

# Rapport de Projet : Draw++

Chekhab Elias  
Guihot Nathan  
Aissaoui Ahmed  
Zheng Lise  
Benmansour Firas

6 janvier 2025

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>La Syntaxe du Langage</b>	<b>4</b>
2.1	Grammaire BNF du Langage Draw++ . . . . .	4
2.2	Variables Obligatoires . . . . .	5
2.3	elements de Base . . . . .	6
2.3.1	Terminaison des Instructions . . . . .	6
2.3.2	Commentaires . . . . .	6
2.3.3	Delimitateurs . . . . .	6
2.4	Types et Variables . . . . .	6
2.4.1	Declaration des Variables . . . . .	6
2.5	Structures de Contrôle . . . . .	7
2.5.1	Conditions . . . . .	7
2.5.2	Boucles . . . . .	8
2.6	Manipulation du Curseur . . . . .	10
2.7	Dessiner des Formes . . . . .	11
2.8	Couleurs et Styles . . . . .	11
<b>3</b>	<b>Description du traducteur</b>	<b>12</b>
3.1	Architecture du traducteur . . . . .	12
3.2	Processus de traduction . . . . .	13
3.2.1	Code Draw++ . . . . .	13
3.2.2	Code C genere . . . . .	13
<b>4</b>	<b>Bibliothèque graphique Draw++ (DPP)</b>	<b>14</b>
4.1	Introduction à Draw++ . . . . .	14
4.2	Fonctionnalites principales de Draw++ . . . . .	14
4.2.1	Gestion des Curseurs . . . . .	14
4.2.2	Dessin de Formes Geometriques . . . . .	15
4.2.3	Gestion des Couleurs . . . . .	15
4.3	Avantages de Draw++ . . . . .	16

4.4	Conclusion . . . . .	16
<b>5</b>	<b>Gestion des erreurs</b>	<b>17</b>
5.1	Types d'erreurs et traitements associes . . . . .	17
5.2	Exemple d'integration . . . . .	18
<b>6</b>	<b>Interface de Developpement Integre (IDE)</b>	<b>19</b>
6.1	Vue d'ensemble . . . . .	19
6.2	Composants principaux . . . . .	19
6.2.1	Barre de menu . . . . .	19
6.2.2	editeur de code . . . . .	20
6.2.3	Zone de previsualisation . . . . .	20
6.2.4	Terminal integre . . . . .	21
6.3	Fonctionnalites d'analyse et debogage . . . . .	21
6.3.1	Analyse lexicale . . . . .	21
6.3.2	Retours d'erreur . . . . .	21
6.3.3	Verification en temps reel . . . . .	22
6.4	Bonnes pratiques d'utilisation . . . . .	22
6.4.1	Initialisation du projet . . . . .	22
6.4.2	Debogage . . . . .	22
<b>7</b>	<b>Exemples d'execution</b>	<b>23</b>
7.1	Exemple 1 : Dessin d'un cercle . . . . .	24
7.2	Exemple 2 : Dessin d'un triangle . . . . .	25
7.3	Exemple 3 : Dessin du drapeau de la France . . . . .	26
7.4	Exemple 4 : Dessin d'une ligne . . . . .	27
7.5	Exemple 5 : Dessin d'un rectangle . . . . .	28
7.6	Exemple 6 : Dessin d'une ellipse . . . . .	29
7.7	Sortie du terminal . . . . .	30
<b>8</b>	<b>Conclusion</b>	<b>31</b>

# Chapitre 1

## Introduction

Draw++ est un langage de programmation spécialement conçu pour la création graphique, visant à simplifier l'interaction avec des éléments visuels à travers un ensemble d'instructions simples et intuitives. Ce langage a été conçu pour offrir une approche accessible et directe de la programmation graphique, permettant aux utilisateurs de dessiner des formes.

Le projet inclut non seulement le langage de programmation proprement dit, mais aussi un environnement de développement intégré (IDE) spécialement conçu pour faciliter l'écriture, le test, et le débogage du code. L'IDE permet aux utilisateurs de visualiser immédiatement l'effet de leur code sur des graphiques à l'écran, favorisant une approche interactive et réactive de la programmation. Grâce à des fonctionnalités telles que la coloration des erreurs syntaxiques, le débogage et la gestion des erreurs, l'IDE rend l'expérience de programmation plus intuitive et accessible, même pour les débutants.

