FACULTÉ DES SCIENCES ET TECHNIQUES
DE MOHAMMEDIA
UNIVERSITÉ HASSAN II DE CASABLANCA

جـامعة الحسن الثـاني بالدار البيضاء
UNIVERSITÉ HASSAN II DE CASABLANCA

STRUCTURES DE DONNÉES

Application Éducative

Version C/Raylib & Python/PySide6

Réalisé par :
Taha Amine Er-Rafiy

Filière :
LST Ingénierie Informatique

Encadré par : Mr. KHOUKHI

Date : Décembre 2025

Remerciements

Au terme de ce travail, je tiens à exprimer ma profonde reconnaissance envers **Monsieur KHOUKHI**, Professeur du module Structures de Données à la Faculté des Sciences et Techniques de Mohammedia. Son enseignement rigoureux et sa pédagogie m'ont permis d'acquérir les fondements théoriques nécessaires à la réalisation de ce projet.

Ce projet m'a permis d'acquérir des compétences techniques significatives dans le domaine du développement logiciel et de la visualisation des structures de données. La conception d'une application complète, alliant programmation système en **C** avec la bibliothèque graphique **Raylib** et développement d'interfaces utilisateur modernes en **Python** avec **PySide6**, constitue une expérience formatrice qui a considérablement enrichi mon parcours académique.

Je mesure pleinement la valeur de cette opportunité qui m'a permis de repousser les frontières de mes connaissances et de démontrer qu'avec un encadrement de qualité et un investissement personnel soutenu, il est possible de mener à bien des projets d'envergure dépassant les attentes initiales.

Taha Amine Er-Rafiy
Mohammedia, Décembre 2025

Table des matières

Chapitre 1

Introduction

1.1 Écran d'Accueil

C - Écran d'Accueil

Python - Écran d'Accueil

1.2 Barre de Navigation

Une **barre de navigation** est présente en haut de chaque module, permettant de basculer facilement entre les différentes sections de l'application. Cette barre contient quatre boutons principaux :

- **Accueil** : Retourne à l'écran d'accueil principal
- **Tableaux** : Accède au module de manipulation et tri des tableaux
- **Listes** : Accède au module des listes chaînées (simples et doubles)
- **Arbres** : Accède au module de visualisation des arbres (binaires et N-aires)
- **Graphes** : Accède au module de manipulation des graphes et algorithmes de chemin

Le bouton correspondant au module actif est mis en surbrillance avec une couleur distinctive (bordure verte). Les autres boutons sont affichés avec un effet de survol au passage de la souris. Cette navigation cohérente permet une expérience utilisateur fluide et intuitive entre les différents modules.

