(11.↩ 3f);`.

Paramètres

braquage	La valeur du diamètre de braquage à utiliser.

Voir également

initialiser()
initialiser(char couleur)

Définition à la ligne 229 du fichier Tortuino.cpp.

2.1.2.8 monterFeutre()

```
void monterFeutre ( )
```

Place le feutre en position haute de telle manière qu'il ne touche pas la feuille en-dessous du robot, en supposant que le collier le tenant et permettant ce déplacement soit correctement ajusté.

Cela permet de choisir quand est-ce que l'on souhaite dessiner ou non, car à certains moments, par exemple si l'on veut se replacer pour continuer une autre partie d'un dessin, cela se révèle assez utile.

Voir également

descendreFeutre()

Définition à la ligne 366 du fichier Tortuino.cpp.

2.1.2.9 reculer()

```
void reculer (
            float distance )
```

Fait reculer le robot Tortuino d'une distance donnée.

Paramètres

distance	La distance en centimètres à parcourir.

Voir également

avancer(float distance)

Définition à la ligne 312 du fichier Tortuino.cpp.

2.1.2.10 stopper()

```
void stopper ( )
```

Permet de mettre à l'arrêt l'exécution en cours que réalise l'Arduino.

Cette fonction peut se révéler utile si le croquis Arduino utilise la fonction loop() mais souhaite à un moment donné stopper le robot. Cette instruction ne rend pas la main à toute exécution qui la suit, elle est donc plutôt à utiliser en dernière, à la toute fin de loop() si c'est bien ce qui est souhaité. Si par contre une simple pause est voulue, plusieurs options sont possibles, comme par exemple utiliser `delay()` pour attendre d'une durée donnée, puis continuer le flot d'exécution ; ou aussi attendreBouton() qui attend indéfiniment l'appui du bouton du robot et après rend la main.

Définition à la ligne 269 du fichier Tortuino.cpp.

2.1.2.11 tournerDroite()

```
void tournerDroite (
            float angle )
```

Fait tourner sur place le robot Tortuino d'un angle fourni vers sa droite.

La rotation s'effectue autour de l'axe de décrit le feutre positionné dans l'emplacement prévu à cet effet, de sorte que s'il reste baisser lors de l'opération, cela ne laisse pas de cercle de tracé derrière le robot. Il peut cependant avoir un petit décalage qui soit tout de même présent après rotation car en réalité le feutre n'est pas parfaitement stable : un petit jeu existe entre le feutre et son guide. L'effet de ce jeu n'est pas pour autant terrible et cela ne se verra pas trop.

Paramètres

angle	L'angle en degrés de rotation vers la gauche à effectuer.

Voir également

tournerGauche(float angle)

Définition à la ligne 353 du fichier Tortuino.cpp.

2.1.2.12 tournerGauche()

```
void tournerGauche (
            float angle )
```

Fait tourner sur place le robot Tortuino d'un angle fourni vers sa gauche.

La rotation s'effectue autour de l'axe que décrit le feutre positionné dans l'emplacement prévu à cet effet, de sorte que s'il reste baissé lors de l'opération, cela ne laisse pas de cercle de tracé derrière le robot. Il peut cependant avoir un petit décalage qui soit tout de même présent après rotation car en réalité le feutre n'est pas parfaitement stable : un petit jeu existe entre le feutre et son guide. L'effet de ce jeu n'est pas pour autant terrible et cela ne se verra pas trop.

Paramètres

angle	L'angle en degrés de rotation vers la gauche à effectuer.

Voir également

tournerDroite(float angle)

Définition à la ligne 328 du fichier Tortuino.cpp.

2.1.2.13 vitesse()

```
void vitesse (
            int v )
```

Règle la vitesse de rotation des moteurs pas à pas.

Une valeur de 10 est largement suffisante. Cette fonction n'est actuellement pas considérée comme intéressante à utilser en dehors du fonctionnement interne de cette bibliothèque. Elle est pour l'instant seulement appelée par initialiser() qui s'occupe de régler le robot pour avoir une vitesse par défaut qui fonctionne tout à fait correctement pour le robot ainsi paramétré.

Paramètres

v	La valeur entière de la vitesse à affecter aux moteurs pas à pas.

Définition à la ligne 284 du fichier Tortuino.cpp.

2.1.3 Documentation des variables

2.1.3.1 BRAQUAGE

```
float BRAQUAGE = 11.3 / 2
```

Le rayon de braquage du robot.

C'est une valeur qui peut être amenée à être calibrée.

Définition à la ligne 117 du fichier Tortuino.cpp.

2.1.3.2 delaiApresBouton

```
const int delaiApresBouton = 500
```

Le délai en ms effectué après que le bouton ait été pressé.

Il permet d'éviter que l'utilisateur se coince le doigt dans le câblage du robot au démarrage de l'exécution du programme de celui-ci.

Définition à la ligne 126 du fichier Tortuino.cpp.

2.1.3.3 delaiEntreBouton

```
const int delaiEntreBouton = 10
```

Le délai en ms entre chaque test du bouton.

Sa petite valeur importe peu, mais le délai reste utile.

Définition à la ligne 125 du fichier Tortuino.cpp.

2.1.3.4 delaiMonterDescendre

```
const int delaiMonterDescendre = 200
```

Le délai en ms d'attente après l'envoi d'une commande au feutre.

Paramétré empiriquement.

Définition à la ligne 129 du fichier Tortuino.cpp.

2.1.3.5 FEUTRE_BAS

```
const int FEUTRE_BAS = 10
```

L'angle de la position basse du servomoteur.

Il a été ajusté empiriquement.

Définition à la ligne 120 du fichier Tortuino.cpp.

2.1.3.6 FEUTRE_HAUT

```
const int FEUTRE_HAUT = 50
```

L'angle de la position haute du servomoteur.

Il a été ajusté empiriquement.

Définition à la ligne 119 du fichier Tortuino.cpp.

2.1.3.7 PERIMETER

```
float PERIMETER = M_PI * 9.2
```

Le périmètre des roues du robot tel que mesuré avec le pneu.

Définition à la ligne 116 du fichier Tortuino.cpp.

2.1.3.8 portBouton

```
const int portBouton = 7
```

Le numéro de la broche qui sert de port pour le bouton permettant le démarrage différé : 7.

Définition à la ligne 122 du fichier Tortuino.cpp.

2.1.3.9 portServo

```
const int portServo = 9
```

Le numéro de la broche pour le port du servomoteur : 9.

Définition à la ligne 123 du fichier Tortuino.cpp.

2.1.3.10 servo

```
Servo servo
```

L'objet qui sert à contrôler le servomoteur soulevant et abaissant le feutre du robot.

Définition à la ligne 134 du fichier Tortuino.cpp.

2.1.3.11 stepperLeft

```
Stepper stepperLeft = Stepper(stepsPerRevolution, 10, 12, 11, 13)
```

L'objet qui sert à contrôler le moteur pas à pas de gauche et qui est relié aux ports 10 à 13.

Définition à la ligne 131 du fichier Tortuino.cpp.

2.1.3.12 stepperRight

```
Stepper stepperRight = Stepper(stepsPerRevolution, 2, 4, 3, 5)
```

L'objet qui sert à contrôler le moteur pas à pas de droite et qui est relié aux ports 2 à 5.

Définition à la ligne 132 du fichier Tortuino.cpp.

2.1.3.13 stepsPerRevolution

```
const int stepsPerRevolution = 64 * 64 / 2
```

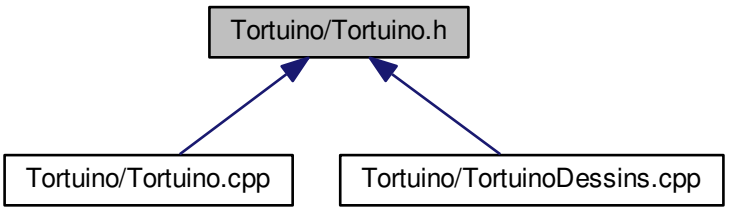
Le nombre de pas par tour que réalise un moteur pas à pas ; c'est une donnée constructeur.

Définition à la ligne 115 du fichier Tortuino.cpp.

2.2 Référence du fichier Tortuino/Tortuino.h

Définition des fonctions implémentées dans Tortuino.cpp.

Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Fonctions

- void initialiser ()
  Initialise la configuration du robot pour qu'il puisse correctement communiquer avec ses différents composants qui le constituent : le servomoteur, les moteurs pas à pas et le bouton de démarrage différé.
- void initialiser (float braquage)
  Cette version de l'opération d'initialisation est utile à la calibration d'un robot car affecte directement au rayon de braquage la moitié de la valeur fournie en entrée.
- void initialiser (char couleur)
  Cette version de l'opération d'initialisation met en place une méthode pour calibrer chaque robot à souhait pour que les erreurs systématiques au moment de la rotation puissent être compensées.
- void attendreBouton ()
  Réalise l'attente nécessaire à la fonctionnalité du démarrage différé.
- void stopper ()
  Permet de mettre à l'arrêt l'exécution en cours que réalise l'Arduino.
- void vitesse (int v)
  Règle la vitesse de rotation des moteurs pas à pas.
- void avancer (float distance)
  Fait avancer le robot Tortuino d'une distance donnée.
- void reculer (float distance)
  Fait reculer le robot Tortuino d'une distance donnée.
- void tournerGauche (float angle)
  Fait tourner sur place le robot Tortuino d'un angle fourni vers sa gauche.
- void tournerDroite (float angle)
  Fait tourner sur place le robot Tortuino d'un angle fourni vers sa droite.
- void monterFeutre ()
  Place le feutre en position haute de telle manière qu'il ne touche pas la feuille en-dessous du robot, en supposant que le collier le tenant et permettant ce déplacement soit correctement ajusté.
- void descendreFeutre ()
  Place le feutre en position basse de telle manière qu'il touche la feuille en-dessous du robot, en supposant que le collier le tenant et permettant ce déplacement soit correctement ajusté.

2.2.1 Description détaillée

Définition des fonctions implémentées dans Tortuino.cpp.

Version

1.1

Auteur

Paul Mabileau paulmabileau@hotmail.fr

Ce fichier constitue l'en-tête de Tortuino.cpp. Il permet de préciser ce qui sera rendu accessible à d'autres programmes. Ici, ce sont des fonctions.

2.2.2 Documentation des fonctions

2.2.2.1 attendreBouton()

```
void attendreBouton ( )
```

Réalise l'attente nécessaire à la fonctionnalité du démarrage différé.

L'exécution de cette fonction bloquera tant que le bouton en question n'a pas été appuyé. Cette fonction est toujours appelée par initialiser(), car cela permet d'éviter que l'utilisateur du robot ne soit surpris par son démarrage alors que le câble de programmation est encore branché et que le robot n'est pas en place. Une petite attente après l'appui du bouton est aussi effectué pour ne pas que l'utilisateur ne trouve son doigt coincé dans les câblages après le départ du robot.

Définition à la ligne 243 du fichier Tortuino.cpp.

2.2.2.2 avancer()

```
void avancer (
            float distance )
```

Fait avancer le robot Tortuino d'une distance donnée.

Paramètres

distance	La distance en centimètres à parcourir.

Voir également

reculer(float distance)

Définition à la ligne 294 du fichier Tortuino.cpp.