# Chapitre 2

## La Syntaxe du Langage

### 2.1 Grammaire BNF du Langage Draw++

Voici la grammaire formelle du langage Draw++ exprimée en notation BNF :

```
<drawpp_program> ::= <statement_list>

<statement_list> ::= <statement> ";" | <statement> ";" <statement_list>

<statement> ::= <variable_declaration>
               | <constant_declaration>
               | <assignment>
               | <compound_assignment>
               | <if_statement>
               | <loop_statement>
               | <cursor_statement>
               | <shape_statement>
               | <comment>

<variable_declaration> ::= "var" <type> <identifier> "=" <expression>

<type> ::= "int" | "float" | "bool" | "color"

<assignment> ::= <identifier> "=" <expression>
<compound_assignment> ::= <identifier> <compound_operator> <expression>
<compound_operator> ::= "+=" | "-=" | "*=" | "/="

<if_statement> ::= "if" "(" <condition> ")" "{" <statement_list> "}"
               [ <elif_chain> ]
               [ "else" "{" <statement_list> "}" ]

<elif_chain> ::= "elif" "(" <condition> ")" "{" <statement_list> "}" [ <elif_chain> ]

<loop_statement> ::= <for_loop> | <while_loop>

<for_loop> ::= "for" "(" [ <variable_declaration> ] ";" <condition> ";" [ <assignment> ] ")"
               "{" <statement_list> "}"

<while_loop> ::= "while" "(" <condition> ")" "{" <statement_list> "}"
```

```

<cursor_statement> ::= <cursor_creation>
                    | <cursor_movement>
                    | <cursor_style>
                    | <cursor_visibility>

<cursor_creation> ::= "cursor" <identifier> "=" "create_cursor" "(" <expression> "," <expression> ")"

<cursor_movement> ::= <identifier> "." ( <move> | <rotate> | <position> )
<move> ::= "move" "(" <expression> ")"
<rotate> ::= "rotate" "(" <expression> ")"
<position> ::= "position" "(" <expression> "," <expression> ")"

<cursor_style> ::= <identifier> "." ( <color> | <thickness> )
<color> ::= "color" "(" <color_value> ")"
<thickness> ::= "thickness" "(" <expression> ")"

<cursor_visibility> ::= <identifier> "." "visible" "("

<shape_statement> ::= <identifier> "." <shape_type>

<shape_type> ::= <draw_line> | <draw_rectangle> | <draw_circle> | <draw_triangle> | <draw_ellipse>

<draw_line> ::= "draw_line" "(" <expression> ")"

<draw_rectangle> ::= "draw_rectangle" "(" <expression> "," <expression> "," <bool_value> ")"

<draw_circle> ::= "draw_circle" "(" <expression> "," <bool_value> ")"

<draw_triangle> ::= "draw_triangle" "(" <expression> "," <expression> "," <bool_value> ")"

<draw_ellipse> ::= "draw_ellipse" "(" <expression> "," <expression> "," <bool_value> ")"

<color_value> ::= <predefined_color> | <custom_color>
<predefined_color> ::= "BLACK" | "WHITE" | "RED" | "GREEN" | "BLUE"
                    | "GRAY" | "LIGHT_GRAY" | "DARK_GRAY"
                    | "ORANGE" | "BROWN" | "PINK" | "CORAL" | "GOLD"
                    | "PURPLE" | "INDIGO" | "TURQUOISE" | "NAVY" | "TEAL"
                    | "FOREST_GREEN" | "SKY_BLUE" | "OLIVE" | "SALMON" | "BEIGE" | "YELLOW"
<custom_color> ::= "rgb" "(" <expression> "," <expression> "," <expression> ")"

<expression> ::= <term> | <term> <operator> <expression>
<term> ::= <identifier> | <number> | <bool_value> | <string_value>

<condition> ::= <expression> <relational_operator> <expression>
            | <bool_value>
            | <identifier>

<operator> ::= "+" | "-" | "*" | "/" | "%"
<relational_operator> ::= "<" | "<=" | ">" | ">=" | "==" | "!="

<bool_value> ::= "true" | "false"

<comment> ::= <single_line_comment> | <multi_line_comment>
<single_line_comment> ::= "//" <text> <newline>
<multi_line_comment> ::= "/*" <text> "*/"

<identifier> ::= [a-zA-Z_][a-zA-Z0-9_]*
<number> ::= [0-9]+ ( "." [0-9]+ )?
<text> ::= [^]*
<newline> ::= "\n" | "\r\n"

```

## 2.2 Variables Obligatoires

Pour garantir un fonctionnement correct, chaque programme Draw++ doit définir les dimensions de la fenêtre graphique au début. Ces dimensions sont spécifiées par deux variables obligatoires :

- `windowWidth` : Définit la largeur de la fenêtre.
- `windowHeight` : Définit la hauteur de la fenêtre.

Voici un exemple de déclaration correcte :

```
1 var int windowHeight = 500;  
2 var int windowHeight = 500;
```

Ces declarations doivent apparaître avant toute autre instruction. Toute omission entraînera une erreur à l'exécution.