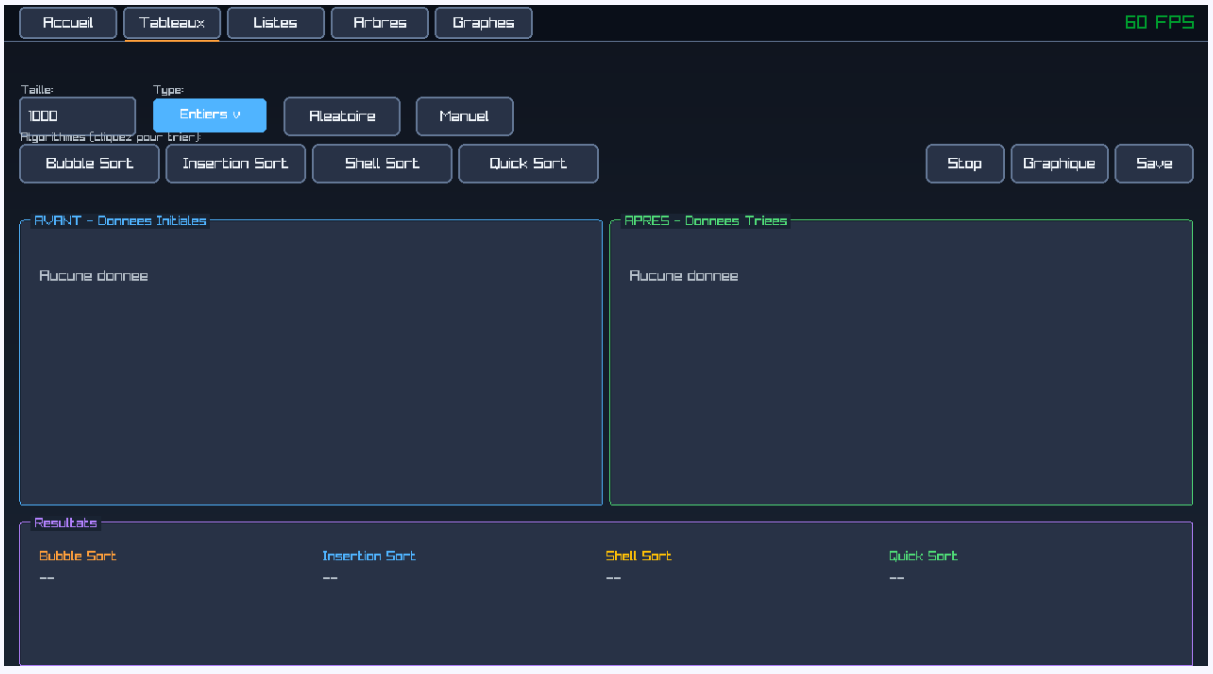
Chapitre 2

Module Tableaux

2.1 Interface Générale

L'interface du module Tableaux offre une disposition ergonomique organisée en plusieurs zones distinctes. La **barre supérieure** contient les contrôles de génération (boutons "Aléatoire" et "Manuel", champ de taille), le menu déroulant de type de données (Entiers, Réels, Caractères, Chaînes), et les boutons d'algorithmes de tri. La zone centrale affiche deux **panels** "AVANT" et "APRÈS" montrant les données initiales et triées côte à côte. En bas, une **barre de résultats** affiche les temps d'exécution de chaque algorithme avec un code couleur. Un **graphique de performance** visualise les courbes de progression des différents tris.

C - Interface



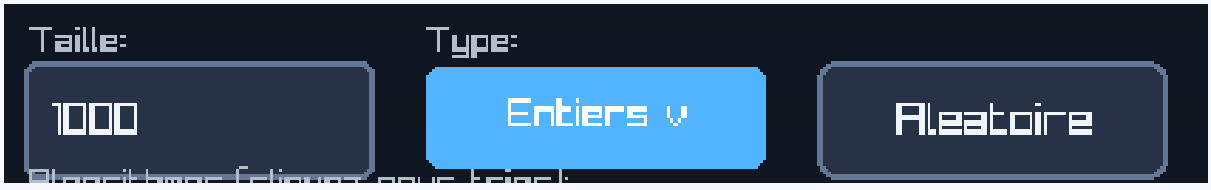
Python - Interface

2.2 Saisie des Données

2.2.1 Saisie Aléatoire

L'utilisateur peut générer automatiquement un tableau de données aléatoires en spécifiant la taille souhaitée dans le champ de saisie. Le bouton **"Aléatoire"** génère des valeurs aléatoires adaptées au type sélectionné : nombres entiers, nombres réels, caractères alphanumriques, ou chaînes de texte. La génération est quasi-instantanée même pour de très grands tableaux (jusqu'à plusieurs millions d'éléments).

C - Génération Aléatoire

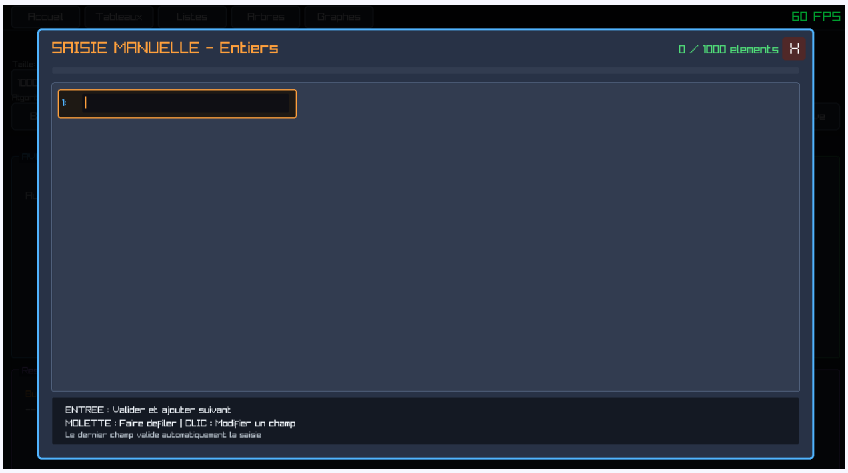


Python - Génération Aléatoire

2.2.2 Saisie Manuelle

Le bouton **"Manuel"** ouvre une fenêtre de saisie interactive permettant d'entrer les valeurs une par une. La progression est affichée avec une barre de progression et un compteur indiquant le nombre d'éléments saisis par rapport au total. L'utilisateur peut naviguer dans les valeurs déjà saisies avec la molette de la souris, et modifier une valeur existante en cliquant dessus.

C - Saisie Manuelle



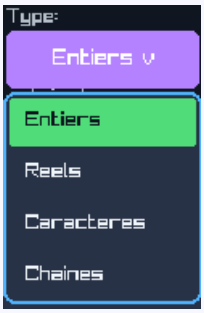
Python - Saisie Manuelle

2.3 Menu Déroulant Type de Données

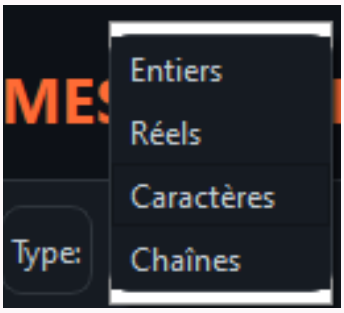
Un menu déroulant situé dans la barre supérieure permet de sélectionner le type de données à manipuler. Quatre types sont disponibles : **Entiers** (nombres entiers signés), **Réels** (nombres

à virgule flottante), **Caractères** (lettres et symboles uniques), et **Chaînes** (mots ou textes). Le type choisi affecte la génération aléatoire, l'affichage dans les panels, et la comparaison lors du tri.

