

Rapport de Projet : Draw++

Chekhab Elias
Guihot Nathan
Aissaoui Ahmed
Zheng Lise
Benmansour Firas

5 janvier 2025

Table des matières

1	Introduction	3
2	La Syntaxe du Langage	4
2.1	Grammaire BNF du Langage Draw++	4
2.2	Variables Obligatoires	7
2.3	Éléments de Base	7
2.3.1	Terminaison des Instructions	7
2.3.2	Commentaires	7
2.3.3	Délimiteurs	8
2.4	Types et Variables	8
2.4.1	Déclaration des Variables	8
2.5	Structures de Contrôle	8
2.5.1	Conditions	8
2.5.2	Boucles	8
2.6	Manipulation du Curseur	9
2.7	Dessiner des Formes	9
2.8	Couleurs et Styles	10
3	Description du traducteur	11
3.1	Architecture du traducteur	11
3.2	Processus de traduction	12
3.2.1	Code Draw++	12
3.2.2	Code C généré	12
4	Bibliothèque graphique SDL2	13
4.1	Introduction à SDL2	13
4.2	Utilisation dans Draw++	13
4.2.1	Gestion de la fenêtre	13
4.2.2	Gestion des couleurs	13
4.3	Implémentation des fonctionnalités graphiques	14
4.3.1	Système de curseur	14

4.3.2	Formes géométriques	14
4.4	Avantages de SDL2	15
5	Gestion des erreurs	16
5.1	Types d'erreurs et traitements associés	16
5.2	Exemple d'intégration	17
6	Interface de Développement Intégré (IDE)	18
6.1	Vue d'ensemble	18
6.2	Composants principaux	18
6.2.1	Barre de menu	18
6.2.2	Éditeur de code	20
6.2.3	Zone de prévisualisation	20
6.2.4	Terminal intégré	21
6.3	Fonctionnalités d'analyse et débogage	21
6.3.1	Analyse lexicale	21
6.3.2	Retours d'erreur	22
6.3.3	Vérification en temps réel	22
6.4	Bonnes pratiques d'utilisation	22
6.4.1	Initialisation du projet	22
6.4.2	Débogage	22
7	Exemples d'exécution	23
7.1	Exemple 1 : Dessin d'un cercle	25
7.2	Exemple 2 : Dessin d'un triangle	27
7.3	Exemple 3 : Dessin du drapeau de la France	28
7.4	Exemple 4 : Dessin d'une ligne	29
7.5	Exemple 5 : Dessin d'un rectangle	31
7.6	Exemple 6 : Dessin d'une ellipse	32
7.7	Sortie du terminal	33
8	Conclusion	34

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 :

```
1 <drawpp_program> ::= <statement_list>
2
3 <statement_list> ::= <statement> ";" | <statement> ";" <
   statement_list>
4
5 <statement> ::= <variable_declaration>
6               | <constant_declaration>
7               | <assignment>
8               | <compound_assignment>
9               | <if_statement>
10              | <loop_statement>
11              | <cursor_statement>
12              | <shape_statement>
13              | <comment>
14
15 <variable_declaration> ::= "var" <type> <identifier> "="
   <expression>
16
17 <type> ::= "int" | "float" | "bool" | "color"
18
19 <assignment> ::= <identifier> "=" <expression>
20 <compound_assignment> ::= <identifier> <compound_operator>
   <expression>
21 <compound_operator> ::= "+=" | "-=" | "*=" | "/="
22
```

```

23 <if_statement> ::= "if" "(" <condition> ")" "{" <
    statement_list> "}"
24             [ <elif_chain> ]
25             [ "else" "{" <statement_list> "}" ]
26
27 <elif_chain> ::= "elif" "(" <condition> ")" "{" <
    statement_list> "}" [ <elif_chain> ]
28
29 <loop_statement> ::= <for_loop> | <while_loop>
30
31 <for_loop> ::= "for" "(" [ <variable_declaration> ] ";" <
    condition> ";" [ <assignment> ] ")"
32             "{" <statement_list> "}"
33
34 <while_loop> ::= "while" "(" <condition> ")" "{" <
    statement_list> "}"
35
36 <cursor_statement> ::= <cursor_creation>
37                     | <cursor_movement>
38                     | <cursor_style>
39                     | <cursor_visibility>
40
41 <cursor_creation> ::= "cursor" <identifier> "=" "
    create_cursor" "(" <expression> "," <expression> ")"
42
43 <cursor_movement> ::= <identifier> "." ( <move> | <rotate>
    > | <position> )
44 <move> ::= "move" "(" <expression> ")"
45 <rotate> ::= "rotate" "(" <expression> ")"
46 <position> ::= "position" "(" <expression> "," <
    expression> ")"
47
48 <cursor_style> ::= <identifier> "." ( <color> | <
    thickness> )
49 <color> ::= "color" "(" <color_value> ")"
50 <thickness> ::= "thickness" "(" <expression> ")"
51
52 <cursor_visibility> ::= <identifier> "." "visible" "("
53
54 <shape_statement> ::= <identifier> "." <shape_type>
55
56 <shape_type> ::= <draw_line> | <draw_rectangle> | <
    draw_circle> | <draw_triangle> | <draw_ellipse>