2.2.2.3 descendreFeutre()

```
void descendreFeutre ( )
```

Place le feutre en position basse de telle manière qu'il touche la feuille en-dessous du robot, en supposant que le collier le tenant et permettant ce déplacement soit correctement ajusté.

Voir également

monterFeutre()

Définition à la ligne 378 du fichier Tortuino.cpp.

2.2.2.4 initialiser() [1/3]

```
void initialiser ( )
```

Initialise la configuration du robot pour qu'il puisse correctement communiquer avec ses différents composants qui le constituent : le servomoteur, les moteurs pas à pas et le bouton de démarrage différé.

Au cours de cette configuration, elle met le robot dans un état standard qui sera ainsi toujours le même au début de l'exécution de chaque essai : la vitesse de rotation des moteurs pas à pas est par défaut de 10 et le feutre est en position basse. Cette fonction à sa fin fait appel à attendreBouton() qui bloquera tant que le bouton de démarrage différé n'est pas appuyé.
L'initialisation est une étape absolument nécessaire au bon fonctionnement du robot ; sans cela, la carte Arduino que contrôle ces fonctions n'est pas en mesure de connaître les différents composants du robot, ni sur quels ports ils se trouvent et ni comment les utiliser. Les fonctions initialiser(char couleur) et initialiser(float braquage) ajoute à la présente fonction quelques détails en plus, mais finissent toujours par l'appeler. Il n'y a donc pas besoin d'appeler par exemple initialiser(float braquage) si cela se révèle utile pour vous et juste ensuite initialiser(), car cela sera redondant et même plutôt contre-productif. Un seul appel est suffisant.

Définition à la ligne 164 du fichier Tortuino.cpp.

2.2.2.5 initialiser() [2/3]

```
void initialiser (
            float braquage )
```

Cette version de l'opération d'initialisation est utile à la calibration d'un robot car affecte directement au rayon de braquage la moitié de la valeur fournie en entrée.

Le reste de l'initialisation est bien entendu aussi réalisé.
Notez bien qu'un paramètre de type `float` est attendu en entrée. Or, d'autres fonctions existent mais ont des signatures différentes. En particulier, initialiser(char couleur) attend un caractère, ce qui revient à un nombre entier. Ainsi, si vous fournissez un nombre entier à la présente fonction, par exemple avec `initialiser(10);`, le compilateur C++ terminera sur une erreur car il y a précisément une ambiguïté entre initialiser(char couleur) et initialiser(float braquage). En effet, ce n'est pas au compilateur d'effectuer le choix implicite entre ces deux fonctions. Pour tout de même parvenir à ce que l'on souhaite dans cet exemple, il est possible de forcer le nombre entier `10` à être converti en `float` grâce à un `f` à la fin. Cela donne donc : `avancer(10f);`, et dans ce cas là, cela fonctionnera comme souhaité car cela appelera effectivement initialiser(float braquage). Si maintenant vous souhaitez appeler `initialiser(11.3);`, pour la même raison, il faut en fait faire `initialiser(11.↩3f);`.

Paramètres

braquage	La valeur du diamètre de braquage à utiliser.

Voir également

initialiser()
initialiser(char couleur)

Définition à la ligne 229 du fichier Tortuino.cpp.

2.2.2.6 initialiser() [3/3]

```
void initialiser (
            char couleur )
```

Cette version de l'opération d'initialisation met en place une méthode pour calibrer chaque robot à souhait pour que les erreurs systématiques au moment de la rotation puissent être compensées.

Le choix qui a été fait ici est de donner à chaque robot une couleur (par exemple de la plaquette d'expérimentation électrique) représentée par la première lettre de son écriture et qui l'identifie de manière unique. Ensuite, grâce à une correspondance établie au préalable, la valeur du rayon de braquage est affectée par cette fonction, retrouvant ainsi le calibrage effectué. Voir le code source pour bien comprendre comment ces associations sont réalisées.
Le reste de l'initialisation est bien entendu aussi réalisé.

Paramètres

couleur	La première lettre de la couleur identifiant le robot utilisé.

Voir également

initialiser()
initialiser(float braquage)

Définition à la ligne 186 du fichier Tortuino.cpp.

2.2.2.7 monterFeutre()

```
void monterFeutre ( )
```

Place le feutre en position haute de telle manière qu'il ne touche pas la feuille en-dessous du robot, en supposant que le collier le tenant et permettant ce déplacement soit correctement ajusté.

Cela permet de choisir quand est-ce que l'on souhaite dessiner ou non, car à certains moments, par exemple si l'on veut se replacer pour continuer une autre partie d'un dessin, cela se révèle assez utile.

Voir également

descendreFeutre()

Définition à la ligne 366 du fichier Tortuino.cpp.

2.2.2.8 reculer()

```
void reculer (
            float distance )
```

Fait reculer le robot Tortuino d'une distance donnée.

Paramètres

distance	La distance en centimètres à parcourir.

Voir également

avancer(float distance)

Définition à la ligne 312 du fichier Tortuino.cpp.

2.2.2.9 stopper()

```
void stopper ( )
```

Permet de mettre à l'arrêt l'exécution en cours que réalise l'Arduino.

Cette fonction peut se révéler utile si le croquis Arduino utilise la fonction loop() mais souhaite à un moment donné stopper le robot. Cette instruction ne rend pas la main à toute exécution qui la suit, elle est donc plutôt à utiliser en dernière, à la toute fin de loop() si c'est bien ce qui est souhaité. Si par contre une simple pause est voulue, plusieurs options sont possibles, comme par exemple utiliser `delay()` pour attendre d'une durée donnée, puis continuer le flot d'exécution ; ou aussi attendreBouton() qui attend indéfiniment l'appui du bouton du robot et après rend la main.

Définition à la ligne 269 du fichier Tortuino.cpp.

2.2.2.10 tournerDroite()

```
void tournerDroite (
            float angle )
```

Fait tourner sur place le robot Tortuino d'un angle fourni vers sa droite.

La rotation s'effectue autour de l'axe de décrit le feutre positionné dans l'emplacement prévu à cet effet, de sorte que s'il reste baisser lors de l'opération, cela ne laisse pas de cercle de tracé derrière le robot. Il peut cependant avoir un petit décalage qui soit tout de même présent après rotation car en réalité le feutre n'est pas parfaitement stable : un petit jeu existe entre le feutre et son guide. L'effet de ce jeu n'est pas pour autant terrible et cela ne se verra pas trop.

Paramètres

angle	L'angle en degrés de rotation vers la gauche à effectuer.

Voir également

tournerGauche(float angle)

Définition à la ligne 353 du fichier Tortuino.cpp.

2.2.2.11 tournerGauche()

```
void tournerGauche (
            float angle )
```

Fait tourner sur place le robot Tortuino d'un angle fourni vers sa gauche.

La rotation s'effectue autour de l'axe que décrit le feutre positionné dans l'emplacement prévu à cet effet, de sorte que s'il reste baissé lors de l'opération, cela ne laisse pas de cercle de tracé derrière le robot. Il peut cependant avoir un petit décalage qui soit tout de même présent après rotation car en réalité le feutre n'est pas parfaitement stable : un petit jeu existe entre le feutre et son guide. L'effet de ce jeu n'est pas pour autant terrible et cela ne se verra pas trop.

Paramètres

angle	L'angle en degrés de rotation vers la gauche à effectuer.

Voir également

tournerDroite(float angle)

Définition à la ligne 328 du fichier Tortuino.cpp.

2.2.2.12 vitesse()

```
void vitesse (
             int v )
```

Règle la vitesse de rotation des moteurs pas à pas.

Une valeur de 10 est largement suffisante. Cette fonction n'est actuellement pas considérée comme intéressante à utilser en dehors du fonctionnement interne de cette bibliothèque. Elle est pour l'instant seulement appelée par initialiser() qui s'occupe de régler le robot pour avoir une vitesse par défaut qui fonctionne tout à fait correctement pour le robot ainsi paramétré.

Paramètres

v	La valeur entière de la vitesse à affecter aux moteurs pas à pas.

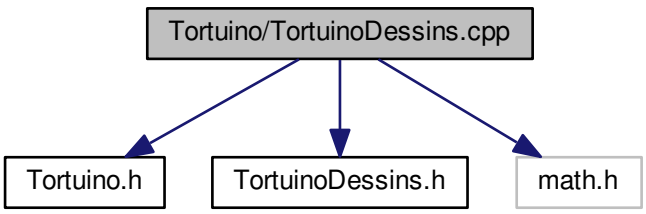
Définition à la ligne 284 du fichier Tortuino.cpp.

2.3 Référence du fichier Tortuino/TortuinoDessins.cpp

Ce fichier met à disposition quelques dessins qui peuvent être intéressants d'essayer.

```
#include "Tortuino.h"
#include "TortuinoDessins.h"
#include <math.h>
```

Graphe des dépendances par inclusion de TortuinoDessins.cpp :



Fonctions

- void polygoneRegulier (int nbCotes, float tailleCote)
  Fait tracer au robot un polygone régulier en fonction du nombre de côtés souhaités et de la taille de chacun de ces côtés.
- void triangle (float tailleCote)
  Trace un triangle equilatéral d'une certaine taille.
- void carre (float tailleCote)
  Trace un carré d'une certaine taille.
- void cercle (float rayon)
  Cette fonction est une tentative de réalisation d'un cercle automatiquement avec juste le rayon souhaité en entrée.
- void arbre (int nbNiveaux, float tailleTronc)
  Trace un arbre récursivement dont l'angle entre les branches est de 90 degrés et qui est symétrique par rapport à l'axe formé par son tronc.
- void arbreSymetrique (int nbNiveaux, float tailleTronc, float angleSeparation)
  Trace un arbre récursivement dont l'angle entre les branches peut être précisé et qui est symétrique par rapport à l'axe formé par son tronc.
- void arbreAsymetrique (int nbNiveaux, float tailleTronc, float angleSeparation, float angleInclinaison)
  Trace un arbre récursivement dont l'angle entre les branches et l'angle entre la branche de gauche et la branche mère moins 45 degrés peuvent être précisés : il est asymétrique par rapport à l'axe formé par son tronc.
- void sapin (int nbNiveaux, float tailleTronc)
  Utilise deux arbres asymétriques pour tracer un sapin, c'est-à-dire un arbre où chaque branche se sépare en trois autres : une continuant vers le haut (le tronc donc) et deux horizontales sur le côté (les branches donc).
- void courbeVonKoch (int nbNiveaux, float taille)
  Trace une `courbe de Von Koch` *paramétrée par son niveau et la taille du segment de départ.*
- void floconVonKoch (int nbNiveaux, float taille)
  Trace un `flocon de Von Koch` *paramétré par son niveau et la taille du segment de départ.*
- void triangleSierpinski (int nbNiveaux, float taille)
  Trace un `triangle de Sierpiński` *paramétré par son niveau et la taille globale du triangle.*
- void maison ()
  Réalise le premier défi que nous proposons pour l'animation Tortuino : une maison avec son toit et sa porte d'entrée.
- void spiraleCarree (int nbCotes, float longueurDepart, float ecart)
  Trace une spirale carrée en fonction du nombre de côtés souhaités, de la longueur du côté de départ et de l'écart en longueur entre deux côtés adjacents.
- void tangram ()
  Dessine le troisième défi de l'animation : un carré divisé en pièces à tangram, nom raccourci en juste tangram.
- void flocon ()
  Réalise le quatrième défi proposé : un flocon à huit branches.