C - Menu Type



Python - Menu Type

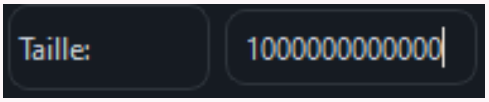


2.4 Taille Illimitée

La taille du tableau n'est pas limitée artificiellement et peut contenir des millions d'éléments, limitée uniquement par la mémoire disponible. L'utilisateur saisit simplement le nombre souhaité dans le champ de taille. Cette capacité permet de tester les performances des algorithmes de tri sur de très grands ensembles de données.

C - Champ Taille

Python - Champ Taille



2.5 Algorithmes de Tri

Quatre algorithmes de tri sont implémentés. Cliquer sur un bouton lance **tous les algorithmes** simultanément pour permettre la comparaison.

2.5.1 Tri à Bulles (Bubble Sort)

Compare les éléments adjacents et les échange s'ils sont dans le mauvais ordre. Complexité : $O(n^2)$.

C

```c
void BubbleSortInt(long long *arr, int n) {
  for (int i = 0; i < n-1; i++)
    for (int j = 0; j < n-i-1; j++)
      if (arr[j] > arr[j+1]) {
        long long t = arr[j]; arr[j] = arr[j+1]; arr[j+1] = t;
      }
}
```

Python

```python
def bubble_sort(self, arr):
  n = len(arr)
  for i in range(n-1):
    for j in range(n-i-1):
      if arr[j] > arr[j+1]:
        arr[j], arr[j+1] = arr[j+1], arr[j]
```

2.5.2 Tri par Insertion (Insertion Sort)

Insère chaque élément à sa position correcte dans la partie déjà triée. Complexité : $O(n^2)$.

C

```c
void InsertionSortInt(long long *arr, int n) {
  for (int i = 1; i < n; i++) {
    long long key = arr[i]; int j = i - 1;
    while (j >= 0 && arr[j] > key) { arr[j+1] = arr[j]; j--; }
    arr[j+1] = key;
  }
}
```

Python

```python
def insertion_sort(self, arr):
  for i in range(1, len(arr)):
    key, j = arr[i], i - 1
    while j >= 0 and arr[j] > key:
      arr[j+1] = arr[j]; j -= 1
    arr[j+1] = key
```

2.5.3 Tri Shell (Shell Sort)

Amélioration du tri par insertion utilisant des intervalles décroissants. Complexité : O(n log n) à O(n^2).

C

```c
void ShellSortInt(long long *arr, int n) {
  for (int gap = n/2; gap > 0; gap /= 2)
    for (int i = gap; i < n; i++) {
      long long t = arr[i]; int j;
      for (j = i; j >= gap && arr[j-gap] > t; j -= gap)
        arr[j] = arr[j-gap];
      arr[j] = t;
    }
}
```

Python

```python
def shell_sort(self, arr):
  n, gap = len(arr), len(arr) // 2
  while gap > 0:
    for i in range(gap, n):
      t, j = arr[i], i
      while j >= gap and arr[j-gap] > t:
        arr[j] = arr[j-gap]; j -= gap
      arr[j] = t
    gap //= 2
```

2.5.4 Tri Rapide (Quick Sort)

Algorithme “diviser pour régner” utilisant un pivot. Complexité moyenne : O(n log n).

C

```c
void QuickSortInt(long long *arr, int lo, int hi) {
  if (lo < hi) {
    long long pivot = arr[hi]; int i = lo - 1;
    for (int j = lo; j < hi; j++)
      if (arr[j] < pivot) { i++;
        long long t = arr[i]; arr[i] = arr[j]; arr[j] = t; }
    long long t = arr[i+1]; arr[i+1] = arr[hi]; arr[hi] = t;
    int pi = i + 1;
    QuickSortInt(arr, lo, pi-1);
    QuickSortInt(arr, pi+1, hi);
  }
}
```

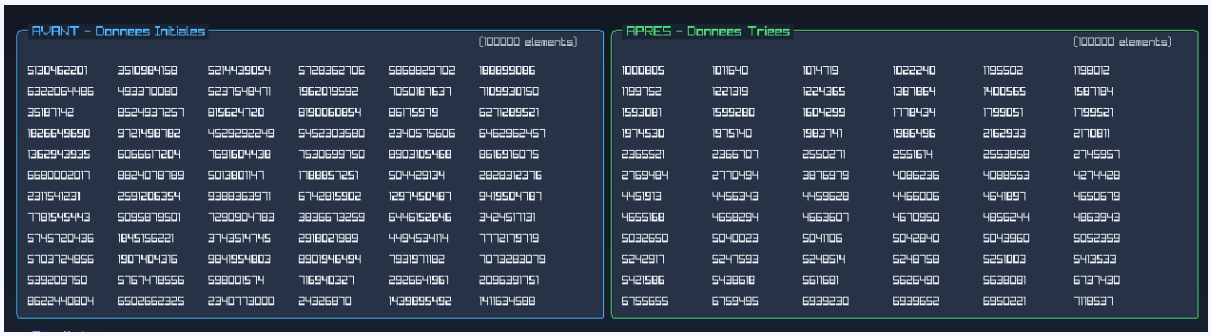
Python

```python
def quick_sort(self, arr, lo, hi):
  if lo < hi:
    pivot, i = arr[hi], lo - 1
    for j in range(lo, hi):
      if arr[j] < pivot:
        i += 1; arr[i], arr[j] = arr[j], arr[i]
    arr[i+1], arr[hi] = arr[hi], arr[i+1]
    pi = i + 1
    self.quick_sort(arr, lo, pi-1)
    self.quick_sort(arr, pi+1, hi)
```