```

```

57
58 <draw_line> ::= "draw_line" "(" <expression> ")"
59
60 <draw_rectangle> ::= "draw_rectangle" "(" <expression> ",
    " <expression> ", " <bool_value> ")"
61
62 <draw_circle> ::= "draw_circle" "(" <expression> ", " <
    bool_value> ")"
63
64 <draw_triangle> ::= "draw_triangle" "(" <expression> ", "
    <expression> ", " <bool_value> ")"
65
66 <draw_ellipse> ::= "draw_ellipse" "(" <expression> ", " <
    expression> ", " <bool_value> ")"
67
68 <color_value> ::= <predefined_color> | <custom_color>
69 <predefined_color> ::= "BLACK" | "WHITE" | "RED" | "GREEN
    " | "BLUE"
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86

```

| "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"

```

87
88 <comment> ::= <single_line_comment> | <multi_line_comment>
      >
89 <single_line_comment> ::= "//" <text> <newline>
90 <multi_line_comment> ::= "/*" <text> "*/"
91
92 <identifiant> ::= [a-zA-Z_][a-zA-Z0-9_]*
93 <number> ::= [0-9]+ ( "." [0-9]+ )?
94 <text> ::= [^"]*
95 <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 déclarations doivent apparaître avant toute autre instruction. Toute omission entraînera une erreur à l'exécution.

2.3 Éléments 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; // D clARATION d'une variable
2 cursor.move(100); // D placement 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 Délimiteurs

Les délimiteurs structurent le code et incluent :

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

2.4 Types et Variables

2.4.1 Déclaration des Variables

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

```
1 var int compteur = 0;
2 var float pi = 3.14159;
3 var color myColor = rgb(255, 0, 0);
4 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>`

2.5.2 Boucles

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

- `while (<condition>) <bloc d'instructions>`
- `for (<initialisation>; <condition>; <mise à jour>) <bloc d'instructions>`

2.6 Manipulation du Curseur

Le curseur est l'objet central pour dessiner. Il est défini avec la syntaxe suivante :

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

Les propriétés et méthodes principales du curseur incluent :

- `cursor.move(distance)` : Déplace le curseur.
- `cursor.rotate(angle)` : Tourne le curseur.
- `cursor.color(<couleur>)` : Définit la couleur.
- `cursor.thickness(<valeur>)` : Définit l'épaisseur du trait.
- `cursor.visible()` : Contrôle la visibilité du curseur.

Exemple de création 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 dessinées avec les méthodes 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 personnalisé avec des couleurs prédéfinies ou des couleurs personnalisées. Exemple :

```
1 c.color(RED);  
2 c.thickness(3); // D'init l' épaisseur du trait  
3 c.color(rgb(255, 0, 0)); // Rouge personnalisé
```

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 généré

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 SDL2

4.1 Introduction à SDL2

La SDL2 (Simple DirectMedia Layer 2) est une bibliothèque multimédia libre écrite en C qui a été utilisée comme base pour l'implémentation graphique de Draw++. Cette bibliothèque offre une couche d'abstraction pour l'accès au matériel informatique, particulièrement adaptée pour le développement d'applications graphiques et de jeux vidéo.

4.2 Utilisation dans Draw++

Dans le cadre du projet Draw++, SDL2 a été principalement utilisée pour implémenter les fonctionnalités graphiques suivantes :

4.2.1 Gestion de la fenêtre

SDL2 nous permet de créer et gérer la fenêtre de rendu avec les fonctions suivantes :

- `SDL_CreateWindow` : Création de la fenêtre d'affichage
- `SDL_CreateRenderer` : Initialisation du moteur de rendu
- `SDL_RenderPresent` : Mise à jour de l'affichage

4.2.2 Gestion des couleurs

La bibliothèque offre des fonctionnalités complètes pour la gestion des couleurs :

- `SDL_SetRenderDrawColor` : Définition de la couleur de dessin
- Support des couleurs RGB avec canal alpha pour la transparence

- Gestion des couleurs prédéfinies (BLACK, WHITE, RED, GREEN, BLUE)

4.3 Implémentation des fonctionnalités graphiques

4.3.1 Système de curseur

Le curseur virtuel de Draw++ est implémenté en utilisant les coordonnées SDL2 :

- Position stockée en coordonnées x, y
- Orientation gérée en degrés
- Déplacement relatif calculé via des transformations trigonométriques