## 2.3 elements de Base

### 2.3.1 Terminaison des Instructions

Chaque instruction doit se terminer par un point-virgule (;) afin de séparer les instructions.

```
1 var int x = 10; // Declaration d'une variable  
2 cursor.move(100); // Déplacement du curseur
```

### 2.3.2 Commentaires

Les commentaires sont ignorés à l'exécution et permettent de documenter le code :

- **Commentaires sur une ligne** : Utilisez //.
- **Commentaires multi-lignes** : Utilisez /\* ... \*/.

```
1 // Ceci est un commentaire sur une ligne  
2 /* Ceci est un commentaire  
3    multi-lignes */
```

### 2.3.3 Delimiteurs


Les delimites structurent le code et incluent :

- {} : Pour delimites les blocs d'instructions.
- () : Pour entourer les arguments des fonctions.

## 2.4 Types et Variables

### 2.4.1 Declaration des Variables

Les variables sont déclarées avec le mot-cle `var`, suivi de leur type, de leur nom et de leur valeur initiale.



```
var int compteur = 0;
var float pi = 3.14159;
var color myColor = rgb(255, 0, 0);
var bool isFilled = true;
```

## 2.5 Structures de Contrôle

### 2.5.1 Conditions

Les structures conditionnelles permettent d'exécuter des blocs de code selon des conditions :

- `if (<condition>) { <bloc d'instructions> }`
- `elif (<condition>) { <bloc d'instructions> }`
- `else { <bloc d'instructions> }`



```
var int windowHeight = 800;
var int windowHeight = 600;

var int size = windowHeight / 3;
var color myColor = rgb(5, 10, 231);

var int i = 3;

cursor c1 = create_cursor(size, 0);

if(i == 0) {
    c1.color(YELLOW);
}
elif(i == 1) {
    c1.color(myColor);
}
else {
    c1.color(INDIGO);
};

c1.draw_rectangle(size, windowHeight, true);
```

FIGURE 2.1 – Exemple de code conditionnel dans Draw++, montrant l'utilisation de la structure if-elif-else pour déterminer la couleur d'un rectangle selon une condition.





FIGURE 2.2 – Rendu graphique correspondant à l’exécution du code conditionnel, illustrant le changement de couleur du rectangle.

### 2.5.2 Boucles

Draw++ prend en charge deux types de boucles principales :

— `while (<condition>) { <bloc d’instructions> }`

```
var int windowHeight = 500;
var int windowWidth = 500;

var int size = 20;
var int maxSize = 200;

cursor c = create_cursor((windowWidth / 2) - 100, (windowHeight / 2) - 100);
c.color(GREEN);

while (size <= maxSize) {
    c.draw_rectangle(size, size, false); // Dessine un carré non rempli
    size = size + 20; // Augmente la taille pour le prochain carré
};
```

FIGURE 2.3 – Exemple d’utilisation de la boucle while pour dessiner des carrés imbriqués.

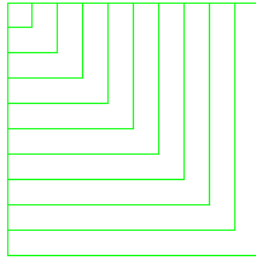


FIGURE 2.4 – Rendu graphique montrant le resultat de la boucle while, illustrant des carres imbriques dans un carre plus grand.

```
— for (<initialisation>; <condition>; <mise à jour>){
    <bloc d'instructions>}
```

```
var int windowHeight = 500;
var int windowWidth = 500;

var int gridSize = 50;

cursor c = create_cursor((windowHeight / 2) - 5 * gridSize, windowWidth / 2);
c.color(RED);

for (var int col = 0; col < 5; col = col + 1) {
    c.move(gridSize);
    c.draw_rectangle(gridSize, gridSize, false); // Dessine une cellule de la grille
};
```

FIGURE 2.5 – Exemple d'utilisation de la boucle for pour dessiner une serie de carres.



FIGURE 2.6 – Rendu graphique illustrant la liste de carres dessines à l'aide de la boucle for.

```

var int windowHeight = 500;
var int windowWidth = 500;

cursor c = create_cursor(windowWidth / 2, windowHeight / 2);
c.color(BLUE);

for (var int i = 0; i < 10; i = i + 1) {
    c.draw_line(50);      // Dessine une marche horizontale
    c.rotate(90);         // Tourne de 90° pour monter
    c.draw_line(50);      // Dessine une marche verticale
    c.rotate(180);        // Retourne à l'horizontale
};

```

FIGURE 2.7 – Exemple d’utilisation de la boucle for pour dessiner une croix, où chaque iteration contribue à former les lignes de la croix

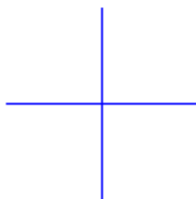


FIGURE 2.8 – Rendu graphique montrant la croix creee par la boucle for, avec deux lignes perpendiculaires dessinees au centre.