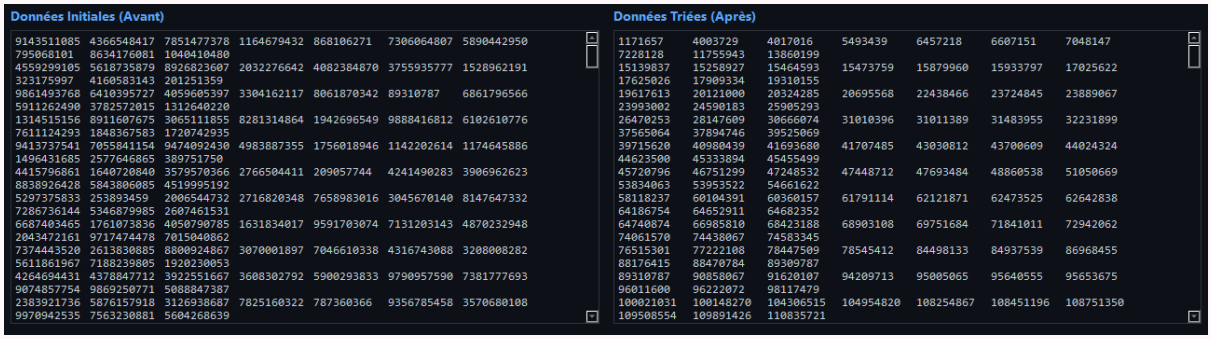
2.6 Affichage Avant/Après Tri

L'interface affiche deux panels côte à côte :
— **AVANT** : Les données initiales non triées
— **APRÈS** : Les données triées par l'algorithme sélectionné
Fonctionnement : Lorsqu'un bouton d'algorithme est cliqué, **tous les tris se lancent en parallèle** (threads séparés). Le panel "APRÈS" affiche le résultat du **tri sélectionné**, tandis que la barre de résultats montre les temps de tous les algorithmes.

C - Affichage Avant/Après



Python - Affichage Avant/Après

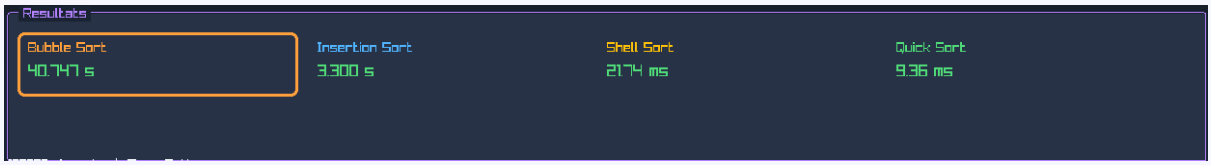


2.7 Barre de Résultats

Une barre de résultats affiche le **temps d'exécution** de chaque algorithme avec un code couleur. L'algorithme sélectionné est encadré. Les tris en cours affichent un compteur animé.

Compteur flexible : L'affichage du temps s'adapte automatiquement à la durée : microsecondes (µs), millisecondes (ms), secondes (s), ou minutes (min) selon le temps écoulé.