2.3.1 Description détaillée

Ce fichier met à disposition quelques dessins qui peuvent être intéressants d'essayer.

Auteur

Paul Mabileau `paulmabileau@hotmail.fr`

Version

1.3

Le fichier TortuinoDessins.cpp implémente un ensemble de fonctions réalisant quelques dessins plus ou moins complexes. Les dessins les plus simples sont par exemple des polygones réguliers tels qu'un triangle, un carré, un hexagone, ... les plus compliqués utilise des motifs récursifs, ce qui est moins simple à programmer, mais tout à fait agréable à contempler, comme par exemple un arbre avec différentes variantes, un flocon de Von Koch ou encore le triangle de Sierpiński. Il présente aussi quelques défis que nous proposons pour l'animation Tortuino et qui ont été réalisés avec succès par les stagiaires d'essai : une maison, une spirale carrée, un tangram et un flocon. Ils se trouvent dans la section défis de ce fichier.

2.3.2 Documentation des fonctions

2.3.2.1 arbre()

```
void arbre (
            int nbNiveaux,
            float tailleTronc )
```

Trace un arbre récursivement dont l'angle entre les branches est de 90 degrés et qui est symétrique par rapport à l'axe formé par son tronc.

C'est donc un cas particulier de arbreSymetrique(int nbNiveaux, float tailleTronc, float angleSeparation).

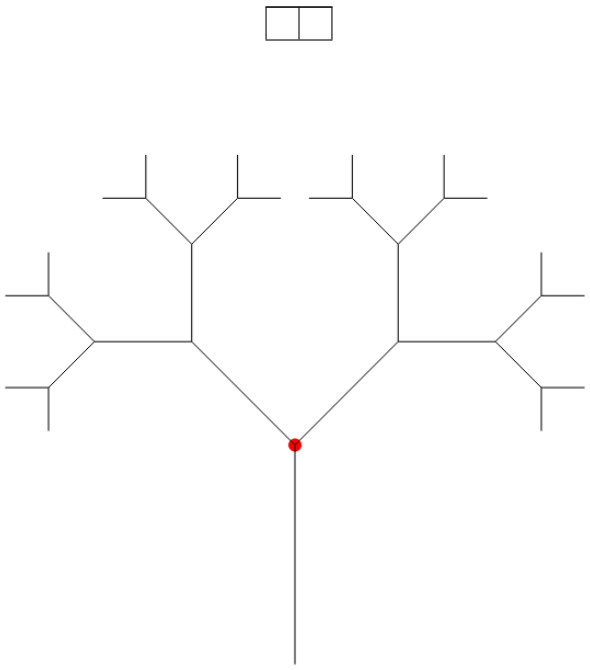


FIGURE 2.1 Un arbre à cinq niveaux

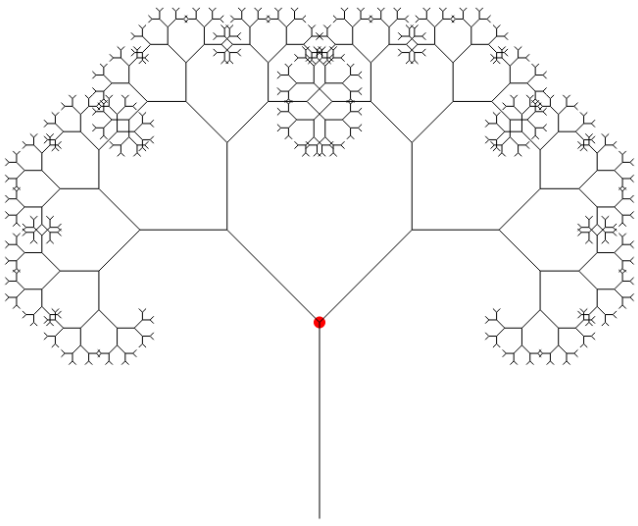


FIGURE 2.2 Un arbre à dix niveaux

Paramètres

nbNiveaux	Le nombre de niveaux que l'arbre comprendra, c'est-à-dire le nombre de fois moins un que l'arbre va se séparer ou autrement dit la distance en nombre de branches entre la racine et chaque feuille.
tailleTronc	La taille du tronc de départ. Les branches qui en partiront auront leurs tailles d'un tier plus petites.

Définition à la ligne 118 du fichier TortuinoDessins.cpp.

2.3.2.2 arbreAsymetrique()

```
void arbreAsymetrique (
            int nbNiveaux,
            float tailleTronc,
            float angleSeparation,
            float angleInclinaison )
```

Trace un arbre récursivement dont l'angle entre les branches et l'angle entre la branche de gauche et la branche mère moins 45 degrés peuvent être précisés : il est asymétrique par rapport à l'axe formé par son tronc.

Il généralise donc un arbre.

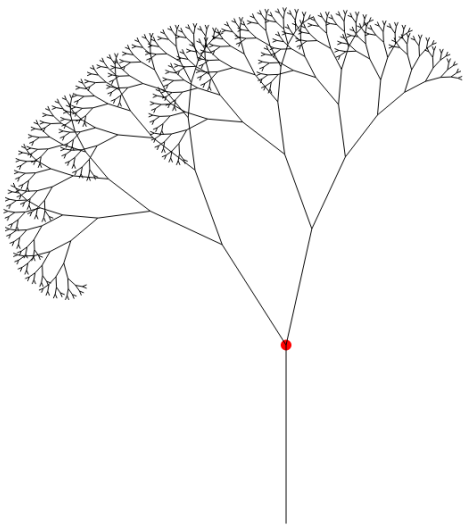


FIGURE 2.3 Un arbre asymétrique à dix niveaux séparés de 45° et inclinés de 10°

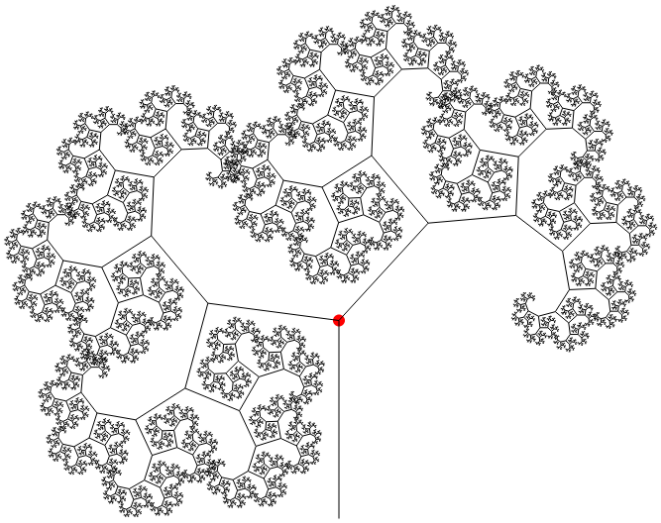


FIGURE 2.4 Un arbre asymétrique à quinze niveaux séparés de 125° et inclinés de 20°

Paramètres

nbNiveaux	Le nombre de niveaux que l'arbre comprendra, c'est-à-dire le nombre de fois moins un que l'arbre va se séparer ou autrement dit la distance en nombre de branches entre la racine et chaque feuille.
tailleTronc	La taille du tronc de départ. Les branches qui en partiront auront leurs tailles d'un tier plus petites.
angleSeparation	L'angle séparant les branches provenant d'une même branche mère.
angleInclinaison	L'angle entre la branche de gauche et la branche mère moins 45 degrés.

Voir également

arbre(int nbNiveaux, float tailleTronc)
arbreSymetrique(int nbNiveaux, float tailleTronc, float angleSeparation)

Définition à la ligne 174 du fichier TortuinoDessins.cpp.

2.3.2.3 arbreSymetrique()

```
void arbreSymetrique (
            int nbNiveaux,
            float tailleTronc,
            float angleSeparation )
```

Trace un arbre récursivement dont l'angle entre les branches peut être précisé et qui est symétrique par rapport à l'axe formé par son tronc.

C'est donc un cas particulier de arbreAsymetrique(int nbNiveaux, float tailleTronc, float angleSeparation, float angleInclinaison).

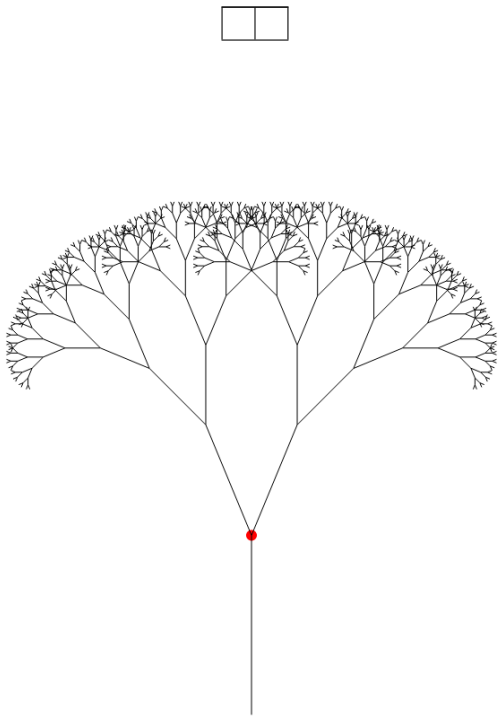


FIGURE 2.5 Un arbre symétrique à dix niveaux séparés de 45°

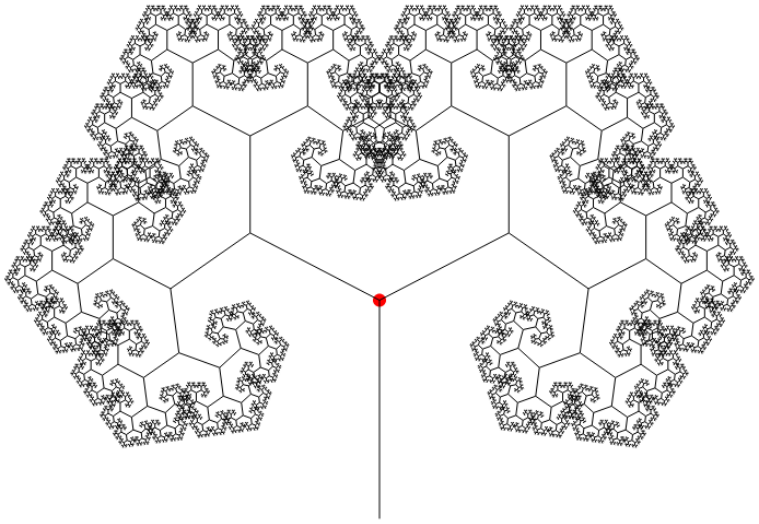


FIGURE 2.6 Un arbre symétrique à quinze niveaux séparés de 125°

Paramètres

nbNiveaux	Le nombre de niveaux que l'arbre comprendra, c'est-à-dire le nombre de fois moins un que l'arbre va se séparer ou autrement dit la distance en nombre de branches entre la racine et chaque feuille.
tailleTronc	La taille du tronc de départ. Les branches qui en partiront auront leurs tailles d'un tier plus petites.
angleSeparation	L'angle séparant les branches provenant d'une même branche mère.