## 2.6 Manipulation du Curseur

Le curseur est l’objet central pour dessiner. Il est defini avec la syntaxe suivante :

```
1 cursor <identifiant> = create_cursor(<x>, <y>);
```

Les proprietes et methodes principales du curseur incluent :

- `cursor.move(distance)` : Deplace le curseur.
- `cursor.rotate(angle)` : Tourne le curseur.
- `cursor.color(<couleur>)` : Definit la couleur.
- `cursor.thickness(<valeur>)` : Definit l’épaisseur du trait.
- `cursor.visible()` : Contrôle la visibilite du curseur.

Exemple de creation et utilisation du curseur :

```
1 cursor c = create_cursor(0, 0);  
2 c.color(RED);  
3 c.move(50);
```

## 2.7 Dessiner des Formes

Les formes peuvent être dessinees avec les methodes du curseur :

- `cursor.draw_line(longueur)` : Dessine une ligne.
- `cursor.draw_rectangle(largeur, hauteur, rempli)` : Dessine un rectangle.
- `cursor.draw_circle(rayon, rempli)` : Dessine un cercle.
- `cursor.draw_triangle(base, hauteur, rempli)` : Dessine un triangle.
- `cursor.draw_ellipse(largeur, hauteur, rempli)` : Dessine une ellipse.

Exemples :

```
1 // Dessiner une ligne  
2 cursor c = create_cursor(0, 0);  
3 c.draw_line(100);  
4  
5 // Dessiner un rectangle rempli  
6 c.draw_rectangle(50, 100, true);  
7  
8 // Dessiner un cercle non rempli  
9 c.draw_circle(30, false);
```

## 2.8 Couleurs et Styles

Le curseur peut être personnalise avec des couleurs predefinies ou des couleurs personnalisées. Exemple :

```
1 c.color(RED);  
2 c.thickness(3); // Definit l'epaisseur du trait  
3 c.color(rgb(255, 0, 0)); // Rouge personnalise
```

# Chapitre 3

## Description du traducteur

### 3.1 Architecture du traducteur

Le traducteur de Draw++ est un système qui convertit le code source écrit dans le langage Draw++ en code C intermédiaire, prêt à être compilé. Ce processus de traduction s'effectue en plusieurs étapes clés, chacune ayant pour but de transformer le code dans un format plus proche de l'exécution.

1. **Analyse lexicale** : Dans cette phase, le code source est décomposé en tokens, qui sont les plus petites unités syntaxiques. Par exemple, des mots-clés comme `cursor`, `var`, `if`, ainsi que les identifiants et les opérateurs, sont extraits du code source.
2. **Analyse syntaxique** : Une fois que les tokens ont été générés, le traducteur construit un arbre syntaxique abstrait (AST). Cet arbre représente la structure logique du programme, où chaque nœud correspond à une construction du langage (par exemple, une expression, une condition, ou une affectation).
3. **Analyse sémantique** : À ce stade, le traducteur vérifie la validité des types, la cohérence des variables, et l'intégrité du programme. Cette étape s'assure que les opérations sont effectuées sur des types compatibles (par exemple, éviter de tenter d'ajouter un entier à une chaîne de caractères).
4. **Génération de code** : Enfin, le traducteur génère le code C intermédiaire, qui est une version optimisée et exécutable de l'AST. Ce code est ensuite prêt à être compilé par un compilateur C classique.

## 3.2 Processus de traduction

Prenons un exemple simple pour illustrer le processus de traduction, où un programme écrit en Draw++ est traduit en C intermédiaire.

### 3.2.1 Code Draw++

Le code suivant est écrit en Draw++ pour créer un curseur et dessiner un rectangle (qui peut également être utilisé pour représenter un carré si les côtés sont égaux) :

```
1 cursor c = create_cursor(0, 0);  
2 c.draw_rectangle(50, 50, false);
```

Le programme commence par déclarer un curseur `c` en appelant la fonction `create_cursor` avec les coordonnées (0, 0). Ensuite, la méthode `draw_rectangle` est utilisée pour dessiner un rectangle de 50 unités de largeur et 50 unités de hauteur. Le paramètre `false` indique que le rectangle ne doit pas être rempli.

### 3.2.2 Code C genere

Le traducteur convertit ce code Draw++ en code C, qui pourrait ressembler à ce qui suit :

```
1 // Creation du curseur  
2 Cursor* c = createCursor(0.0, 0.0);  
3  
4 // Dessin du rectangle (non rempli)  
5 drawRectangle(c, 50.0, 50.0, false);
```

Dans cette étape de la traduction :

- La fonction `create_cursor(0, 0)` en Draw++ est traduite en `createCursor(0.0, 0.0)` en C. Les coordonnées sont converties en types `float` pour maintenir la cohérence avec la représentation C des positions en coordonnées flottantes.
- La méthode `c.draw_rectangle(50, 50, false)` devient un appel de fonction `drawRectangle(c, 50.0, 50.0, false)` en C. Le curseur `c` est passé comme un pointeur (`Cursor*`), et les dimensions du rectangle sont converties en `float` pour correspondre à la signature de la fonction générée.