C - Barre Résultats

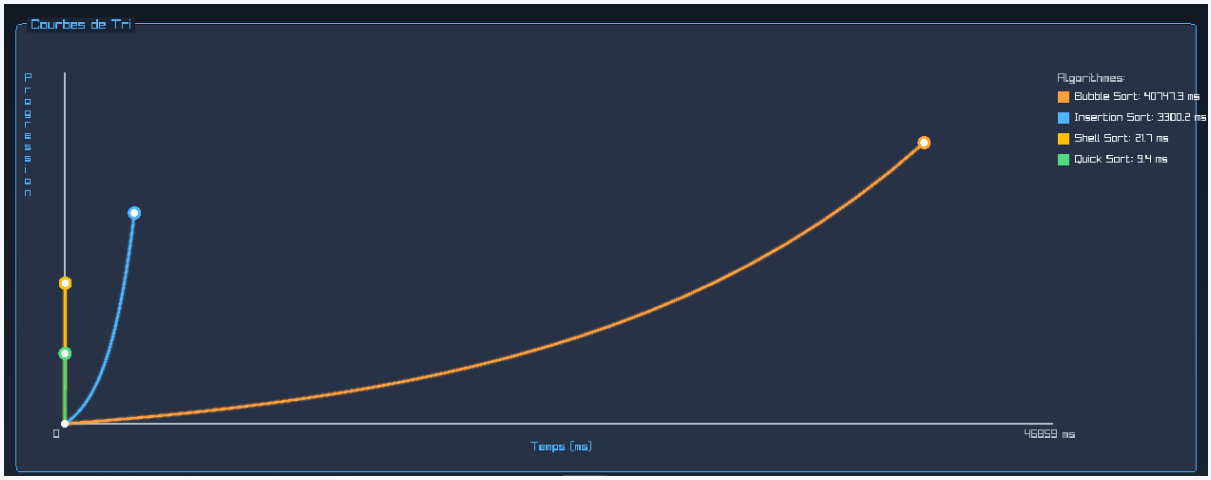


Python - Barre Résultats

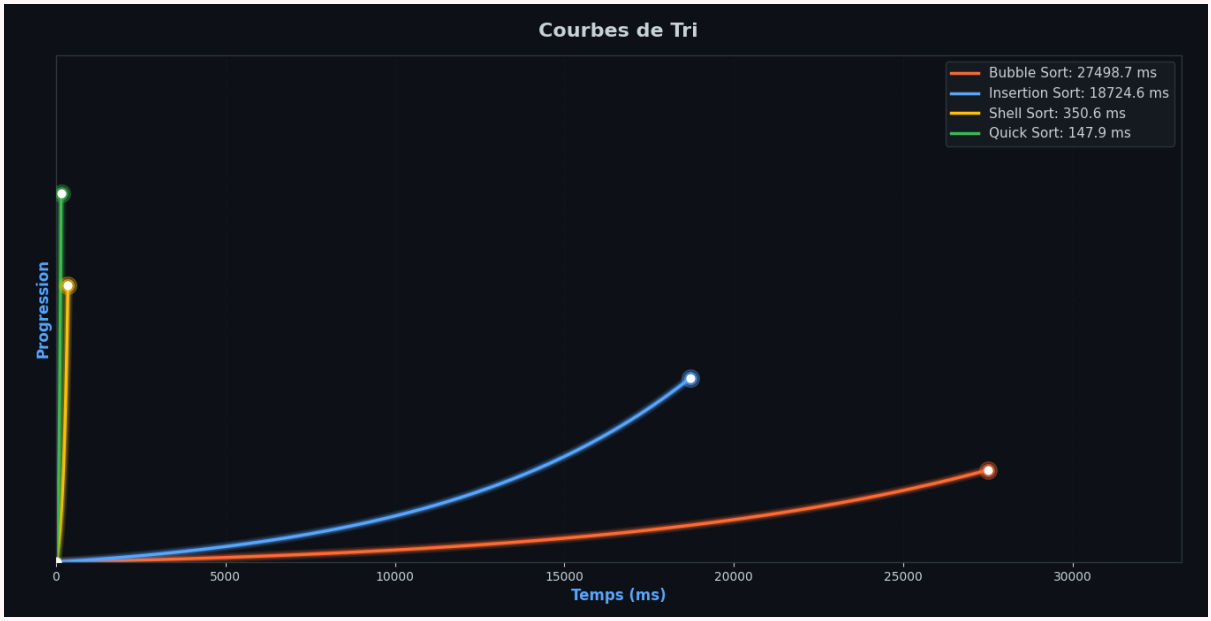
2.8 Graphique de Performance

Un graphique interactif visualise les **courbes de progression** de chaque algorithme. Les axes représentent le temps (ms) et la progression. Une légende affiche les temps finaux.