Voir également

arbre(int nbNiveaux, float tailleTronc)

Définition à la ligne 145 du fichier TortuinoDessins.cpp.

2.3.2.4 carre()

```
void carre (
            float tailleCote )
```

Trace un carré d'une certaine taille.

En réalité, ce n'est qu'une adaptation de polygoneRegulier(int nbCotes, float tailleCote) au cas particulier du carré.

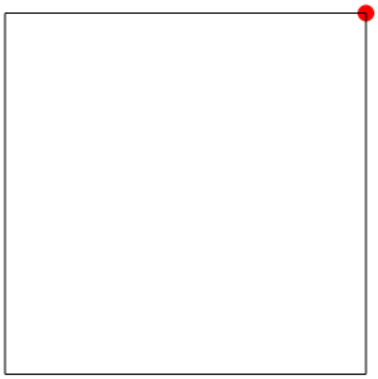


FIGURE 2.7 Un carré

Paramètres

tailleCote	La taille des côtés du carré.

Définition à la ligne 78 du fichier TortuinoDessins.cpp.

2.3.2.5 cercle()

```
void cercle (
            float rayon )
```

Cette fonction est une tentative de réalisation d'un cercle automatiquement avec juste le rayon souhaité en entrée.

Seulement, cela ne fonctionne pas trop car il est difficile de décorréler les paramètres du robot pour pouvoir calculer les valeurs nécessaires à une approximation relativement correcte d'un cercle par un polygone régulier au grand nombre de côtés. Cette fonction est en cours développement, une meilleure version peut venir à être rendue disponble.

Paramètres

rayon	Le rayon du cercle.

Définition à la ligne 92 du fichier TortuinoDessins.cpp.

2.3.2.6 courbeVonKoch()

```
void courbeVonKoch (
            int nbNiveaux,
            float taille )
```

Trace une courbe de Von Koch paramétrée par son niveau et la taille du segment de départ.

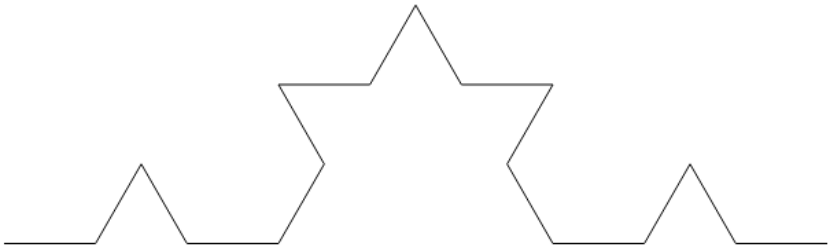


FIGURE 2.8 Une courbe de Von Koch à trois niveaux

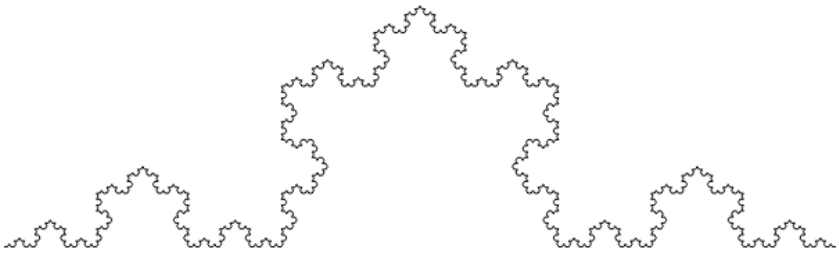


FIGURE 2.9 Une courbe de Von Koch à six niveaux

Paramètres

nbNiveaux	Le nombre de niveaux de la courbe de Von Koch, par convention 1 donne un trait seulement.
taille	La taille du segment à partir de laquelle l'opération récursive est itérée, c'est-à-dire que contrairement à un arbre ou un polygone tels qu'implémentés ici, ajouter plus de niveaux ou de côtés n'augmente pas la taille globale de la courbe ; autrement dit, le segment de départ sert d'étalon pour en déduire à l'avance la taille des côtés engendrés.

Définition à la ligne 239 du fichier TortuinoDessins.cpp.

2.3.2.7 flocon()

```
void flocon ( )
```

Réalise le quatrième défi proposé : un flocon à huit branches.

Chacune des branches comporte trois "feuilles" de sorte qu'elles aient leurs terminaisons alignées. Il a aussi sa forme fixée.

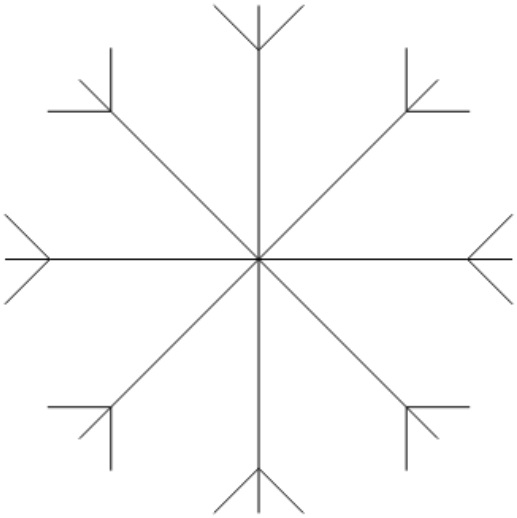


FIGURE 2.10 Le défi flocon

Définition à la ligne 443 du fichier TortuinoDessins.cpp.

2.3.2.8 floconVonKoch()

```
void floconVonKoch (
            int nbNiveaux,
            float taille )
```

Trace un flocon de Von Koch paramétré par son niveau et la taille du segment de départ.

C'est en fait une répétition de la courbeVonKoch(int nbNiveaux, float taille) : six fois séparées par un angle intérieur de 120 degrés.

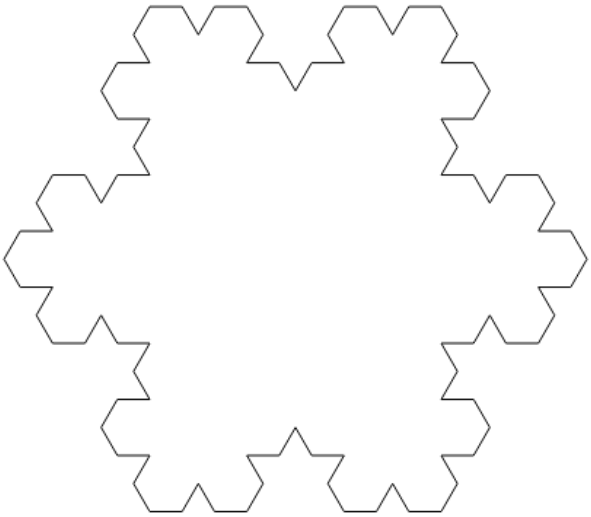


FIGURE 2.11 Un flocon de Von Koch à trois niveaux

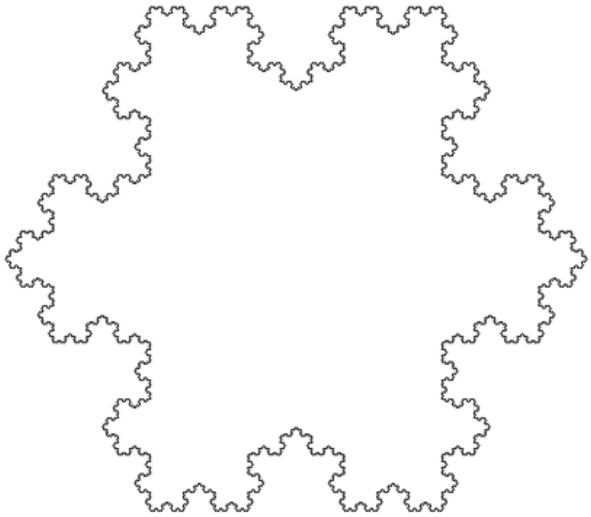


FIGURE 2.12 Un flocon de Von Koch à six niveaux

Paramètres

nbNiveaux	Le nombre de niveaux du flocon de Von Koch, par convention 1 donne un triangle seulement.
taille	La taille du segment à partir de laquelle l'opération récursive est itérée, c'est-à-dire que contrairement à un arbre ou un polygone tels qu'implémentés ici, ajouter plus de niveaux ou de côtés n'augmente pas la taille globale du flocon ; autrement dit, le segment de départ sert d'étalon pour en déduire à l'avance la taille des côtés engendrés.

Définition à la ligne 275 du fichier TortuinoDessins.cpp.

2.3.2.9 maison()

```
void maison ( )
```

Réalise le premier défi que nous proposons pour l'animation Tortuino : une maison avec son toit et sa porte d'entrée.

Elle n'accepte pas de paramètres en entrée, car sa forme est fixée et que mettre à disposition des moyens de personnaliser chaque partie de la maison est long et en soi pas particulièrement intéressant. Le dessin est ainsi toujours le même.

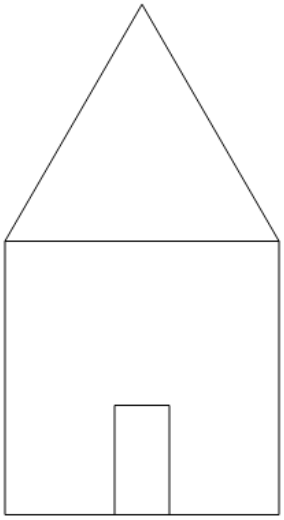


FIGURE 2.13 Le défi maison

Définition à la ligne 337 du fichier TortuinoDessins.cpp.

2.3.2.10 polygoneRegulier()

```
void polygoneRegulier (
            int nbCotes,
            float tailleCote )
```

Fait tracer au robot un polygone régulier en fonction du nombre de côtés souhaités et de la taille de chacun de ces côtés.

Voir l'article Wikipédia suivant pour plus de détails sur cette figure géométrique. Un nombre de côtés de 3 donne un triangle, 4 un carré, 5 un pentagone régulier, ...

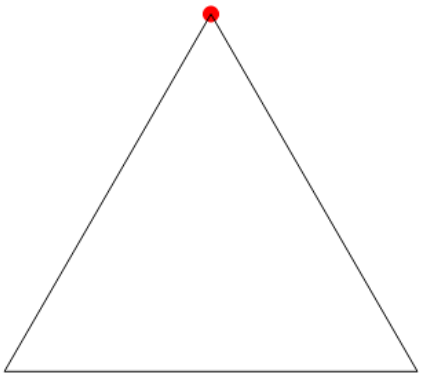


FIGURE 2.14 Un triangle

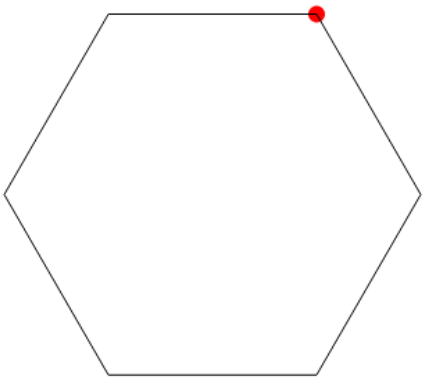


FIGURE 2.15 Un hexagone

Paramètres

nbCotes	Le nombre de côtés du polygone à tracer.
tailleCote	La taille de chacun des côtés.

Définition à la ligne 47 du fichier TortuinoDessins.cpp.

