INF600E - Travail de session

Maëliss Teissier - TEIM01568708

Groupe 20

Session Hiver 2022 - UQAM

Contents

1 Specifications du language maedraw

Le langage maedraw est un langage déclaratif de dessin, typé statiquement et interprété, permettant de déclarer et déplacer des mines de crayons sur un canevas (une matrice de pixels). Le langage permet de changer leur couleur et leur taille de trait.

1.1 Typage et primitives

Le langage maedraw est un langage typé statiquement. La déclaration d'une variable doit être précédée de son type définitif. Les classes ne sont pas implémentées dans ce langage, car il semblait superflu d'offrir la création de nouveaux types étant donné l'objectif du langage. En revanche, un certain nombre de primitives ont été créées pour les besoins du langage :

1.1.1 Type int :

Type entier signé, peut prendre toutes les valeurs entre -2147483648 et 2147483647 (32 bits signé).

```
// Exemple de déclaration d'une variable de type int :
int index = 0;
```

1.1.2 Type bool :

Type classique booleen, peut prendre la valeur ‘true‘ ou la valeur ‘false‘.

```
// Exemple de déclaration d'une variable de type bool :
bool isEqual = true;
```

1.1.3 Type color :

Couleur donnée à la mine de crayon. Le type color peut prendre les valeurs suivantes : red, yellow, blue, black, white et rgb[int, int, int] (chaque int devant être entre 0 et 255).

```
// Exemples de déclarations de variables de type color :
color pen_color = red;
color salmon = rgb[250,128,114];
```

1.1.4 Type pos :

Position {x, y} sur le canevas, utilisée pour placer la mine de crayon (déclaration) ou déplacer la mine de crayon (action 'MOVE_POS' décrite plus tard). Prend la valeur '{int,int}'

```
// Exemples de déclarations de variables de type pos :
pos initial_position = {200,200};
```

1.1.5 Type tip_size :

Grosseur de la mine du crayon. Elle prend les valeurs small, med ou large.

```
// Exemples de déclarations de variables de type color :
tip_size small_size = small;
tip_size med_size = med;
tip_size large_size = large;
```

1.1.6 Type tip :

La mine de crayon. Elle prend la valeur pencil qui nécessite une couleur, une taille et une position à sa déclaration.

```
// Exemples de déclarations de variables de type tip :

// avec instanciation préalable des valeurs
color salmon = rgb[250,128,114];
tip_size small_size = small;
pos initial_position = {200,200};
tip my_small_tip = pencil[salmon, small_size, initial_position];

// avec les valeurs non instanciées préalablement
tip new_tip = pencil[rgb[240,230,140], med, {300, 200}];
```

Note : pour cette version du langage, seulement la valeur 'pencil' de type 'tip' a été implémentée. La mine 'pencil' a une forme d'ellipse comme un crayon. Mais le code permet d'étendre le type 'tip' grâce à l'héritage et le type pourrait alors prendre d'autres valeurs comme, par exemple, la valeur 'marker' et avoir une forme rectangle ...

1.2 Opérations

Le langage offre différentes opérations sur certains types :

1.2.1 Opérations arithmétiques

Sur les entiers (int) : Les opérations arithmétiques offertes sur les entiers sont l'addition, la soustraction, la multiplication, la division entière et le modulo.

Sur les couleurs (color) et entiers (int) : Les opérations d'addition et de soustractions sont offertes sur les couleurs et entiers. La syntaxe de l'arithmétique couleur/entiers requiert que la couleur soit l'expression de gauche et l'entier l'expression de droite (ex : bleu + 4 et non 4 + bleu); Additionner un int à une couleur permet d'éclaircir une couleur (selon les échelles d'éclaircissement offertes par l'api de couleurs java). Soustraire un int à une couleur permet de foncer une couleur (selon les échelles d'obscurcissement offertes par l'api de couleurs java).

```
// Exemples d'addition color/int
color dodger_blue = rgb[30,144,255];
tip large_pen = pencil[dodger_blue, large, {0,500}];
int i = 0;
// On éclaircit la couleur 7 fois
while (i < 7){
 large_pen 100 MOVE_ANGLE 0 DEG;

 // addition color/int
 dodger_blue = dodger_blue + 1;
 large_pen COLOR dodger_blue;
 i = i + 1;
}
```