# Chapitre 4

## Bibliothèque graphique Draw++ (DPP)

### 4.1 Introduction à Draw++

Draw++ (DPP) est une bibliothèque graphique conçue pour offrir une interface intuitive et performante pour le dessin graphique. Construit sur la base de SDL2, DPP enrichit les fonctionnalités de cette dernière en intégrant des abstractions et outils adaptés à la manipulation graphique. Cette bibliothèque vise à réduire la complexité de l'utilisation directe de SDL2 tout en permettant une créativité sans limite.

### 4.2 Fonctionnalités principales de Draw++

La bibliothèque Draw++ offre des fonctionnalités avancées pour simplifier et enrichir le processus de création graphique. Voici une présentation détaillée de ses principales capacités.

#### 4.2.1 Gestion des Curseurs

Le concept central de DPP repose sur l'utilisation des curseurs. Ces entités permettent de simplifier le positionnement et la gestion graphique.

- **Création de curseurs** : Créez plusieurs curseurs avec `create_cursor()` pour une manipulation flexible.
- **Opérations intuitives** : Déplacement `move_cursor()`, rotation `rotate_cursor()`, et contrôle de la visibilité.
- **Personnalisation** : Changez la couleur (`set_cursor_color`) et l'épaisseur des traces (`set_cursor_thickness`).

Exemple :

```
1 Cursor* c = create_cursor(100, 100); // Creation d'un
   curseur
2 set_cursor_color(c, red);           // Couleur rouge
3 move_cursor(c, 50);                 // Deplacement de 50
   pixels
4 rotate_cursor(c, 90);               // Rotation de 90
   degres
5 cursor_draw_line(c, 100);           // Dessin d'une ligne
```

## 4.2.2 Dessin de Formes Geometriques

Draw++ permet de dessiner des formes geometriques en simplifiant les calculs complexes.

- **Lignes** : Tracez des lignes avec `cursor_draw_line()`.
- **Rectangles et cercles** : Gerez les dimensions et remplissages avec `cursor_draw_rectangle()` et `cursor_draw_circle()`.
- **Ellipses et triangles** : Creez des formes avancees avec `cursor_draw_ellipse()` et `cursor_draw_triangle()`.

Exemple :

```
1 Cursor* c = create_cursor(200, 150);
2
3 // Dessiner un rectangle rempli
4 cursor_draw_rectangle(c, 200, 100, true);
5
6 // Dessiner un cercle non rempli
7 cursor_draw_circle(c, 50, false);
8
9 // Dessiner une ellipse
10 cursor_draw_ellipse(c, 120, 80, true);
11
12 // Dessiner un triangle
13 cursor_draw_triangle(c, 100, 50, false);
```

## 4.2.3 Gestion des Couleurs

La gestion des couleurs est intuitive avec DPP.

- **Couleurs predefinies** : Utilisez `red`, `green`, `blue`, etc.
- **Couleurs personnalisées** : Creez des couleurs avec `custom_color()`.
- **Attributs dynamiques** : Modifiez facilement les couleurs et styles des formes dessinees.



Exemple :

```
1 // Creer une couleur personnalisee
2 SDL_Color myColor = custom_color(128, 0, 128, 255); //
   Violet opaque
3
4 // Appliquer la couleur au curseur
5 set_cursor_color(c, myColor);
```

## 4.3 Avantages de Draw++

Draw++ se distingue par plusieurs atouts majeurs :

- **Facilite d'utilisation** : Une interface claire pour acceder aux outils graphiques sans complexite.
- **Flexibilite** : Une conception modulaire permettant d'ajouter facilement des fonctionnalites.
- **Support et documentation** : Une documentation complete et des exemples pour guider les developpeurs.

## 4.4 Conclusion

Draw++ offre une solution puissante et intuitive pour la programmation graphique, reduisant la complexite de SDL2 tout en apportant des outils specialises et adaptes à divers projets. Que ce soit pour un usage educatif ou professionnel, DPP simplifie le developpement tout en augmentant la creativite des utilisateurs.

# Chapitre 5

## Gestion des erreurs

La gestion des erreurs dans Draw++ repose sur une architecture modulaire qui couvre l'intégralité du pipeline de compilation : lexicale, syntaxique et sémantique. Chaque étape identifie des erreurs spécifiques et propose des solutions adaptées pour aider l'utilisateur à corriger son code rapidement et efficacement.

### 5.1 Types d'erreurs et traitements associés

- **Erreurs lexicales** : Detectées lors de la conversion du code en tokens.
  - *Caractères invalides* : Présence de symboles non reconnus dans le langage.
    - **Traitement** : Proposer de supprimer ou remplacer les caractères non valides.
  - *Chaînes non terminées* : Oubli d'un guillemet fermant.
    - **Traitement** : Suggester d'ajouter le guillemet manquant (").
- **Erreurs syntaxiques** : Surviennent lors de l'analyse de la structure logique du code.
  - *Jetons inattendus* : Utilisation incorrecte de symboles ou d'instructions (exemple : ) non appariée).
    - **Traitement** : Vérifier les parenthèses, accolades et points-virgules.
  - *Types ou constructions attendus* : Absence d'un type ou mot-clé attendu.
    - **Traitement** : Proposer des exemples valides, comme `var int myVar = 0;`
  - *Terminaison manquante* : Oubli d'un point-virgule à la fin d'une

instruction.