4.3.2 Formes géométriques

Avec la bibliothèque SDL2, il est possible de dessiner une variété de formes géométriques. Voici quelques exemples :

- **Lignes** : Utilisez la fonction `SDL_RenderDrawLine` pour dessiner des lignes entre deux points spécifiés.

```
1 SDL_RenderDrawLine(renderer, x1, y1, x2, y2);
```

- **Rectangles** : Les rectangles peuvent être dessinés avec les fonctions `SDL_RenderDrawRect` pour des rectangles vides et `SDL_RenderFillRect` pour des rectangles pleins.

```
1 SDL_RenderDrawRect(renderer, &rect); //
   Rectangle vide
2 SDL_RenderFillRect(renderer, &rect); //
   Rectangle plein
```

- **Cercles et Ellipses** : Bien que SDL2 ne fournisse pas de fonction spécifique pour dessiner des cercles ou des ellipses, ces formes peuvent être tracées en utilisant des algorithmes qui calculent les points sur le périmètre de la forme. Il est ainsi possible de dessiner des cercles ou des ellipses pleins ou seulement leurs contours.
- **Triangles** : Les triangles peuvent être créés en reliant trois points à l'aide de la fonction `SDL_RenderDrawLine`.

En utilisant ces primitives de dessin, vous pouvez facilement créer des formes géométriques adaptées à vos besoins dans des applications graphiques.

4.4 Avantages de SDL2

L'utilisation de SDL2 dans Draw++ présente plusieurs avantages :

- **Portabilité** : SDL2 fonctionne sur de nombreuses plateformes (Windows, Linux, MacOS)
- **Performance** : Accès direct aux fonctionnalités graphiques matérielles
- **Simplicité** : API claire et bien documentée
- **Légèreté** : Bibliothèque légère avec peu de dépendances
- **Communauté active** : Ressources et support disponibles

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** : Détectées 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** : Suggérer 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 suggérer 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** : Suggérer de convertir les types ou d'utiliser des types compatibles.
 - *Variables déjà déclarées* : Réutilisation 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 Développement Intégré (IDE)

6.1 Vue d'ensemble

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

- Un éditeur de code avec numérotation des lignes
- Une zone de prévisualisation graphique
- Un terminal intégré

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 : Créer 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** : Exécution du code
- **Help** : Accès à l'aide

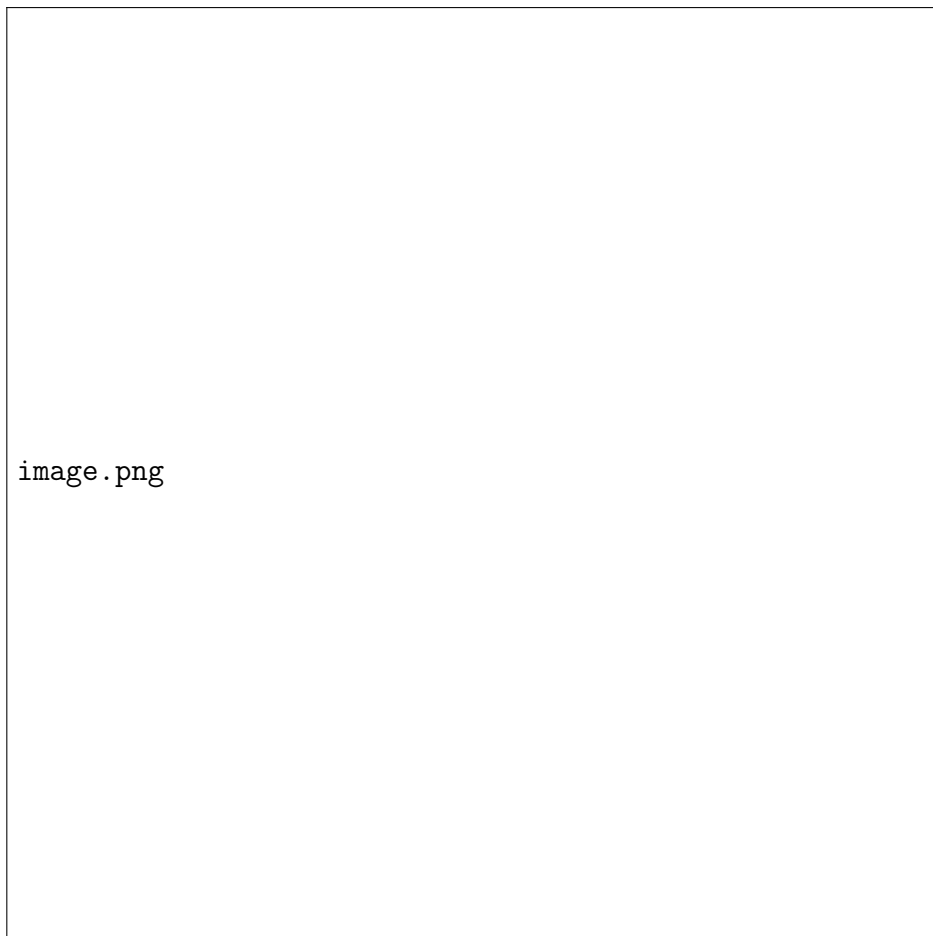


FIGURE 6.1 – Interface de Développement Intégré Draw++.