C - Graphique Performance



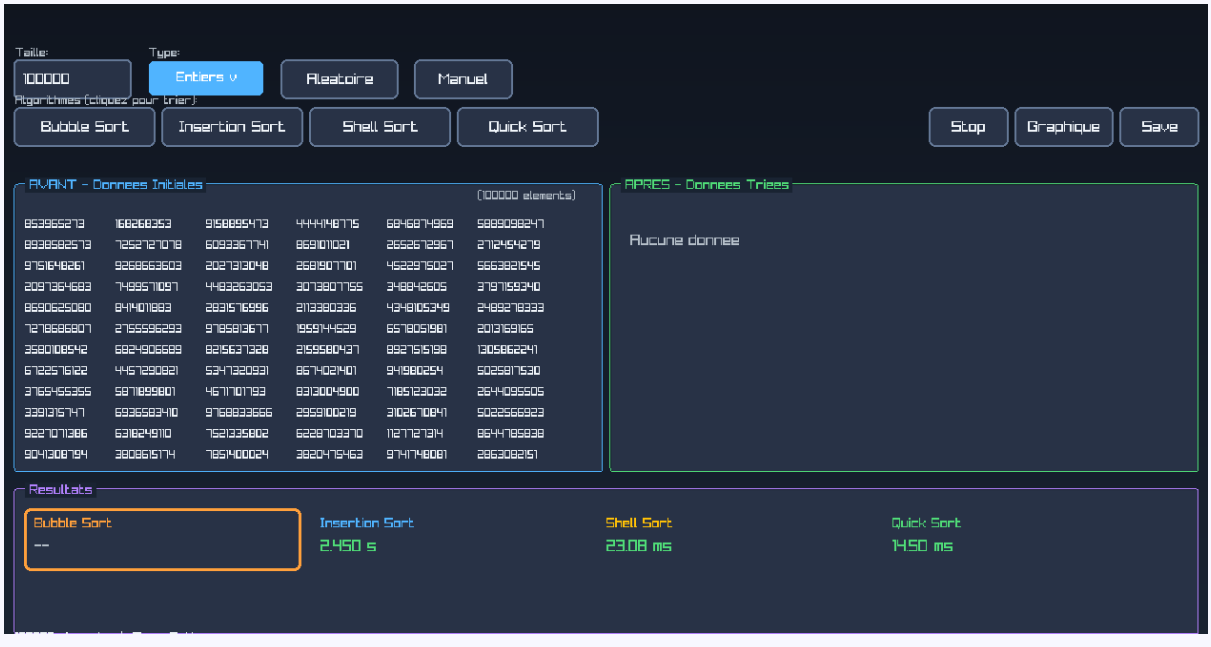
Python - Graphique Performance



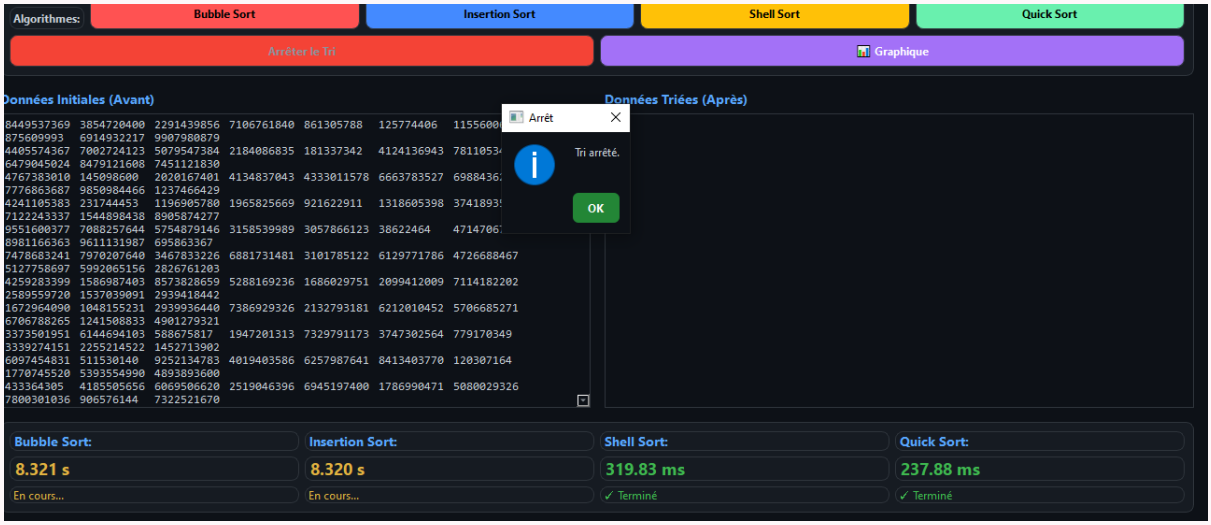
2.9 Bouton Arrêter

Le bouton **"Stop"** permet d'**interrompre** tous les tris en cours d'exécution. Utile pour les grands tableaux dont le tri prend trop de temps.

C - Bouton Stop



Python - Bouton Stop



2.10 Visualisation Animée des Tris (Python)

La version Python propose une **interface visuelle animée** pour observer le tri en temps réel. Les éléments du tableau sont représentés par des barres dont la hauteur correspond à la valeur, permettant de visualiser les comparaisons et échanges.

NB : Cette fonctionnalité est spécifique à la version Python.