- **Traitement** : Indiquer la ligne concernée et suggerer d'ajouter un ;.
- **Erreurs sémantiques** : Vérifient la validité logique et le sens du code.
  - *Variables non déclarées* : Utilisation d'une variable sans déclaration préalable.
    - **Traitement** : Fournir des exemples de déclaration pour corriger l'erreur.
  - *Conflits de type* : Tentative d'assigner des types incompatibles.
    - **Traitement** : Suggester de convertir les types ou d'utiliser des types compatibles.
  - *Variables déjà déclarées* : Reutilisation d'un identifiant existant.
    - **Traitement** : Proposer un nouveau nom ou la suppression de la déclaration redondante.
  - *Utilisation incorrecte des curseurs* : Méthode appelée sur un objet non initialisé ou non valide.
    - **Traitement** : Fournir un exemple d'initialisation correcte.
- **Erreurs spécifiques aux curseurs** : Ces erreurs sont propres au fonctionnement graphique de Draw++.
  - *Méthode invalide* : Appel d'une méthode inexistante pour un curseur.
    - **Traitement** : Lister les méthodes valides comme `move(distance)` ou `rotate(angle)`.
  - *Propriétés incompatibles* : Essayer d'utiliser une couleur ou une épaisseur invalide.
    - **Traitement** : Fournir des exemples valides, comme `cursor.color(RED);`.

## 5.2 Exemple d'intégration

Un exemple typique de correction automatique :

```
1 var int x = 10
2 cursor c = create_cursor(0, 0);
3 c.move(100);
```

Erreur détectée : *"Expected TokenType.SEMICOLON at line 1"*

**Suggestions :**

- Ajouter un point-virgule après `var int x = 10;`.

# Chapitre 6

## Interface de Developpement Integre (IDE)

### 6.1 Vue d'ensemble

L'IDE Draw++ propose un environnement de developpement complet specialement conçu pour le langage Draw++. L'interface se divise en trois zones principales :

- Un editeur de code avec numerotation des lignes
- Une zone de previsualisation graphique
- Un terminal integre

### 6.2 Composants principaux

#### 6.2.1 Barre de menu

La barre de menu en haut de l'interface offre plusieurs options essentielles :

- **File** : Gestion des fichiers
  - New : Creer un nouveau fichier
  - Open : Ouvrir un fichier existant
  - Save : Sauvegarder le fichier courant
  - Download Image : Exporter le rendu graphique
  - Close Tab : Fermer l'onglet actif
  - Exit : Quitter l'application
- **Run** : Execution du code
- **Help** : Accès à l'aide

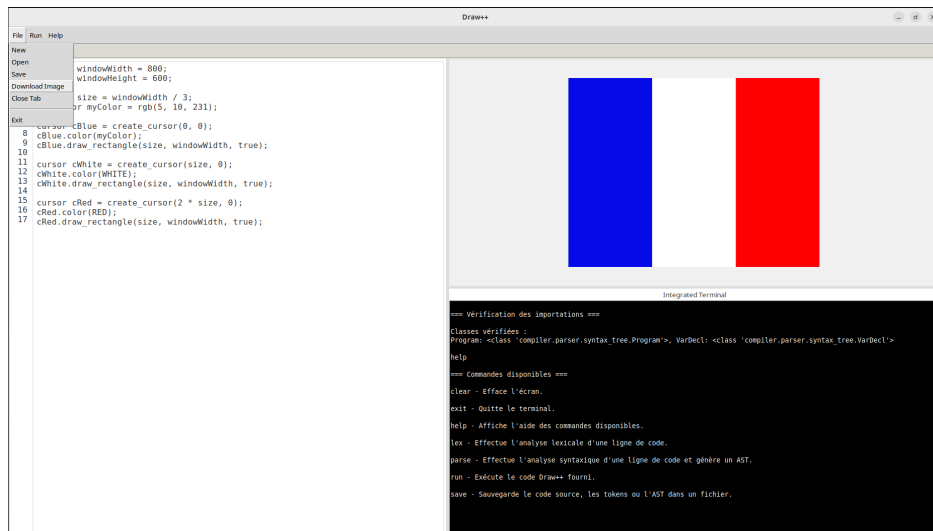


FIGURE 6.1 – Interface de Développement Integre Draw++.