6.2.2 Éditeur de code

L'éditeur intègre plusieurs fonctionnalités avancées :

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

6.2.3 Zone de prévisualisation

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 intégré

Le terminal propose plusieurs commandes essentielles :

1	<code>clear</code>	- Efface l'écran
2	<code>exit</code>	- Quitte le terminal
3	<code>help</code>	- Affiche l'aide des commandes disponibles
4	<code>lex</code>	- Effectue l'analyse lexicale d'une ligne de code
5	<code>parse</code>	- Effectue l'analyse syntaxique et genere l'AST
6	<code>run</code>	- Execute le code Draw++
7	<code>save</code>	- Sauvegarde le code source, les tokens ou l'AST

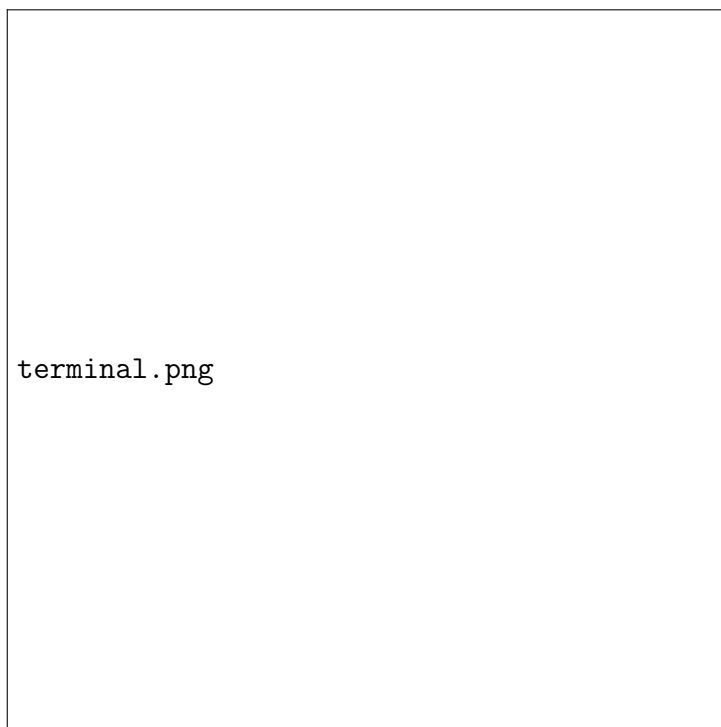
6.3 Fonctionnalités d'analyse et débogage

6.3.1 Analyse lexicale

Le terminal affiche les informations détaillées pour chaque token généré :

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

Un exemple de sortie d'analyse lexicale :



6.3.2 Retours d'erreur

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

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

6.3.3 Vérification en temps réel

L'IDE effectue plusieurs vérifications pendant la saisie :

- Validation des variables obligatoires (`windowWidth`, `windowHeight`)
- Vérification de la syntaxe
- Détection des erreurs de type
- Validation des appels de méthodes sur les curseurs

6.4 Bonnes pratiques d'utilisation

6.4.1 Initialisation du projet

Pour démarrer un nouveau projet efficacement :

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

6.4.2 Débogage

Pour un débogage efficace :

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

Chapitre 7

Exemples d'exécution

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

7.1 Exemple 1 : Dessin d'un cercle

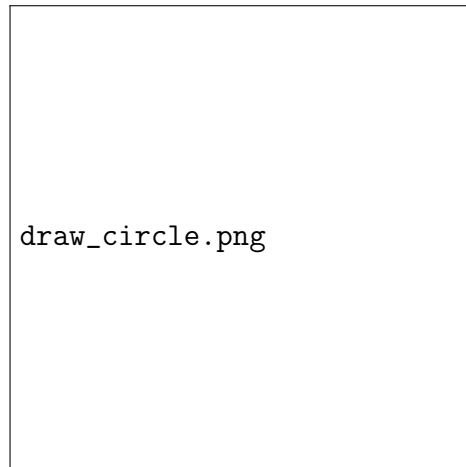


FIGURE 7.1 – Rendu graphique du cercle.