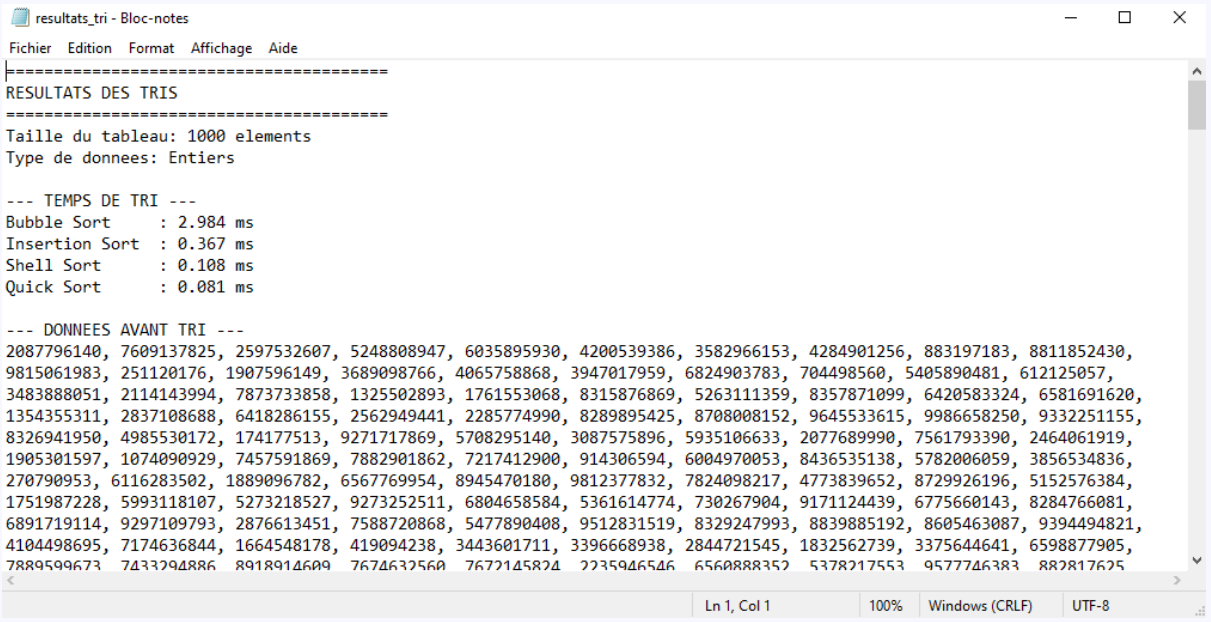
Python - Visualisation Animée

2.11 Bouton Sauvegarder

Le bouton **"Save"** exporte les résultats dans un fichier `resultats_tri.txt` contenant : les temps de tri, les données avant et après tri, et les informations du tableau.

NB : Cette fonctionnalité est disponible uniquement dans la version C/Raylib.

C - Bouton Sauvegarder



Chapitre 3

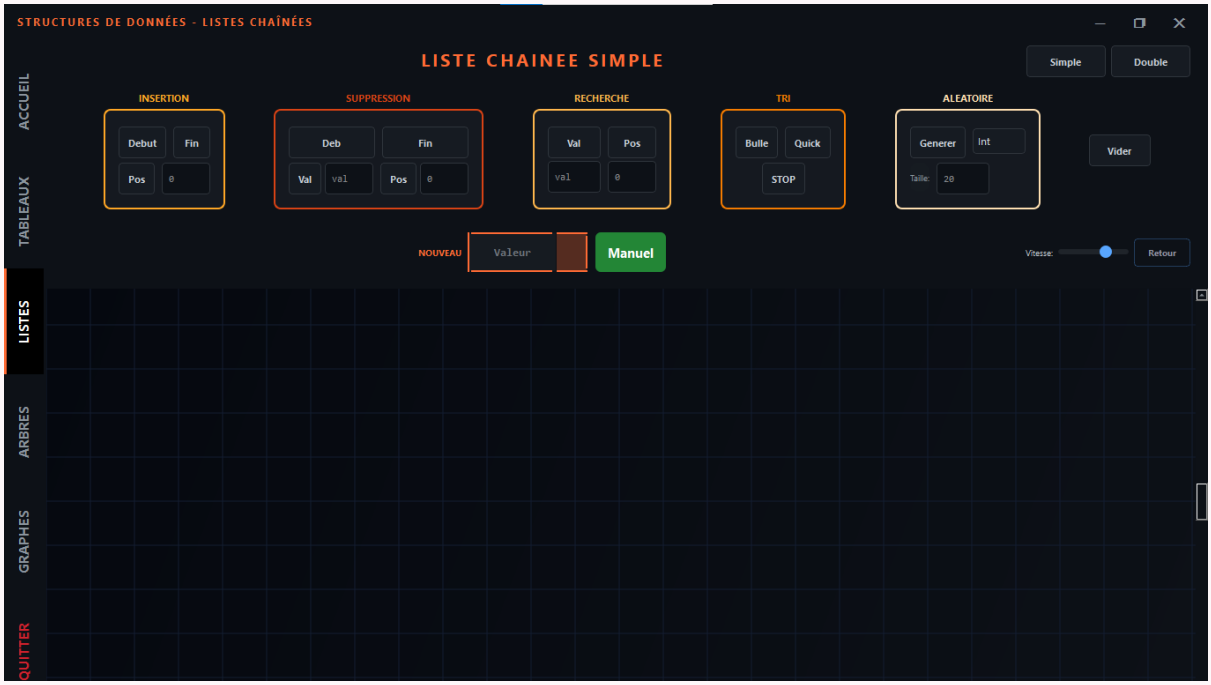
Module Listes

3.1 Interface Générale

L'interface du module Listes offre une expérience interactive pour manipuler des listes chaînées (simples ou doubles). La zone centrale affiche la **visualisation graphique** des nœuds connectés par des flèches, avec possibilité de défilement horizontal pour les longues listes. Un **nœud d'entrée** flottant permet de saisir directement une nouvelle valeur. Sur les côtés, des **cadres colorés** regroupent les opérations : INSERTION (vert), SUPPRESSION (rouge), GÉNÉRATION (bleu), RECHERCHE (jaune), et TRI (violet). Le titre dynamique en haut indique le type de liste actif ("LISTE CHAINEE SIMPLE" ou "LISTE CHAINEE DOUBLE").