Figure 1: Affichage de l'exécution du code d'addition color/int

```
// Exemples de soustraction color/int
color aqua_marine = rgb[127,255,212];
tip large_pen2 = pencil[aqua_marine, large, {0,600}];
i = 0;
// On fonce la couleur 7 fois
while (i < 7){
 large_pen2 100 MOVE_ANGLE 0 DEG;

 // soustraction color/int
```

```
 aqua_marine = aqua_marine - 1;
 large_pen2 COLOR aqua_marine;
 i = i + 1;
}
```

Figure 2: Affichage de l'exécution du code de soustraction color/int

Sur les couleurs (color) : L'opération d'addition est offerte sur deux couleurs. Additionner deux couleurs ensemble agit comme mélanger deux couleurs. En revanche le mélange de couleurs se fait selon les lois de calculs de couleur RGB et le comportement d'un mélange RGB n'est pas le même qu'un mélange de peinture dans la réalité. En effet, en RGB par exemple, jaune mélangé à bleu donne gris et non vert.

```
// Exemples d'addition de color/color
color sea_green = rgb[46,139,87];
color cyan = rgb[0,255,255];

// affiche un peu de sea_green, un peu de cyan puis un peu de sea_green + cyan
tip large_pen3 = pencil[sea_green, large, {0,700}];
large_pen3 100 MOVE_ANGLE 0 DEG;
large_pen3 COLOR cyan;
large_pen3 100 MOVE_ANGLE 0 DEG;
large_pen3 COLOR cyan + sea_green;
large_pen3 100 MOVE_ANGLE 0 DEG;

// affiche un peu de red, un peu de yellow puis un peu de red + yellow
tip large_pen4 = pencil[red, large, {0,800}];
large_pen4 100 MOVE_ANGLE 0 DEG;
large_pen4 COLOR yellow;
large_pen4 100 MOVE_ANGLE 0 DEG;
large_pen4 COLOR yellow + red;
large_pen4 100 MOVE_ANGLE 0 DEG;
```

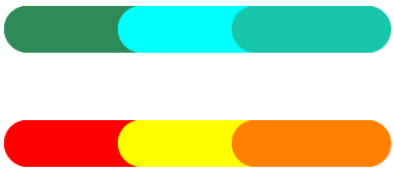

Figure 3: Affichage de l'exécution du code d'additions de color/color

1.2.2 Opérations booléennes

Les opérations booléennes == (equal), ! (not), < (lower than) et > (greater than) sont implémentées sur les entiers.

1.3 Actions

Afin de contrôler le crayon, plusieurs actions sont possibles :

1.3.1 Notion ON et OFF du crayon

les actions ON et OFF permettent au crayon de se mettre en mode "traçage" (comme posé sur la feuille) ou en mode "non-traçage" (comme relevé de la feuille). Ainsi, si la mine est à OFF, les actions de déplacement du crayon se feront sans impact sur le canevas. Si la mine est à ON, la mine tracera un trait en se déplaçant.

Lors de sa déclaration, une mine est automatiquement sur ON. Tant qu'elle n'est pas mise sur OFF elle restera sur ON et inversement.

Syntaxe :

identifiant	ON | OFF
tip	mot réservé

```
// Exemple d'utilisation de ON et OFF.

// Les variables de type tip sont toujours sur ON lors de leur déclaration.
tip small_pen = pencil[SPRING_GREEN, small, {500,500}];

// La ligne suivante va donc tracer un trait horizontal de 300 pixels
// vers la gauche en partant de la position d'initialisation {500,500}
small_pen 300 MOVE_ANGLE 0 DEG;

// On "relève" la mine pour la déplacer sans tracer
small_pen OFF;
// La mine revient a sa position intiale
small_pen MOVE_POS {500,500};

// On "pose" la mine
small_pen ON;

// On trace un trait vertical de 300 pixels vers le haut
small_pen 300 MOVE_ANGLE 90 DEG;
```

1.3.2 Changer la couleur d'une mine : COLOR

L'action COLOR permet de changer la couleur d'une mine.