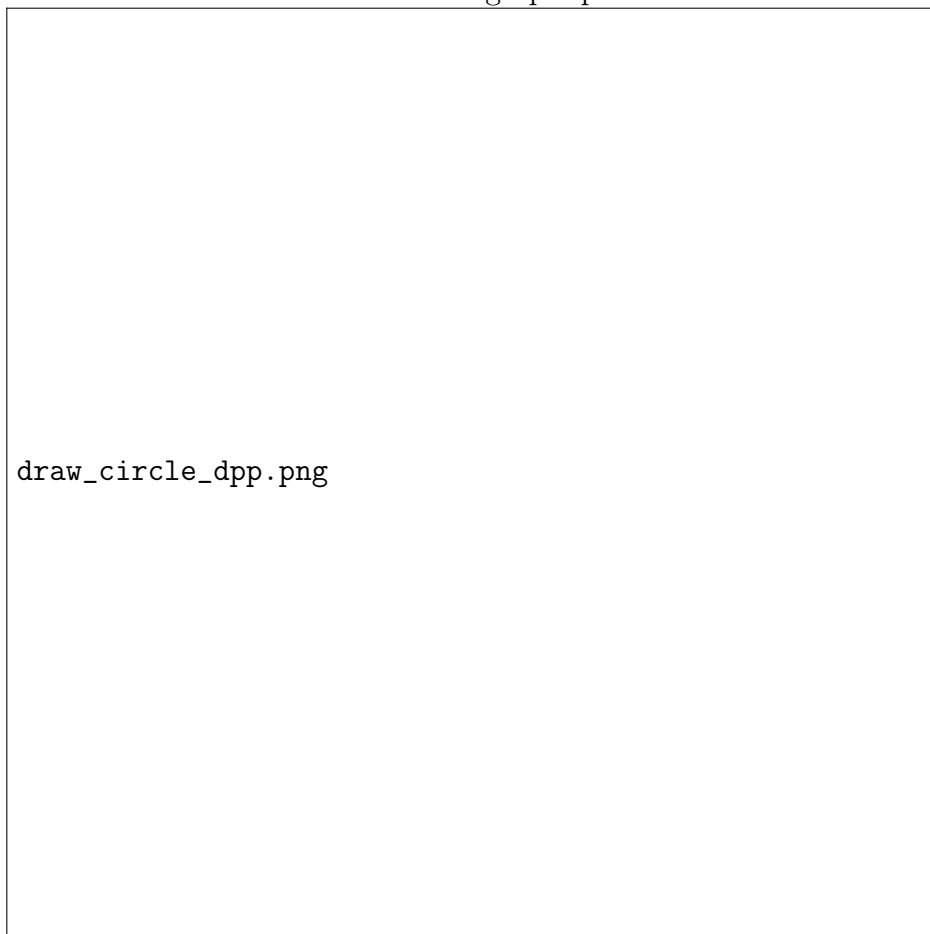


FIGURE 7.2 – Code DPP pour dessiner un cercle.

7.2 Exemple 2 : Dessin d'un triangle

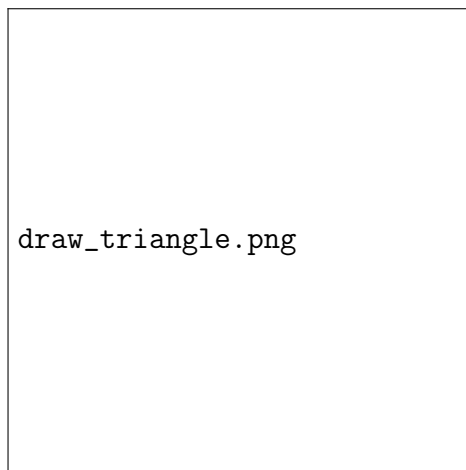


FIGURE 7.3 – Rendu graphique du triangle.



FIGURE 7.4 – Code DPP₂₇ 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

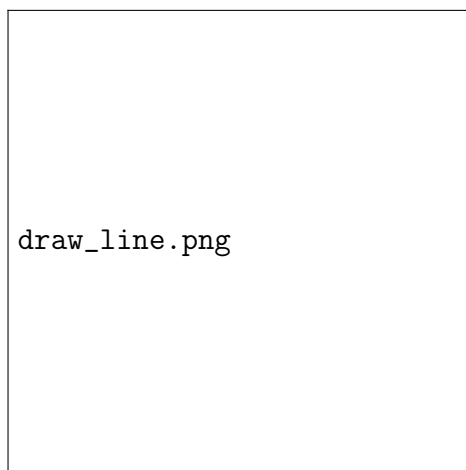


FIGURE 7.7 – Rendu graphique de la ligne.