2.3.2.11 sapin()

```
void sapin (
            int nbNiveaux,
            float tailleTronc )
```

Utilise deux arbres asymétriques pour tracer un sapin, c'est-à-dire un arbre où chaque branche se sépare en trois autres : une continuant vers le haut (le tronc donc) et deux horizontales sur le côté (les branches donc).

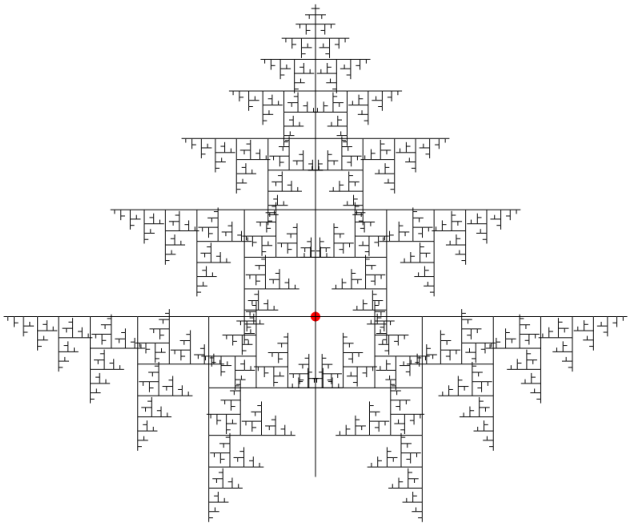


FIGURE 2.16 Un sapin à dix niveaux

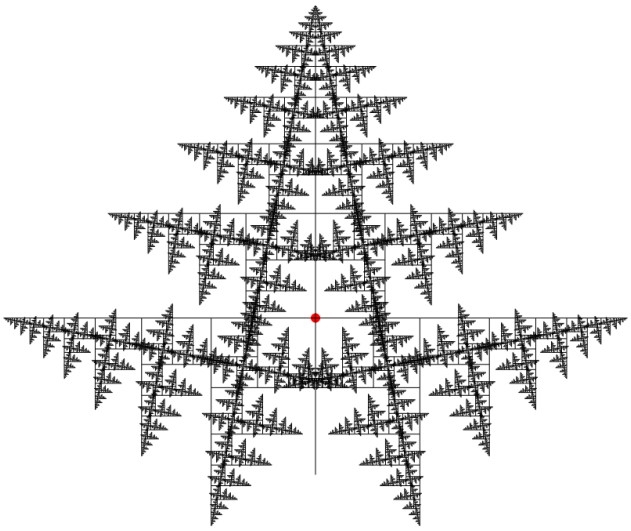


FIGURE 2.17 Un sapin à quinze niveaux

Paramètres

nbNiveaux	Le nombre de niveaux que l'arbre comprendra, c'est-à-dire le nombre de fois moins un que l'arbre va se séparer ou autrement dit la distance en nombre de branches entre la racine et chaque feuille.
tailleTronc	La taille du tronc de départ. Les branches qui en partiront auront leurs tailles d'un tier plus petites.

Voir également

arbre(int nbNiveaux, float tailleTronc)

Définition à la ligne 214 du fichier TortuinoDessins.cpp.

2.3.2.12 spiraleCarree()

```
void spiraleCarree (
            int nbCotes,
            float longueurDepart,
            float ecart )
```

Trace une spirale carrée en fonction du nombre de côtés souhaités, de la longueur du côté de départ et de l'écart en longueur entre deux côtés adjacents.

C'est l'implémentation du deuxième défi Tortuino. Une spirale carrée est une spirale composés d'un certain nombre de segments reliés l'un à la suite de l'autre et séparés d'un angle de droit.

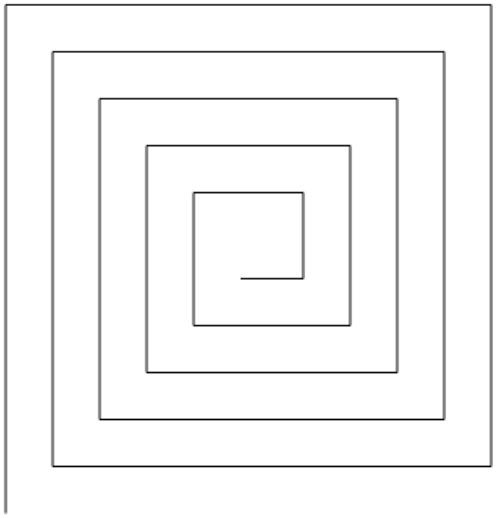


FIGURE 2.18 Une spirale carrée à vingt côtés avec une longueur de départ de 2 et un écart de 3

Paramètres

nbCotes	Le nombre de côtés de la spirale à tracer. 1 fait un seul segment.
longueurDepart	La longueur en centimètres du premier côté de la spirale.
ecart	L'écart en centimètres entre deux côtés adjacents de la spirale, c'est-à-dire entre les côtés qui se trouvent dans la même direction à partir du point de départ - au nord, au sud, à l'est ou à l'ouest. Ce n'est pas directement l'écart entre la longueur du côté en cours et son suivant, mais plutôt son quadruple.

Définition à la ligne 380 du fichier TortuinoDessins.cpp.

2.3.2.13 tangram()

```
void tangram ( )
```

Dessine le troisième défi de l'animation : un carré divisé en pièces à tangram, nom raccourci en juste tangram.

Le carré en question est composé de cinq triangles, un plus petit carré et un parallélogramme. De même que la maison(), la geométrie est fixée et n'accepte donc pas de paramètres permettant d'ajuster sa forme.

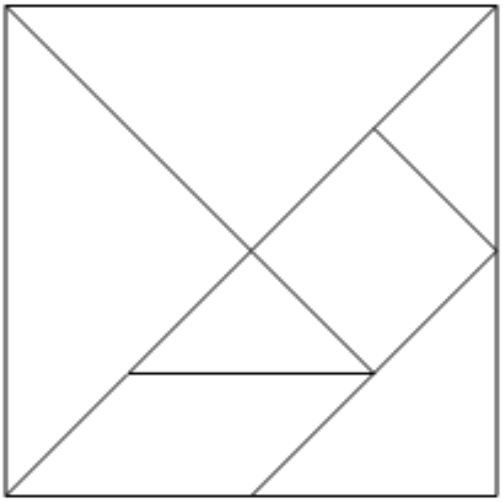


FIGURE 2.19 Le défi tangram

Définition à la ligne 400 du fichier TortuinoDessins.cpp.

2.3.2.14 triangle()

```
void triangle (
            float tailleCote )
```

Trace un triangle equilatéral d'une certaine taille.

En réalité, ce n'est qu'une adaptation de polygoneRegulier(int nbCotes, float tailleCote) au cas particulier du triangle equilatéral.

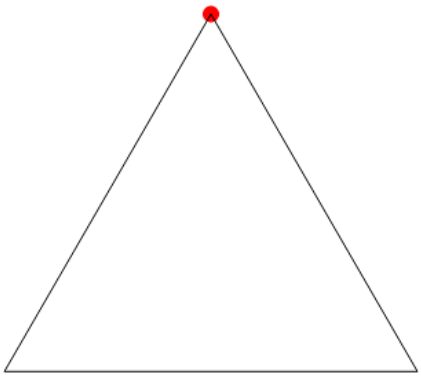


FIGURE 2.20 Un triangle

Paramètres

tailleCote	La taille des côtés du triangle.

Définition à la ligne 64 du fichier TortuinoDessins.cpp.

2.3.2.15 triangleSierpinski()

```
void triangleSierpinski (
            int nbNiveaux,
            float taille )
```

Trace un triangle de Sierpiński paramétré par son niveau et la taille globale du triangle.

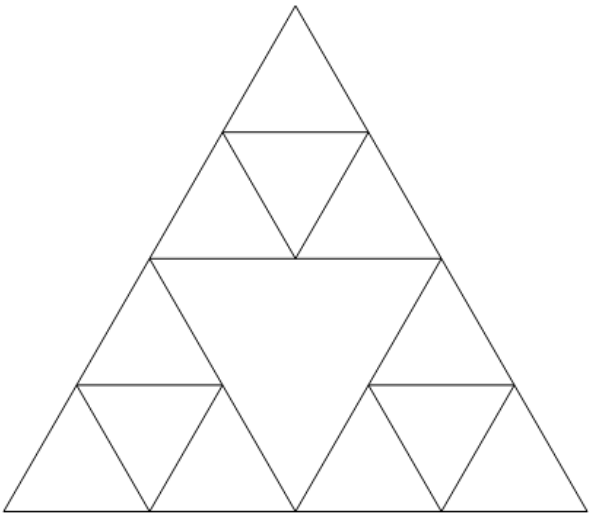


FIGURE 2.21 Un triangle de Sierpinski à trois niveaux

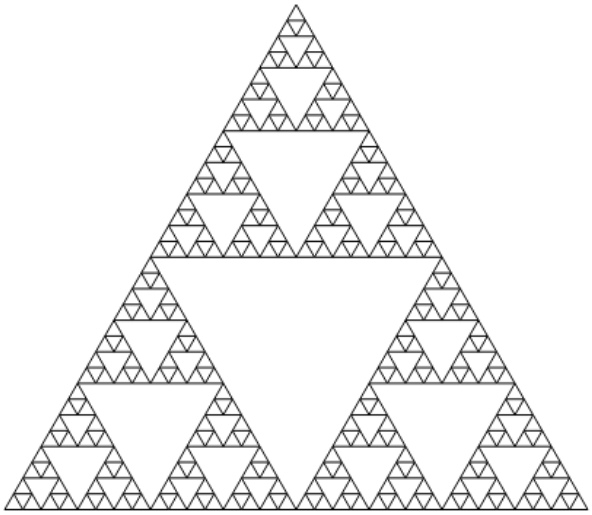


FIGURE 2.22 Un triangle de Sierpinski à six niveaux

Paramètres

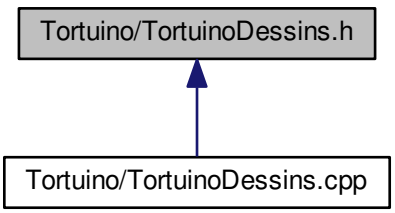
nbNiveaux	Le nombre de niveaux du triangle de Sierpiński.
taille	La taille du segment de départ qui sera conservée au fur et à mesure des itérations de l'algorithme de Sierpiński ; idem à ce que fait floconVonKoch(int nbNiveaux, float taille)

Définition à la ligne 300 du fichier TortuinoDessins.cpp.

2.4 Référence du fichier Tortuino/TortuinoDessins.h

Définition des fonctions implémentées dans TortuinoDessins.cpp.

Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Fonctions

- void triangle (float tailleCote)
  Trace un triangle equilatéral d'une certaine taille.
- void carre (float tailleCote)
  Trace un carré d'une certaine taille.
- void polygoneRegulier (int nbCotes, float tailleCote)
  Fait tracer au robot un polygone régulier en fonction du nombre de côtés souhaités et de la taille de chacun de ces côtés.
- void cercle (float rayon)
  Cette fonction est une tentative de réalisation d'un cercle automatiquement avec juste le rayon souhaité en entrée.
- void arbre (int nbNiveaux, float tailleTronc)
  Trace un arbre récursivement dont l'angle entre les branches est de 90 degrés et qui est symétrique par rapport à l'axe formé par son tronc.
- void arbreSymetrique (int nbNiveaux, float tailleTronc, float angleSeparation)
  Trace un arbre récursivement dont l'angle entre les branches peut être précisé et qui est symétrique par rapport à l'axe formé par son tronc.
- void arbreAsymetrique (int nbNiveaux, float tailleTronc, float angleSeparation, float angleInclinaison)
  Trace un arbre récursivement dont l'angle entre les branches et l'angle entre la branche de gauche et la branche mère moins 45 degrés peuvent être précisés : il est asymétrique par rapport à l'axe formé par son tronc.
- void sapin (int nbNiveaux, float tailleTronc)
  Utilise deux arbres asymétriques pour tracer un sapin, c'est-à-dire un arbre où chaque branche se sépare en trois autres : une continuant vers le haut (le tronc donc) et deux horizontales sur le côté (les branches donc).
- void courbeVonKoch (int nbNiveaux, float taille)
  Trace une `courbe de Von Koch` paramétrée par son niveau et la taille du segment de départ.
- void floconVonKoch (int nbNiveaux, float taille)
  Trace un `flocon de Von Koch` paramétré par son niveau et la taille du segment de départ.
- void triangleSierpinski (int nbNiveaux, float taille)
  Trace un `triangle de Sierpiński` paramétré par son niveau et la taille globale du triangle.
- void maison ()
  Réalise le premier défi que nous proposons pour l'animation Tortuino : une maison avec son toit et sa porte d'entrée.
- void spiraleCarree (int nbCotes, float longueurDepart, float ecart)
  Trace une spirale carrée en fonction du nombre de côtés souhaités, de la longueur du côté de départ et de l'écart en longueur entre deux côtés adjacents.
- void tangram ()
  Dessine le troisième défi de l'animation : un carré divisé en pièces à tangram, nom raccourci en juste tangram.
- void flocon ()
  Réalise le quatrième défi proposé : un flocon à huit branches.

2.4.1 Description détaillée

Définition des fonctions implémentées dans TortuinoDessins.cpp.

Version

1.1

Auteur

Paul Mabileau `paulmabileau@hotmail.fr`

Ce fichier constitue l'en-tête de TortuinoDessins.cpp. Il permet de préciser ce qui sera rendu accessible à d'autres programmes. Ici, ce sont des fonctions.

2.4.2 Documentation des fonctions

2.4.2.1 arbre()

```
void arbre (
            int nbNiveaux,
            float tailleTronc )
```

Trace un arbre récursivement dont l'angle entre les branches est de 90 degrés et qui est symétrique par rapport à l'axe formé par son tronc.

C'est donc un cas particulier de arbreSymetrique(int nbNiveaux, float tailleTronc, float angleSeparation).

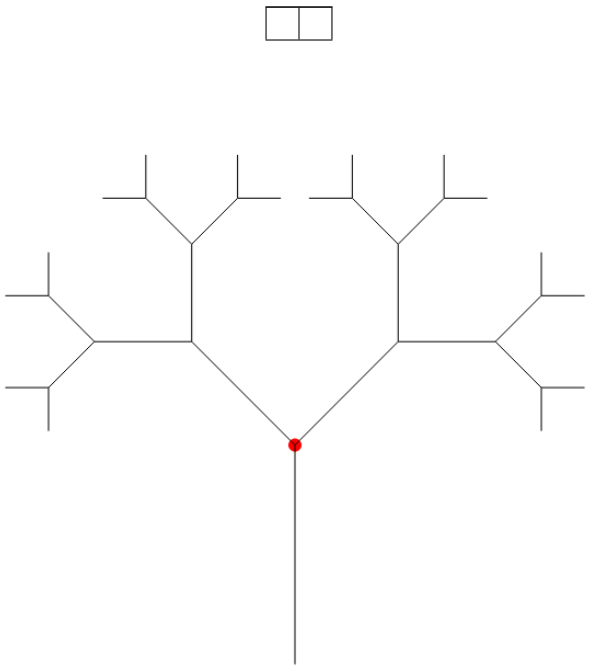


FIGURE 2.23 Un arbre à cinq niveaux

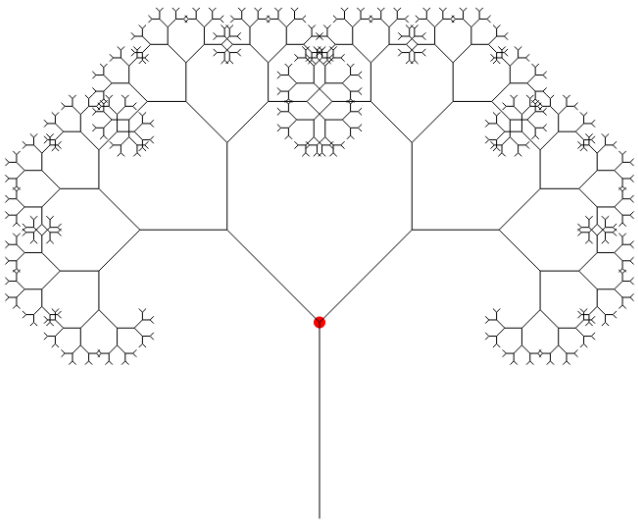


FIGURE 2.24 Un arbre à dix niveaux

Paramètres

nbNiveaux	Le nombre de niveaux que l'arbre comprendra, c'est-à-dire le nombre de fois moins un que l'arbre va se séparer ou autrement dit la distance en nombre de branches entre la racine et chaque feuille.
tailleTronc	La taille du tronc de départ. Les branches qui en partiront auront leurs tailles d'un tier plus petites.

Définition à la ligne 118 du fichier TortuinoDessins.cpp.

2.4.2.2 arbreAsymetrique()

```
void arbreAsymetrique (
            int nbNiveaux,
            float tailleTronc,
            float angleSeparation,
            float angleInclinaison )
```

Trace un arbre récursivement dont l'angle entre les branches et l'angle entre la branche de gauche et la branche mère moins 45 degrés peuvent être précisés : il est asymétrique par rapport à l'axe formé par son tronc.

Il généralise donc un arbre.

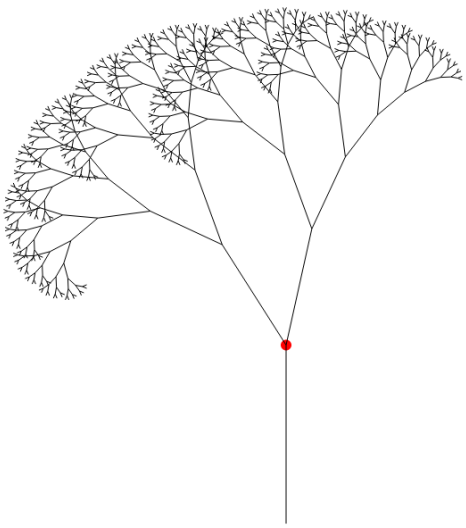


FIGURE 2.25 Un arbre asymétrique à dix niveaux séparés de 45° et inclinés de 10°

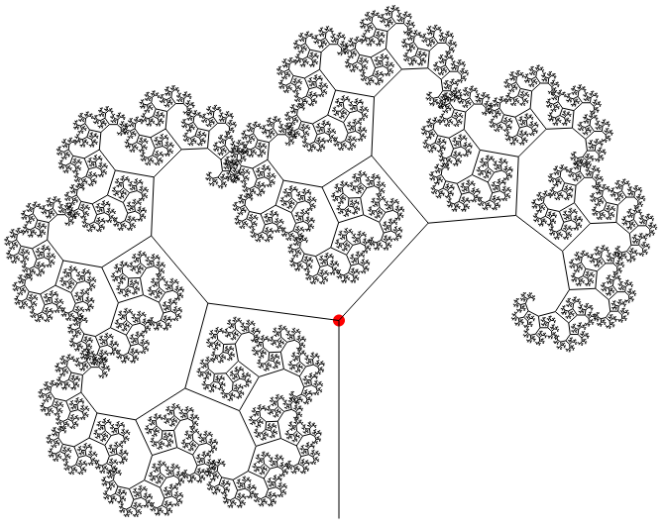


FIGURE 2.26 Un arbre asymétrique à quinze niveaux séparés de 125° et inclinés de 20°

Paramètres

nbNiveaux	Le nombre de niveaux que l'arbre comprendra, c'est-à-dire le nombre de fois moins un que l'arbre va se séparer ou autrement dit la distance en nombre de branches entre la racine et chaque feuille.
tailleTronc	La taille du tronc de départ. Les branches qui en partiront auront leurs tailles d'un tier plus petites.
angleSeparation	L'angle séparant les branches provenant d'une même branche mère.
angleInclinaison	L'angle entre la branche de gauche et la branche mère moins 45 degrés.

Voir également

arbre(int nbNiveaux, float tailleTronc)
arbreSymetrique(int nbNiveaux, float tailleTronc, float angleSeparation)

Définition à la ligne 174 du fichier TortuinoDessins.cpp.

2.4.2.3 arbreSymetrique()

```
void arbreSymetrique (
            int nbNiveaux,
            float tailleTronc,
            float angleSeparation )
```

Trace un arbre récursivement dont l'angle entre les branches peut être précisé et qui est symétrique par rapport à l'axe formé par son tronc.

C'est donc un cas particulier de arbreAsymetrique(int nbNiveaux, float tailleTronc, float angleSeparation, float angleInclinaison).

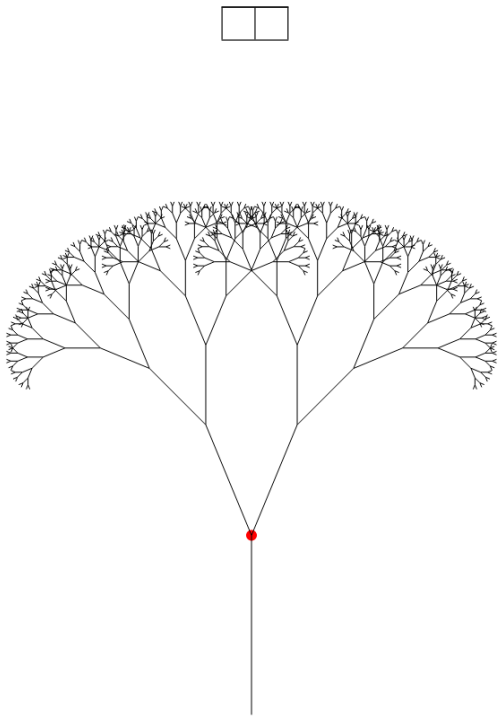


FIGURE 2.27 Un arbre symétrique à dix niveaux séparés de 45°

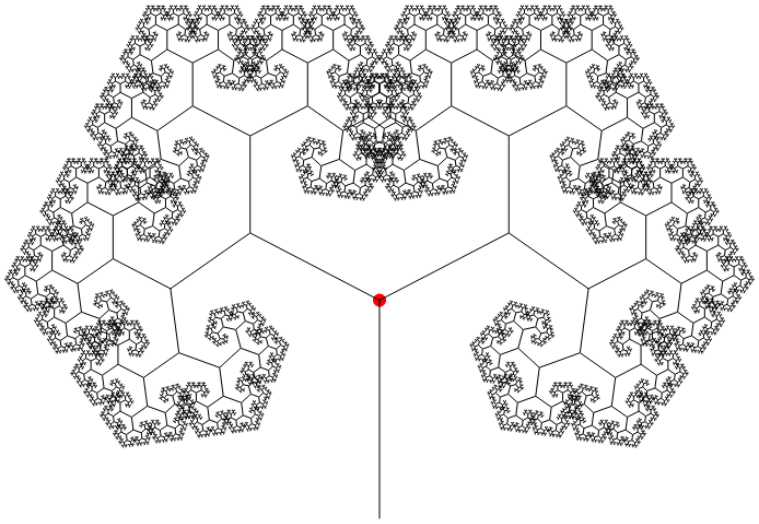


FIGURE 2.28 Un arbre symétrique à quinze niveaux séparés de 125°

Paramètres

nbNiveaux	Le nombre de niveaux que l'arbre comprendra, c'est-à-dire le nombre de fois moins un que l'arbre va se séparer ou autrement dit la distance en nombre de branches entre la racine et chaque feuille.
tailleTronc	La taille du tronc de départ. Les branches qui en partiront auront leurs tailles d'un tier plus petites.
angleSeparation	L'angle séparant les branches provenant d'une même branche mère.