Syntaxe :

identifiant	COLOR	identifiant
tip	mot réservé	color

```
// Exemples d'utilisation de l'action COLOR

// on declare une tip bleu
tip large_pen5 = pencil[blue, large, {0,850}];
// le trait tracé sera bleu
large_pen5 100 MOVE_ANGLE 0 DEG;
// on change la couleur de la tip pour rouge
large_pen5 COLOR red;
// le trait tracé sera maintenant rouge
large_pen5 100 MOVE_ANGLE 0 DEG;
```

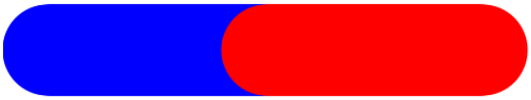

Figure 4: Affichage de l'exécution du code de l'action COLOR

1.3.3 Déplacement dans une direction : MOVE_ANGLE

L'action MOVE_ANGLE permet de déplacer la pointe du crayon d'un nombre de pas (pixels) dans la direction d'un angle du cercle trigonométrique en considérant que la position actuelle du crayon est le centre du cercle. Le crayon peut donc se diriger dans toutes les directions de 0 à 360 degrés.

Exemple dans la figure ci-dessous dessinée avec le langage maedraw (dont le code est fourni en annexe), le crayon commence à la position {500,500} puis fait 300 pas dans la direction d'angle 0 degré. Le crayon revient sur la position {500,500} puis fait 300 pas dans la direction d'angle 30 degrés et ainsi de suite pour les angles 45, 60, 90, 120,135, 150, 180, 210, 235, 240, 270, 300, 315 et 330 degrés.

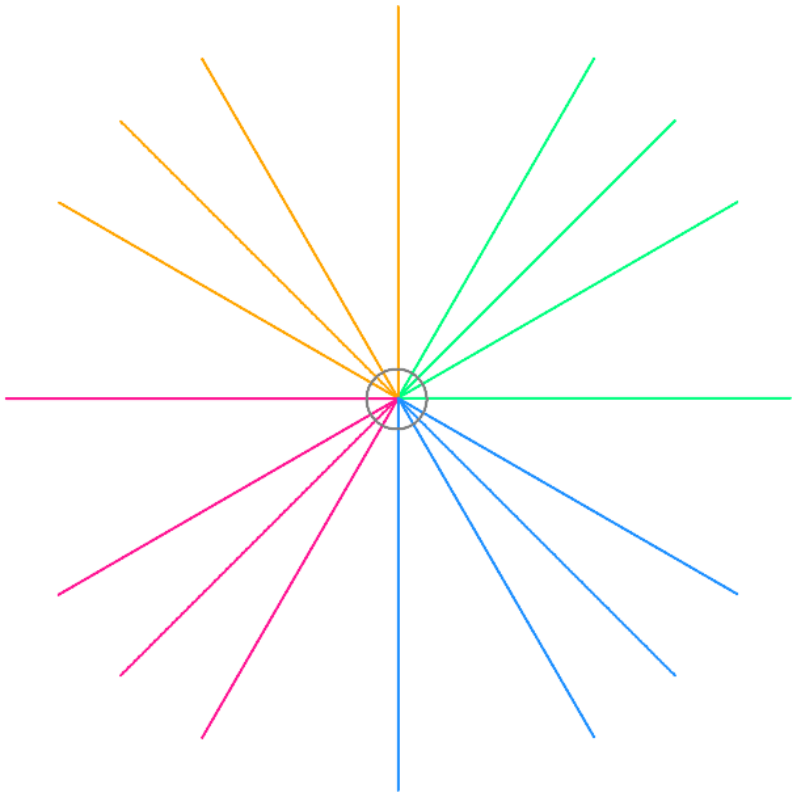

Figure 5: Cercle Trigonometrique dessiné avec maedraw

Syntaxe :

```
my_pen 300 MOVE_ANGLE 90 DEG;
```

identifiant	nombre de pas	MOVE_ANGLE	degré angle directionnel	DEG
tip	int	mot reservé	int	mot réservé

Ainsi, les angles de degrés 0, 90, 180 et 270 sont utilisés respectivement pour les directions $\rightarrow$, $\leftarrow$, $\uparrow$ et $\downarrow$:

```
// Exemple d'utilisation de l'action MOVE_ANGLE. Dessine un carré
tip med_pen = pencil[blue, med, {500,500}];

// 300 pas dans la direction DROITE
small_pen 300 MOVE_ANGLE 0 DEG;

// 300 pas dans la direction HAUT
small_pen 300 MOVE_ANGLE 90 DEG;

// 300 pas dans la direction GAUCHE
small_pen 300 MOVE_ANGLE 180 DEG;

// 300 pas dans la direction BAS
small_pen 300 MOVE_ANGLE 270 DEG;
```

1.3.4 Déplacement vers une nouvelle position : MOVE_POS

L'action MOVE_POS permet de déplacer la pointe du crayon de sa position actuelle jusqu'à la position donnée en argument à l'action.