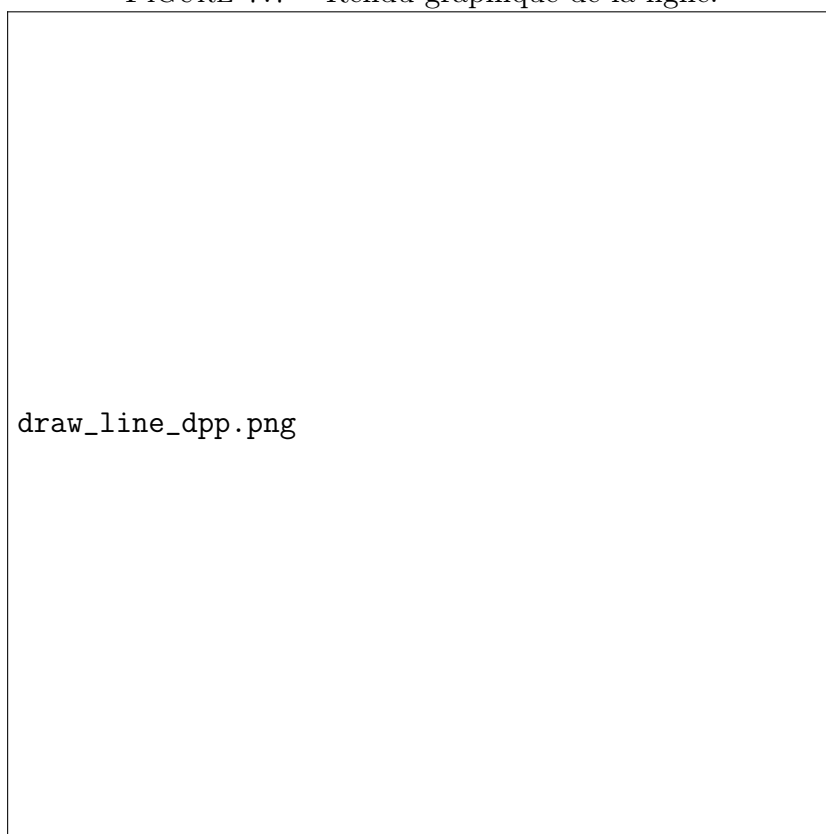


FIGURE 7.8 – Code DPP pour dessiner une ligne.