## 6.2.2 editeur de code

L'éditeur intègre plusieurs fonctionnalités avancées :

- Numerotation automatique des lignes
- Coloration syntaxique pour le langage Draw++
- Mise en évidence des erreurs
- Support pour les commentaires (`//` et `/* */`)

## 6.2.3 Zone de previsualisation

La zone de prévisualisation offre les caractéristiques suivantes :

- Affichage du rendu graphique
- Mise à jour lors de l'exécution du code
- Zone redimensionnable
- Aperçu fidèle des dimensions spécifiées par `windowWidth` et `windowHeight`

## 6.2.4 Terminal integre

Le terminal propose plusieurs commandes essentielles :

1	cléar	- Effacé l'écran
2	éxit	- Quitté le terminal
3	hélp	- Affiché l'aide des commandes disponibles
4	lêx	- Efféctué l'analysé lexicalé d'uné ligné dé codé
5	parsé	- Efféctué l'analysé syntaxiqué ét généré l'AST
6	run	- Exécuté le codé Draw++
7	savé	- Sauvégardé le codé sourcé, les tokéns ou l'AST

## 6.3 Fonctionnalites d'analyse et debogage

### 6.3.1 Analyse lexicale

Le terminal affiche les informations detaillées pour chaque token genere :

- Type du token
- Valeur
- Numero de ligne
- Position dans la ligne

Un exemple de sortie d'analyse lexicale :

```
Integrated Terminal
lexErreur : Fournissez une ligne de code.

lex var int i;
|
=== Tokens générés ===
Token(TokenType.VAR, var, line=1, column=1)
Token(TokenType.INT, int, line=1, column=5)
Token(TokenType.IDENTIFIER, i, line=1, column=9)
Token(TokenType.SEMICOLON, ;, line=1, column=10)
Token(TokenType.EOF, None, line=1, column=10)
```

### 6.3.2 Retours d'erreur

L'IDE fournit des messages d'erreur detaillées pour différents types d'erreurs :

- Erreurs de syntaxe
- Erreurs lexicales
- Erreurs sémantiques
- Problèmes de compilation

### 6.3.3 Verification en temps reel

L'IDE effectue plusieurs verifications pendant la saisie :

- Validation des variables obligatoires (`windowWidth`, `windowHeight`)
- Verification de la syntaxe
- Detection des erreurs de type
- Validation des appels de methodes sur les curseurs

## 6.4 Bonnes pratiques d'utilisation

### 6.4.1 Initialisation du projet

Pour demarrer un nouveau projet efficacement :

- Toujours commencer par definir `windowWidth` et `windowHeight`
- Organiser le code avec des commentaires descriptifs
- Sauvegarder regulièrement le travail

### 6.4.2 Debogage

Pour un debogage efficace :

- Utiliser la commande `lex` pour verifier l'analyse lexicale
- Employer la commande `parse` pour valider la structure du code
- Consulter le terminal pour les messages d'erreur detaillées

# Chapitre 7

## Exemples d'exécution

Dans cette section, nous présentons plusieurs exemples d'exécution utilisant le langage Draw++. Chaque exemple inclut le rendu graphique genere et la sortie du terminal.



## 7.1 Exemple 1 : Dessin d'un cercle

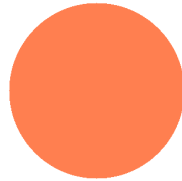


FIGURE 7.1 – Rendu graphique du cercle.

```
var int windowHeight = 800;  
var int windowHeight = 600;  
  
var int posX = windowHeight / 2;  
var int posY = windowHeight / 2;  
  
cursor c1 = create_cursor(posX, posY);  
c1.color(CORAL);  
c1.draw_circle(150,true);
```

FIGURE 7.2 – Code DPP pour dessiner un cercle.

## 7.2 Exemple 2 : Dessin d'un triangle

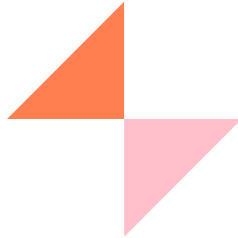


FIGURE 7.3 – Rendu graphique du triangle.

```
var int windowHeight = 800;  
var int windowHeight = 600;  
  
cursor c1 = create_cursor((windowWidth / 2) - 100, (windowHeight / 2) - 100);  
c1.color(PINK);  
c1.draw_triangle(200,200, true);  
  
cursor c2 = create_cursor((windowWidth / 2) - 100, (windowHeight / 2) - 100);  
c2.rotate(180);  
c2.color(CORAL);  
c2.draw_triangle(200,200, true);
```

FIGURE 7.4 – Code DPP pour dessiner un triangle.

### 7.3 Exemple 3 : Dessin du drapeau de la France

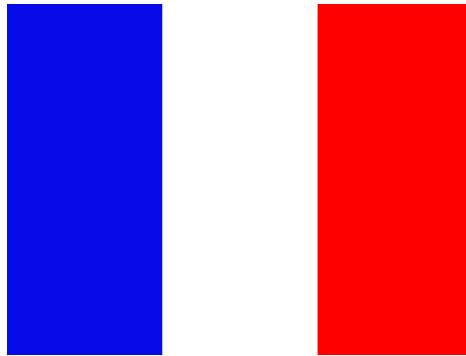


FIGURE 7.5 – Rendu graphique du drapeau de la France.