Syntaxe :

identifiant	MOVE_POS	position ou la mine doit se rendre
tip	mot reservé	pos

```
// (exemple complet dans le fichier move_pos.maedraw)
// Exemple d'utilisation de l'action MOVE_POS
color my_purple = MEDIUM_VIOLET_RED;
tip pen = pencil[my_purple, med, {500,200}];

// On déplace la tip a la position {500,500},
// elle est sur ON donc le chemin est tracé
pen MOVE_POS {500, 500};
// Eclaircissement de la couleur
my_purple = my_purple + 1;
pen COLOR my_purple;

// Autre déplacement de la mine du crayon toujours sur ON
pen MOVE_POS {800, 500};
// Eclaircissement de la couleur
my_purple = my_purple + 1;
pen COLOR my_purple;

...
```

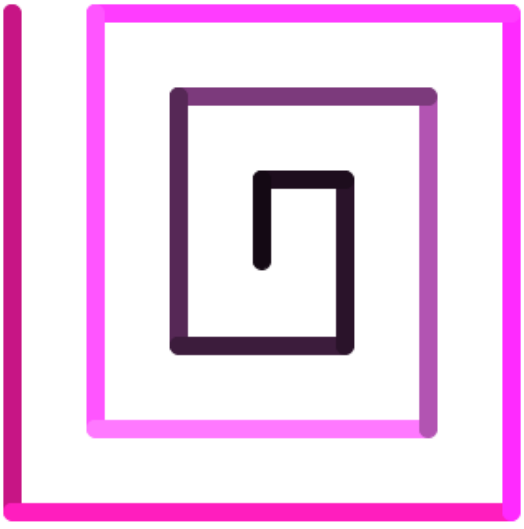

Figure 6: Affichage de l'exécution du code de l'action MOVE_POS

1.4 Fonctions et Procédures

Le langage implémente les fonctions et procédures. Dans le programme, elles doivent être déclarées entre l'entête et les instructions. Dans une librairie, elles doivent être déclarées entre l'entête et les déclarations de variables globales. Les fonctions sont déclarées de la sorte :

```
fonction nomDeFonction(type nomDeParametre1, type nomDeParametre2):Type_de_retour {
// corps de la fonction
    retour expression;
}

// exemple d'une vrai fonction
function calculateNewCoordinatesPeeks(int: x, int: y ):pos{
        return {x, y * -1}
    }
}
```

Les procédures sont déclarées de la même manière que les fonctions, mais ne doivent pas posséder de retour d'expression

```
// exemple d'une vrai procédure implémentée dans libshapes.maedraw
procedure drawQuarterCircleUpCounterCW(pen: tip, c:color, size:int){
    int angle = 90;
    pen COLOR c;
    pen ON ;
    while (angle < 180){
    pen size MOVE_ANGLE angle DEG ;
    angle = angle + 5;
    }
}
```

1.5 Instructions

Instructions implémentées dans le language :

1.5.1 if/else

Classique if / else implémenté avec des accolades autour des blocs d'instructions. Le else est optionnel. Exemple :

```
if expression_booleenne {
// instructions
} else {
// instructions
}
```

1.5.2 while

Classique while

```
while expression_booleenne {
// instructions
}
```

les instructions if et while associées aux opérations de modulo permettent par exemple de dessiner des pointillés

```
tip med_pen = pencil[blue, med, {0,0}];

int i = 0;
while i < 80 {
    if (i % 2 == 0){
        med_pen 1 MOVE_ANGLE 315 DEG;
    } else {
        med_pen OFF;
        med_pen 20 MOVE_ANGLE 315 DEG;
        med_pen ON;
    }
    i = i + 1;
}
```