7.5 Exemple 5 : Dessin d'un rectangle

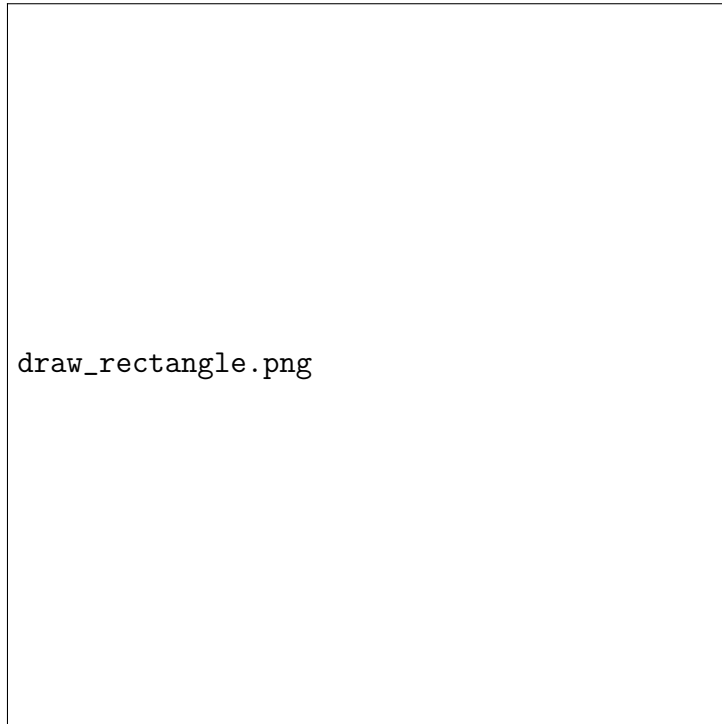
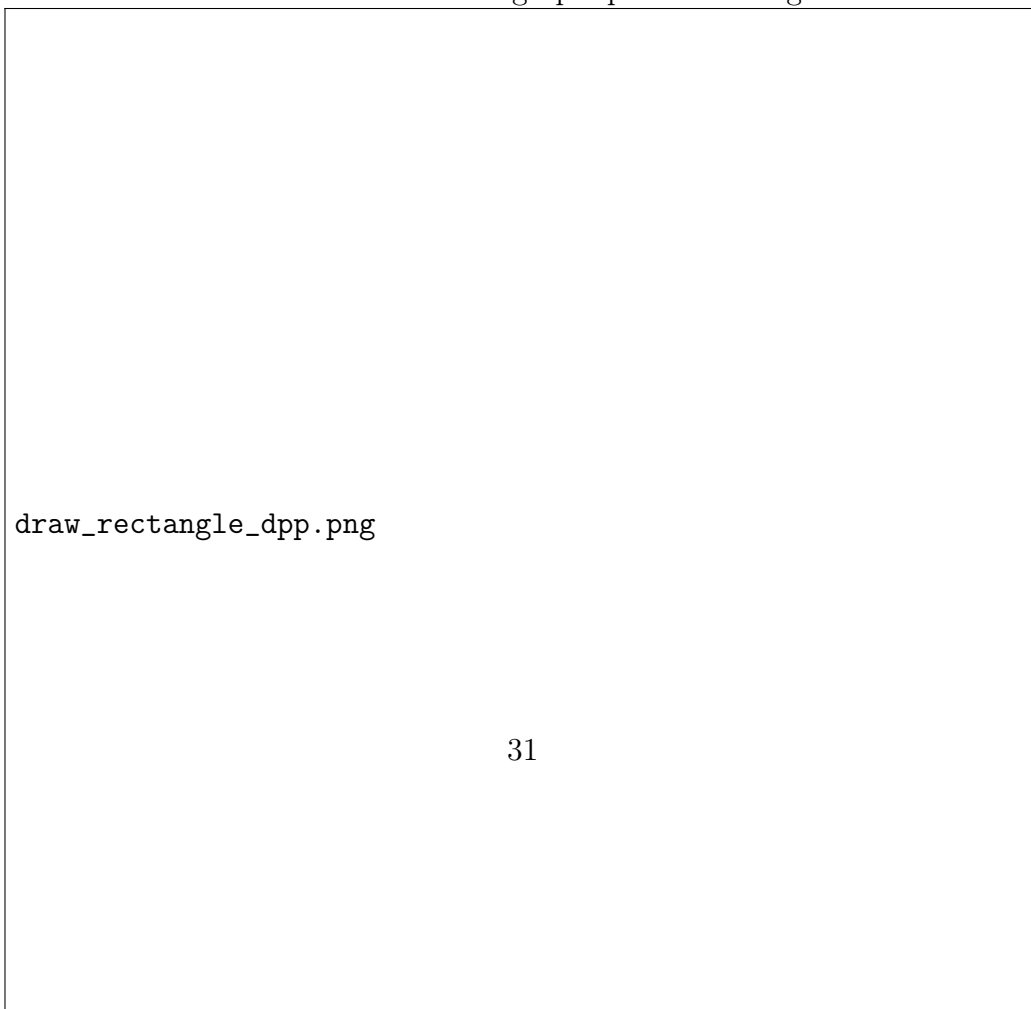