Voir également

arbre(int nbNiveaux, float tailleTronc)

Définition à la ligne 145 du fichier TortuinoDessins.cpp.

2.4.2.4 carre()

```
void carre (
            float tailleCote )
```

Trace un carré d'une certaine taille.

En réalité, ce n'est qu'une adaptation de polygoneRegulier(int nbCotes, float tailleCote) au cas particulier du carré.

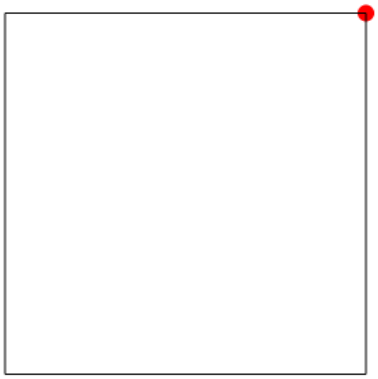


FIGURE 2.29 Un carré

Paramètres

tailleCote	La taille des côtés du carré.

Définition à la ligne 78 du fichier TortuinoDessins.cpp.

2.4.2.5 cercle()

```
void cercle (
            float rayon )
```

Cette fonction est une tentative de réalisation d'un cercle automatiquement avec juste le rayon souhaité en entrée.

Seulement, cela ne fonctionne pas trop car il est difficile de décorréler les paramètres du robot pour pouvoir calculer les valeurs nécessaires à une approximation relativement correcte d'un cercle par un polygone régulier au grand nombre de côtés. Cette fonction est en cours développement, une meilleure version peut venir à être rendue disponble.

Paramètres

rayon	Le rayon du cercle.

Définition à la ligne 92 du fichier TortuinoDessins.cpp.

2.4.2.6 courbeVonKoch()

```
void courbeVonKoch (
            int nbNiveaux,
            float taille )
```

Trace une courbe de Von Koch paramétrée par son niveau et la taille du segment de départ.

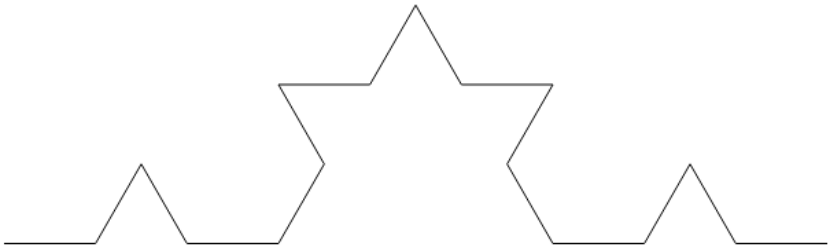


FIGURE 2.30 Une courbe de Von Koch à trois niveaux

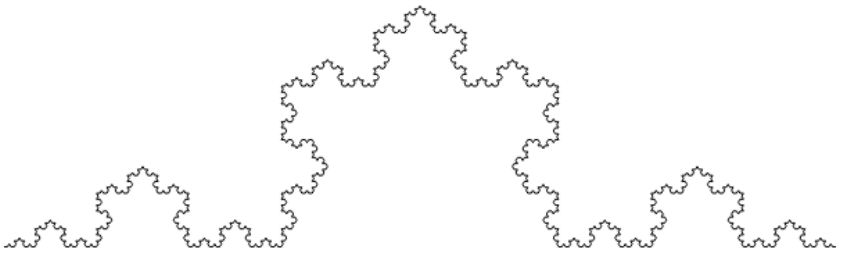


FIGURE 2.31 Une courbe de Von Koch à six niveaux

Paramètres

nbNiveaux	Le nombre de niveaux de la courbe de Von Koch, par convention 1 donne un trait seulement.
taille	La taille du segment à partir de laquelle l'opération récursive est itérée, c'est-à-dire que contrairement à un arbre ou un polygone tels qu'implémentés ici, ajouter plus de niveaux ou de côtés n'augmente pas la taille globale de la courbe ; autrement dit, le segment de départ sert d'étalon pour en déduire à l'avance la taille des côtés engendrés.

Définition à la ligne 239 du fichier TortuinoDessins.cpp.

2.4.2.7 flocon()

```
void flocon ( )
```

Réalise le quatrième défi proposé : un flocon à huit branches.

Chacune des branches comporte trois "feuilles" de sorte qu'elles aient leurs terminaisons alignées. Il a aussi sa forme fixée.

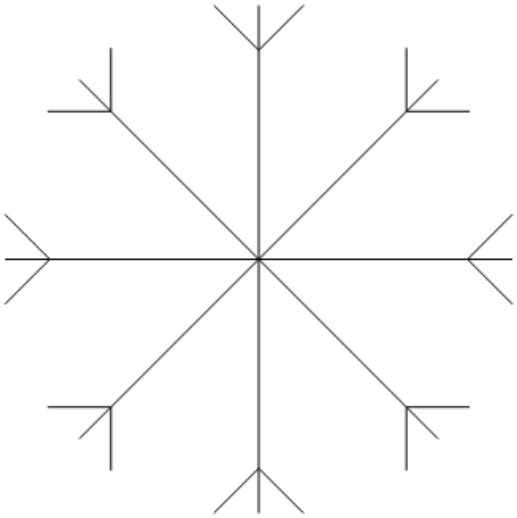


FIGURE 2.32 Le défi flocon

Définition à la ligne 443 du fichier TortuinoDessins.cpp.

2.4.2.8 floconVonKoch()

```
void floconVonKoch (
            int nbNiveaux,
            float taille )
```

Trace un `flocon de Von Koch` paramétré par son niveau et la taille du segment de départ.

C'est en fait une répétition de la courbeVonKoch(int nbNiveaux, float taille) : six fois séparées par un angle intérieur de 120 degrés.

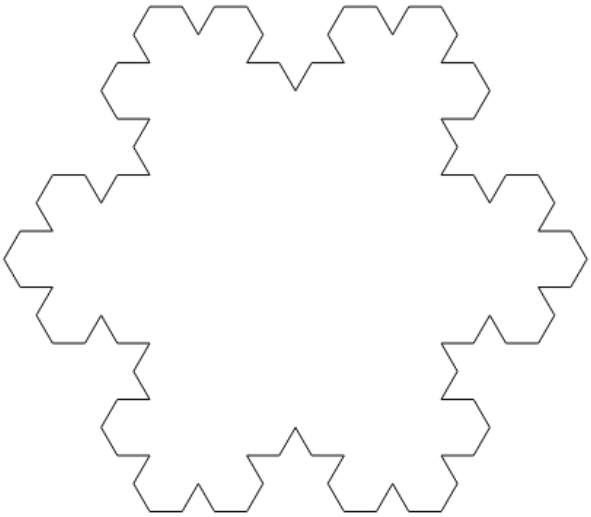


FIGURE 2.33 Un flocon de Von Koch à trois niveaux

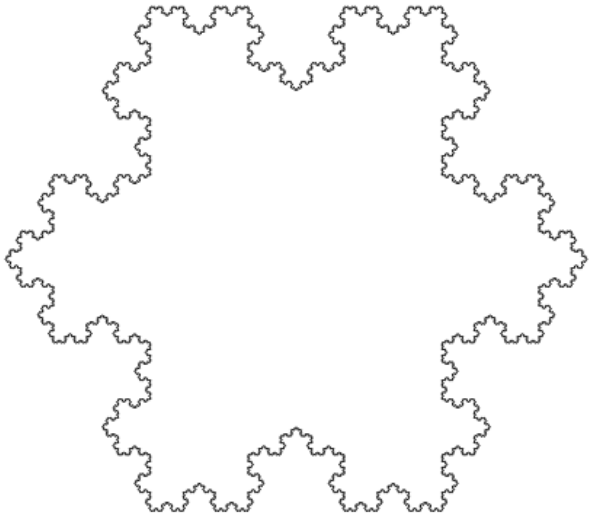


FIGURE 2.34 Un flocon de Von Koch à six niveaux

Paramètres

nbNiveaux	Le nombre de niveaux du flocon de Von Koch, par convention 1 donne un triangle seulement.
taille	La taille du segment à partir de laquelle l'opération récursive est itérée, c'est-à-dire que contrairement à un arbre ou un polygone tels qu'implémentés ici, ajouter plus de niveaux ou de côtés n'augmente pas la taille globale du flocon ; autrement dit, le segment de départ sert d'étalon pour en déduire à l'avance la taille des côtés engendrés.

Définition à la ligne 275 du fichier TortuinoDessins.cpp.

2.4.2.9 maison()

```
void maison ( )
```

Réalise le premier défi que nous proposons pour l'animation Tortuino : une maison avec son toit et sa porte d'entrée.

Elle n'accepte pas de paramètres en entrée, car sa forme est fixée et que mettre à disposition des moyens de personnaliser chaque partie de la maison est long et en soi pas particulièrement intéressant. Le dessin est ainsi toujours le même.

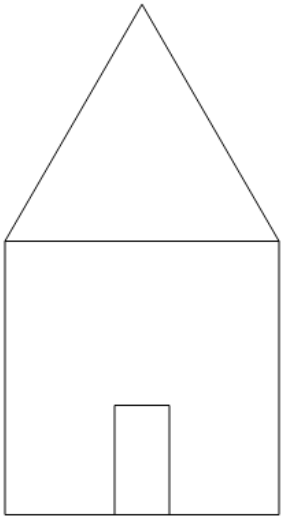


FIGURE 2.35 Le défi maison

Définition à la ligne 337 du fichier TortuinoDessins.cpp.

2.4.2.10 polygoneRegulier()

```
void polygoneRegulier (
            int nbCotes,
            float tailleCote )
```

Fait tracer au robot un polygone régulier en fonction du nombre de côtés souhaités et de la taille de chacun de ces côtés.

Voir l'article Wikipédia suivant pour plus de détails sur cette figure géométrique. Un nombre de côtés de 3 donne un triangle, 4 un carré, 5 un pentagone régulier, ...