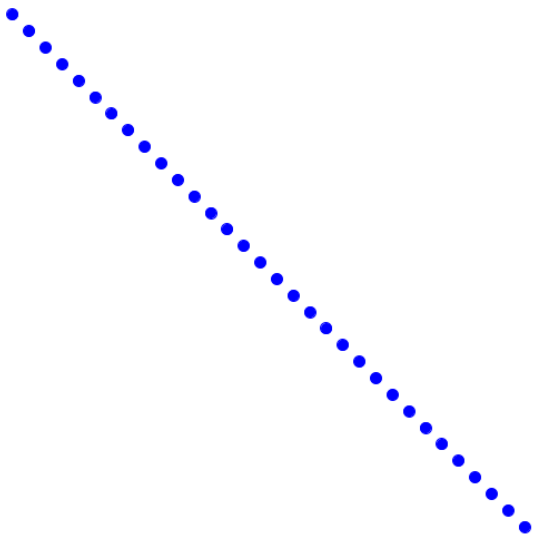

Figure 7: Affichage de l'exécution du code de ligne poitillée

1.6 Librairies et programme principal

Le langage implémente un système de librairies et de programme principal. Il est important de noter que les noms des fichiers de code en langage maedraw doivent terminer par l'extension '.maedraw'.

1.6.1 Librairies

Les imports de librairies : Une librairie peut être importée, soit dans le programme principal, soit dans une autre librairie. Les imports sont les premiers éléments du code attendus. Pour importer une librairie, on utilise le mot réservé 'import' ainsi que le chemin vers la librairie (sans l'extension '.maedraw') suivie d'un point virgule. Le chemin doit etre relatif au dossier courant du processus de l'interpréteur Exemple :

```
import VaryingTests/libraries/lib_color_palette;
import VaryingTests/libraries/lib_shapes;
```

Code d'une librairie Le terme réservé __LIB__ attendu juste après les imports, permet de déclarer que le code du fichier sera une librairie. Pour le programme principal, on utilisera __MAIN__

Les librairies peuvent seulement contenir des déclarations de fonctions et procédures ainsi que des déclarations de variables globales.

Le langage n'implémente pas encore d'espace de nommage (namespace). Il faut donc faire attention aux noms des fonctions/procédures et noms des variables globales. Le langage n'accepte pas de doublon et lancera une erreur de sémantique si une fonction, procédure ou variable globale de même nom est déclarée deux fois dans les librairies importées ou le programme principal (l'erreur indiquera dans quelle librairie l'erreur apparaît). Une vérification est également faite sur les imports de librairies pour qu'on n'importe pas deux librairies du même nom.

1.7 Programme Principal

Après l'import des librairies, le programme principal attend le terme réservé __MAIN__ qui différencie une librairie du programme principal.

1.7.1 Canevas

Le langage requiert ensuite de déclarer la taille en pixel du canevas de dessin avec la syntaxe suivante :

CANVAS	largeur(px)	,	hauteur(px)
mot réservé	int	,	int

Exemple d'entête d'un fichier de code de programme principal :

```
import VaryingTests/libraries/lib_color_palette;
import VaryingTests/libraries/lib_shapes;

__MAIN__
CANVAS 2000,1000;

//...
```

1.8 Commentaires

Les commentaires de lignes sont implémentés :

```
// exemple de commentaire
```

2 Choix de conception et limitations

J'ai voulu rendre visuellement le concept du crayon qui se déplace sur un canevas. Je voulais donc pouvoir rendre les déplacements du crayon à l'écran avec un concept de temps. J'ai utilisé les librairies de java swing et awt afin d'utiliser les interfaces graphiques proposées par java. Malheureusement, ces outils peuvent dessiner des formes à l'écran, mais ne permettent pas de dessiner un nouvel élément à l'écran tout en conservant ceux déjà dessinés. J'ai donc eu besoin d'un concept d'une structure de données qui conserve chaque position du crayon et qui redessine le canevas complet à chaque déplacement. Ce n'est pas optimal et quand le dessin commence à devenir trop gros, on peut observer une latence de plus en plus grande dans l'affichage du dessin. Je suis tout de même satisfaite du rendu et si j'avais eu plus de temps j'aurais surement continué à étendre et améliorer cet outil.

3 Pour la suite

Il serait intéressant d'ajouter les doubles, les opérations sinus et cosinus pour faire des calculs d'angles, une exportation du résultat final en PNG, une option qui dessine uniquement le résultat final au lieu de jouer le tracé de crayon (bien que cette option soit fascinante pour les enfants, il serait plus pratique pour le développement d'avoir un affichage immédiat plutôt que d'attendre que le dessin complet soit tracé).