FIGURE 7.9 – Rendu graphique du rectangle.



7.6 Exemple 6 : Dessin d'une ellipse

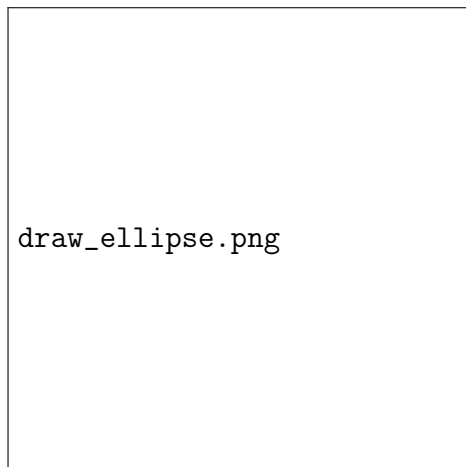


FIGURE 7.11 – Rendu graphique de l'ellipse.

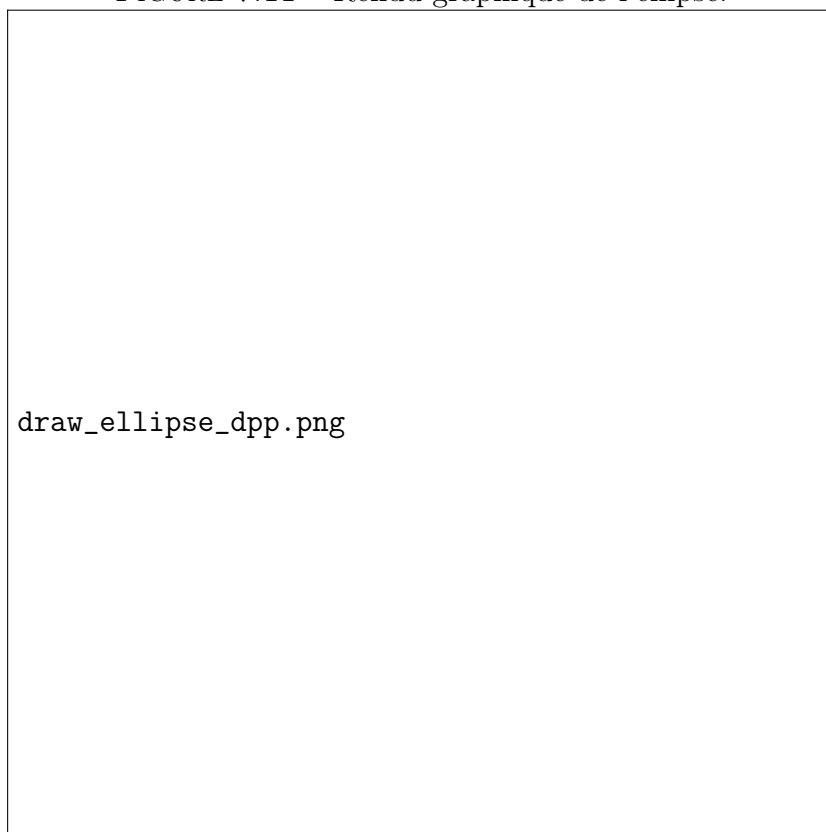
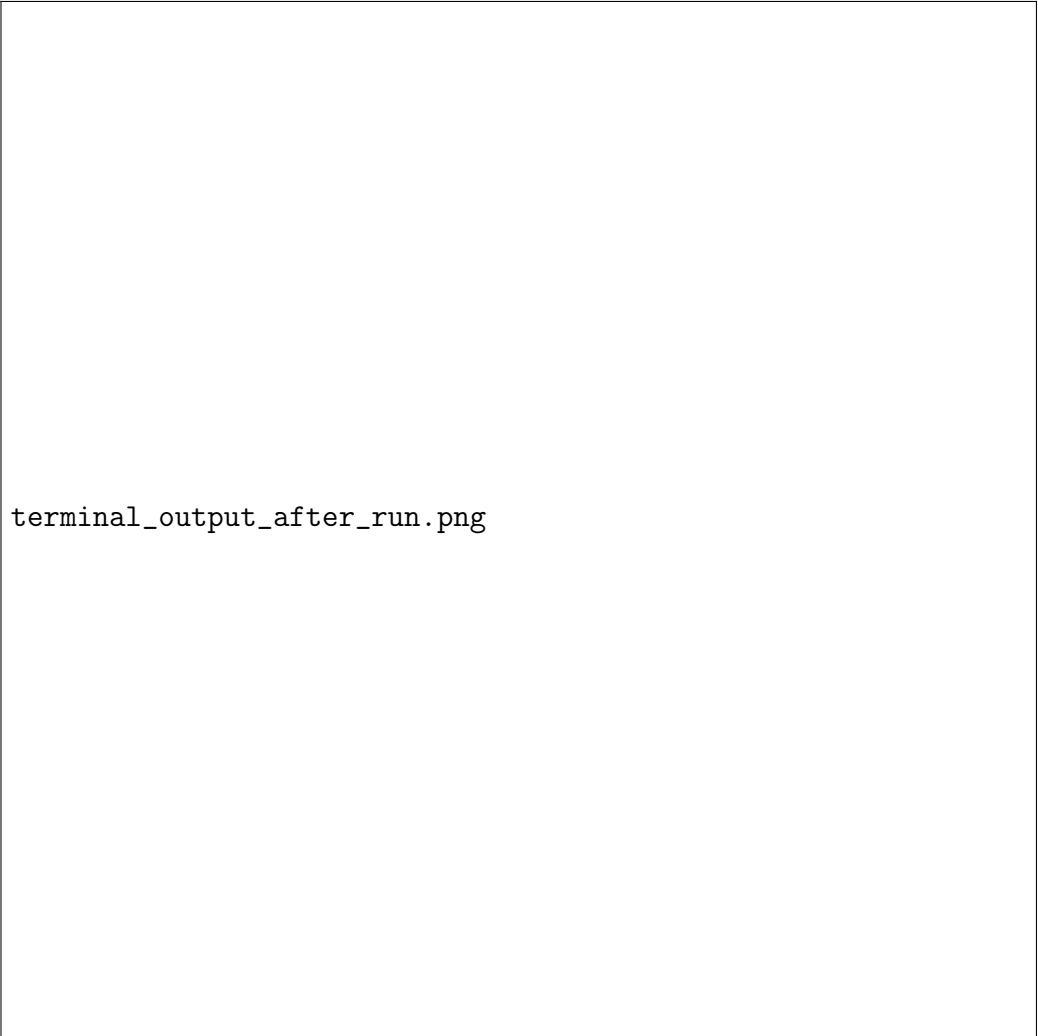


FIGURE 7.12 – Code DPP pour dessiner une ellipse.

7.7 Sortie du terminal



`terminal_output_after_run.png`

Chapitre 8

Conclusion

Le projet Draw++ a permis la conception d'un langage graphique minimaliste, couplé 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 exécution fluide et précise du code. La structure du langage, combinée à des outils de manipulation des formes, offre une plateforme puissante pour explorer l'interaction entre programmation et visualisation graphique.