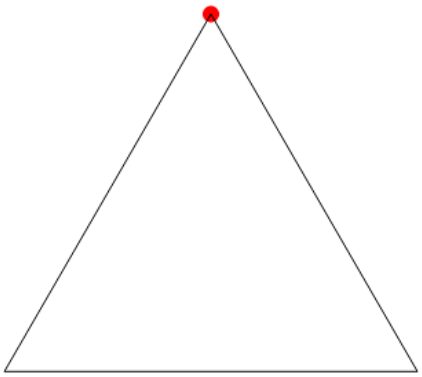


FIGURE 2.36 Un triangle

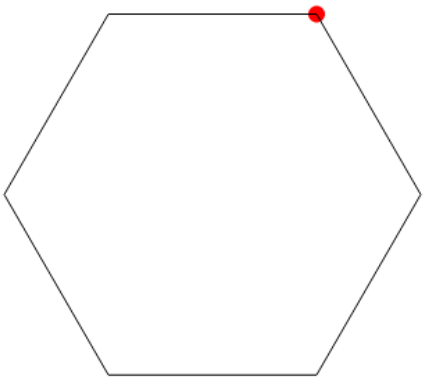


FIGURE 2.37 Un hexagone

Paramètres

nbCotes	Le nombre de côtés du polygone à tracer.
tailleCote	La taille de chacun des côtés.

Définition à la ligne 47 du fichier TortuinoDessins.cpp.

2.4.2.11 sapin()

```
void sapin (
            int nbNiveaux,
            float tailleTronc )
```

Utilise deux arbres asymétriques pour tracer un sapin, c'est-à-dire un arbre où chaque branche se sépare en trois autres : une continuant vers le haut (le tronc donc) et deux horizontales sur le côté (les branches donc).

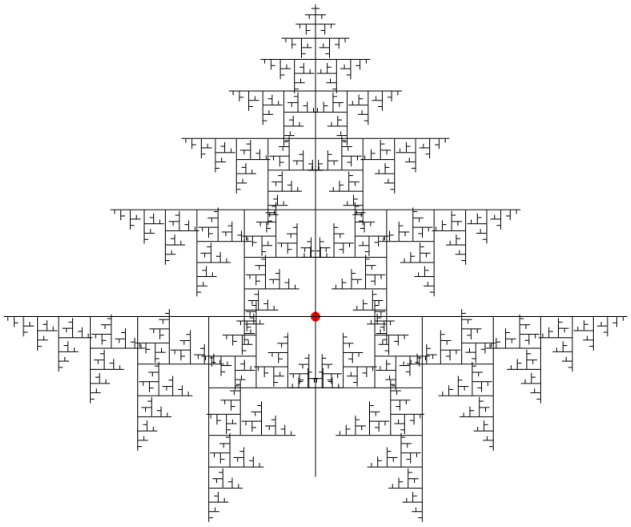


FIGURE 2.38 Un sapin à dix niveaux

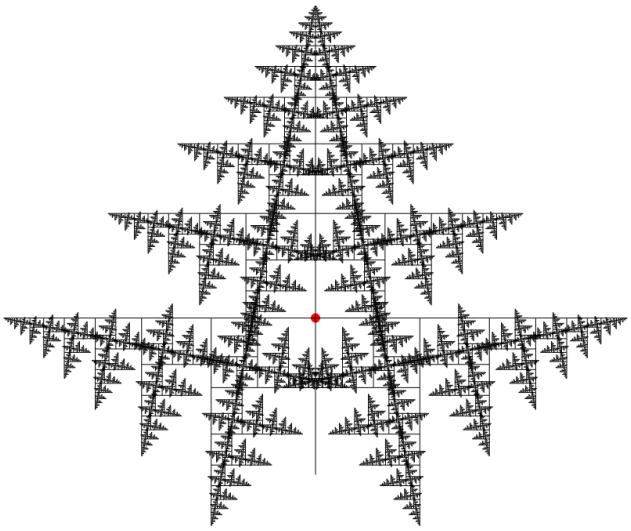


FIGURE 2.39 Un sapin à quinze niveaux

Paramètres

nbNiveaux	Le nombre de niveaux que l'arbre comprendra, c'est-à-dire le nombre de fois moins un que l'arbre va se séparer ou autrement dit la distance en nombre de branches entre la racine et chaque feuille.
tailleTronc	La taille du tronc de départ. Les branches qui en partiront auront leurs tailles d'un tier plus petites.

Voir également

arbre(int nbNiveaux, float tailleTronc)

Définition à la ligne 214 du fichier TortuinoDessins.cpp.

2.4.2.12 spiraleCarree()

```
void spiraleCarree (
            int nbCotes,
            float longueurDepart,
            float ecart )
```

Trace une spirale carrée en fonction du nombre de côtés souhaités, de la longueur du côté de départ et de l'écart en longueur entre deux côtés adjacents.

C'est l'implémentation du deuxième défi Tortuino. Une spirale carrée est une spirale composés d'un certain nombre de segments reliés l'un à la suite de l'autre et séparés d'un angle de droit.

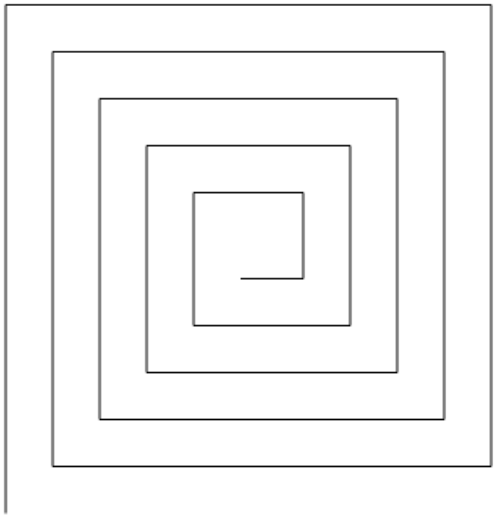


FIGURE 2.40 Une spirale carrée à vingt côtés avec une longueur de départ de 2 et un écart de 3

Paramètres

nbCotes	Le nombre de côtés de la spirale à tracer. 1 fait un seul segment.
longueurDepart	La longueur en centimètres du premier côté de la spirale.
ecart	L'écart en centimètres entre deux côtés adjacents de la spirale, c'est-à-dire entre les côtés qui se trouvent dans la même direction à partir du point de départ - au nord, au sud, à l'est ou à l'ouest. Ce n'est pas directement l'écart entre la longueur du côté en cours et son suivant, mais plutôt son quadruple.

Définition à la ligne 380 du fichier TortuinoDessins.cpp.

2.4.2.13 tangram()

```
void tangram ( )
```

Dessine le troisième défi de l'animation : un carré divisé en pièces à tangram, nom raccourci en juste tangram.

Le carré en question est composé de cinq triangles, un plus petit carré et un parallélogramme. De même que la maison(), la geométrie est fixée et n'accepte donc pas de paramètres permettant d'ajuster sa forme.

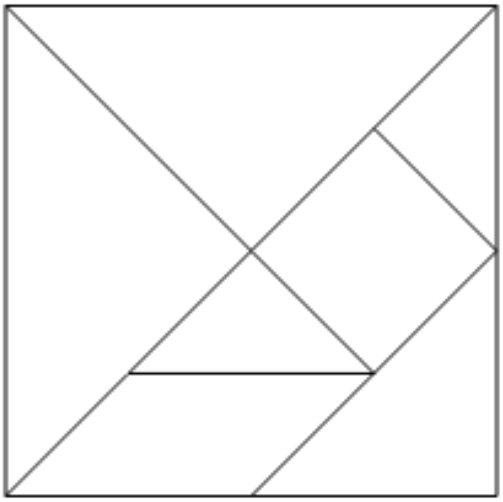


FIGURE 2.41 Le défi tangram

Définition à la ligne 400 du fichier TortuinoDessins.cpp.

2.4.2.14 triangle()

```
void triangle (
            float tailleCote )
```

Trace un triangle equilatéral d'une certaine taille.

En réalité, ce n'est qu'une adaptation de polygoneRegulier(int nbCotes, float tailleCote) au cas particulier du triangle equilatéral.

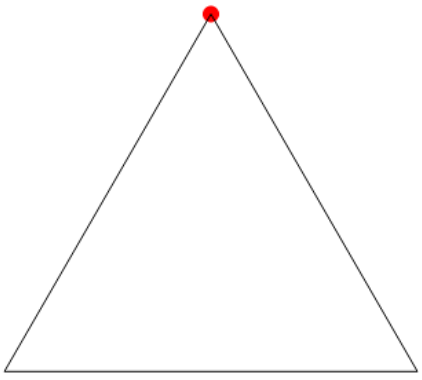


FIGURE 2.42 Un triangle

Paramètres

tailleCote	La taille des côtés du triangle.

Définition à la ligne 64 du fichier TortuinoDessins.cpp.

2.4.2.15 triangleSierpinski()

```
void triangleSierpinski (
            int nbNiveaux,
            float taille )
```

Trace un triangle de Sierpiński paramétré par son niveau et la taille globale du triangle.

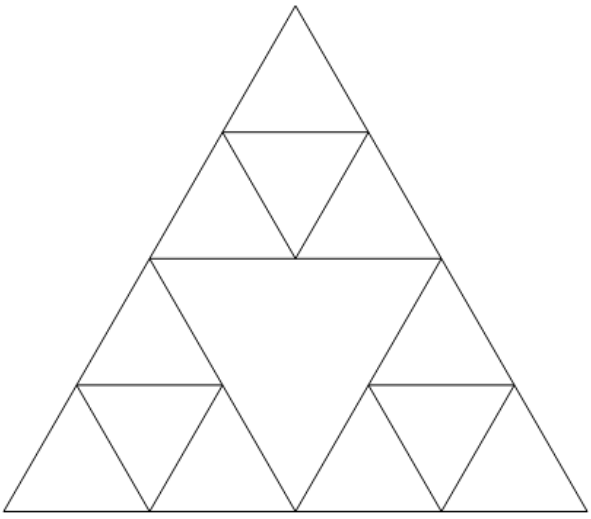


FIGURE 2.43 Un triangle de Sierpinski à trois niveaux

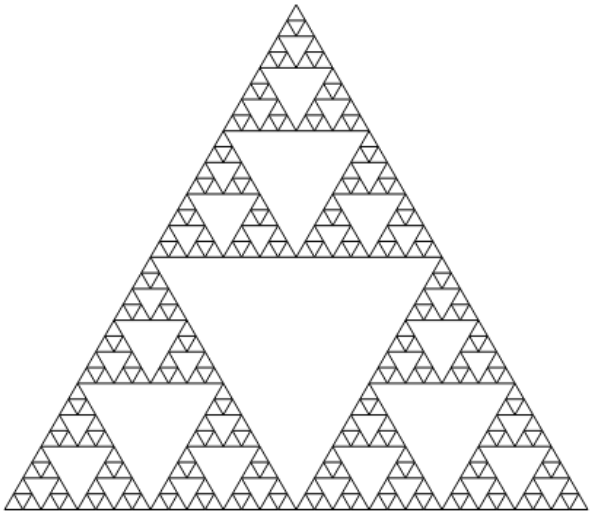


FIGURE 2.44 Un triangle de Sierpinski à six niveaux

Paramètres

nbNiveaux	Le nombre de niveaux du triangle de Sierpiński.
taille	La taille du segment de départ qui sera conservée au fur et à mesure des itérations de l'algorithme de Sierpiński ; idem à ce que fait floconVonKoch(int nbNiveaux, float taille)

Définition à la ligne 300 du fichier TortuinoDessins.cpp.

Index