4 Compilation et interprétation du projet

Un fichier README.md est fourni et décrit toutes les étapes en détail pour la compilation et l'exécution de l'interpréteur.

5 Exemple plus avancé d'utilisation du language

Code disponible dans le dossier de VaryingTests. Nom du fichier : big_drawing.maedraw

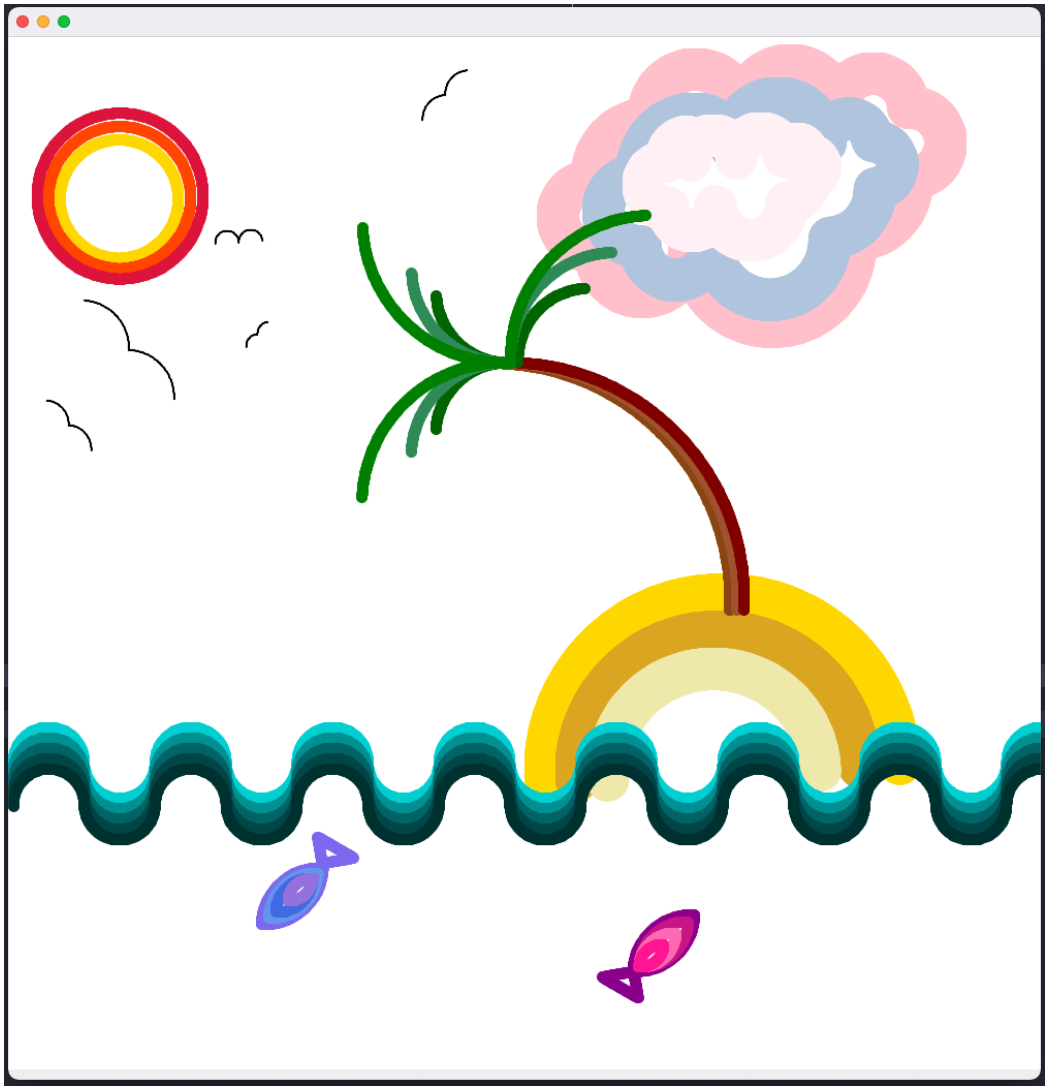

Figure 8: Dessin réalisé avec le language maedraw