FIGURE 7.6 – Code DPP pour dessiner le drapeau de la France.

## 7.4 Exemple 4 : Dessin d'une ligne



FIGURE 7.7 – Rendu graphique de la ligne.

```
var int windowHeight = 800;  
var int windowHeight = 600;  
  
cursor c1 = create_cursor(0, 0);  
c1.move(200);  
c1.rotate(90);  
c1.color(NAVY);  
c1.draw_line(windowHeight);
```

FIGURE 7.8 – Code DPP pour dessiner une ligne.

## 7.5 Exemple 5 : Dessin d'un rectangle



FIGURE 7.9 – Rendu graphique du rectangle.

```
var int windowHeight = 800;  
var int windowHeight = 600;  
  
cursor c1 = create_cursor((windowWidth / 2) - 100, (windowHeight / 2) - 100);  
  
c1.color(FOREST_GREEN);  
c1.draw_rectangle(200,200, false);
```

FIGURE 7.10 – Code DPP pour dessiner un rectangle.

## 7.6 Exemple 6 : Dessin d'une ellipse

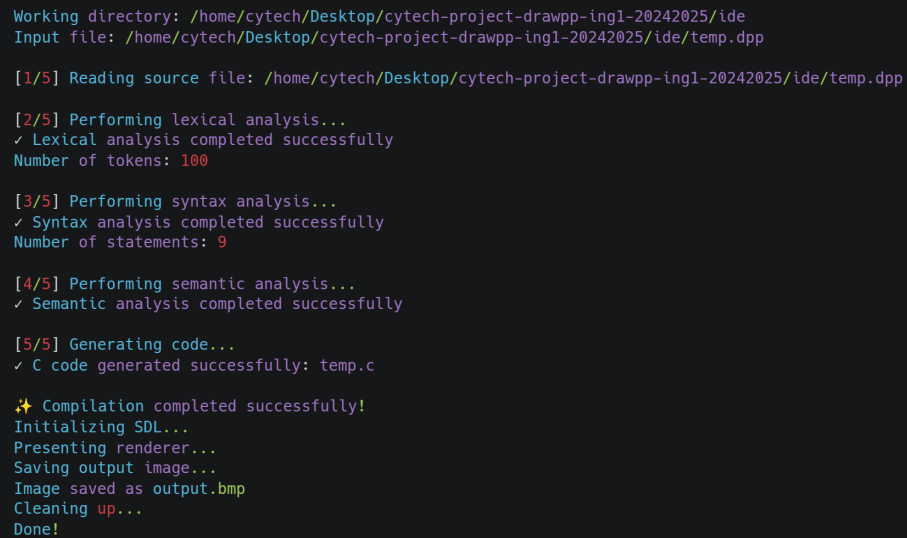


FIGURE 7.11 – Rendu graphique de l'ellipse.

```
var int windowHeight = 800;  
var int windowHeight = 600;  
  
var int posX = windowHeight / 2;  
var int posY = windowHeight / 2;  
  
cursor c1 = create_cursor(posX, posY);  
c1.color(TEAL);  
c1.draw_ellipse(300, 150, true);
```

FIGURE 7.12 – Code DPP pour dessiner une ellipse.

## 7.7 Sortie du terminal



```
Working directory: /home/cytech/Desktop/cytech-project-drawpp-ing1-20242025/ide
Input file: /home/cytech/Desktop/cytech-project-drawpp-ing1-20242025/ide/temp.dpp

[1/5] Reading source file: /home/cytech/Desktop/cytech-project-drawpp-ing1-20242025/ide/temp.dpp

[2/5] Performing lexical analysis...
✓ Lexical analysis completed successfully
Number of tokens: 100

[3/5] Performing syntax analysis...
✓ Syntax analysis completed successfully
Number of statements: 9

[4/5] Performing semantic analysis...
✓ Semantic analysis completed successfully

[5/5] Generating code...
✓ C code generated successfully: temp.c

✨ Compilation completed successfully!
Initializing SDL...
Presenting renderer...
Saving output image...
Image saved as output.bmp
Cleaning up...
Done!
```

# Chapitre 8

## Conclusion

Le projet Draw++ a permis la conception d'un langage graphique minimaliste, couple à un traducteur robuste, capable de transformer des instructions simples en formes visuelles. Le traducteur, conçu avec une attention particulière à la gestion des erreurs et à l'analyse syntaxique, garantit une execution fluide et precise du code. La structure du langage, combinee à des outils de manipulation des formes, offre une plateforme puissante pour explorer l'interaction entre programmation et visualisation graphique.