C - Interface

Python - Interface

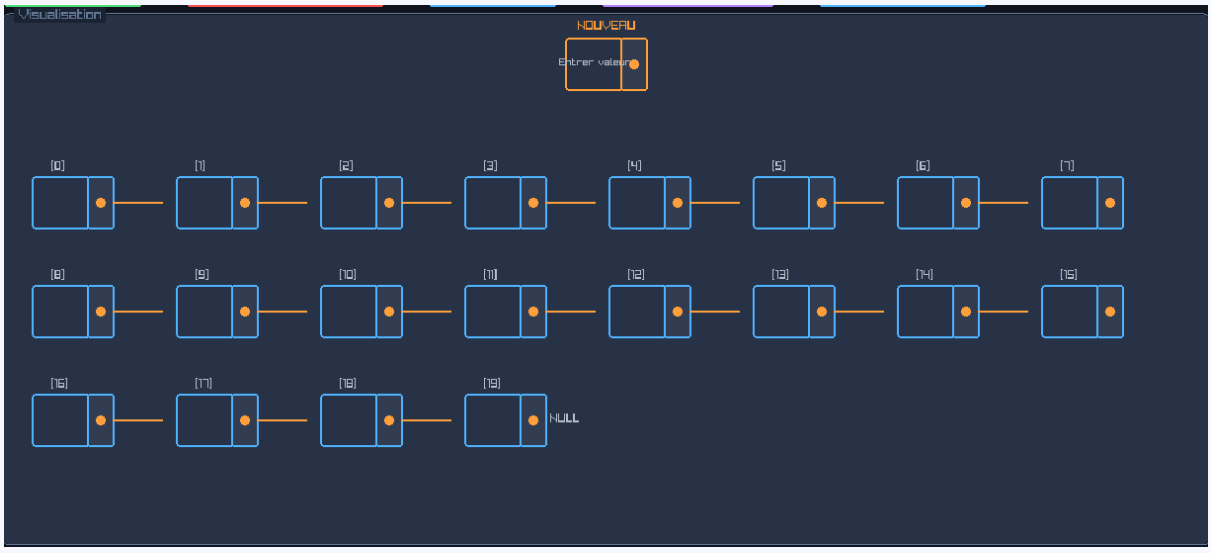


3.2 Types de Listes

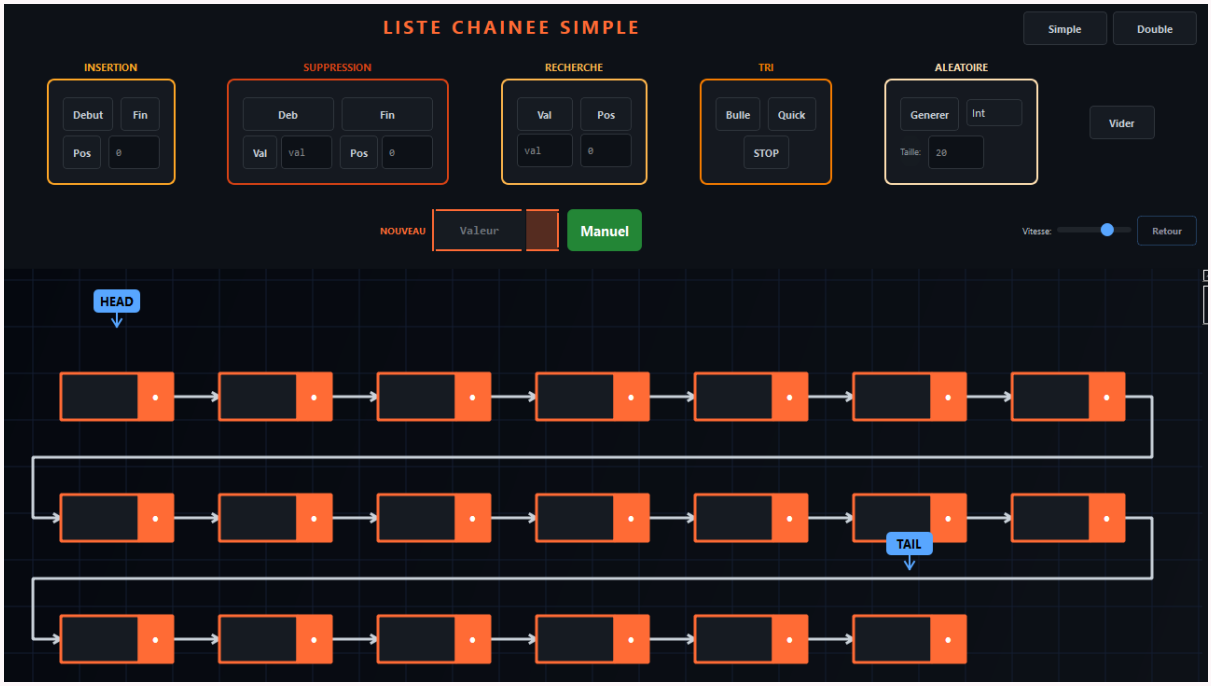
3.2.1 Liste Simplement Chaînée

Dans une liste simplement chaînée, chaque nœud contient une **donnée** (texte jusqu'à 32 caractères) et un **pointeur** vers le nœud suivant. Le dernier nœud pointe vers NULL, visualisé par une case vide. Les flèches montrent le sens du parcours (unidirectionnel, de gauche à droite). Cette structure est optimale pour les insertions/suppressions en tête de liste (O(1)).

C - Liste Simple



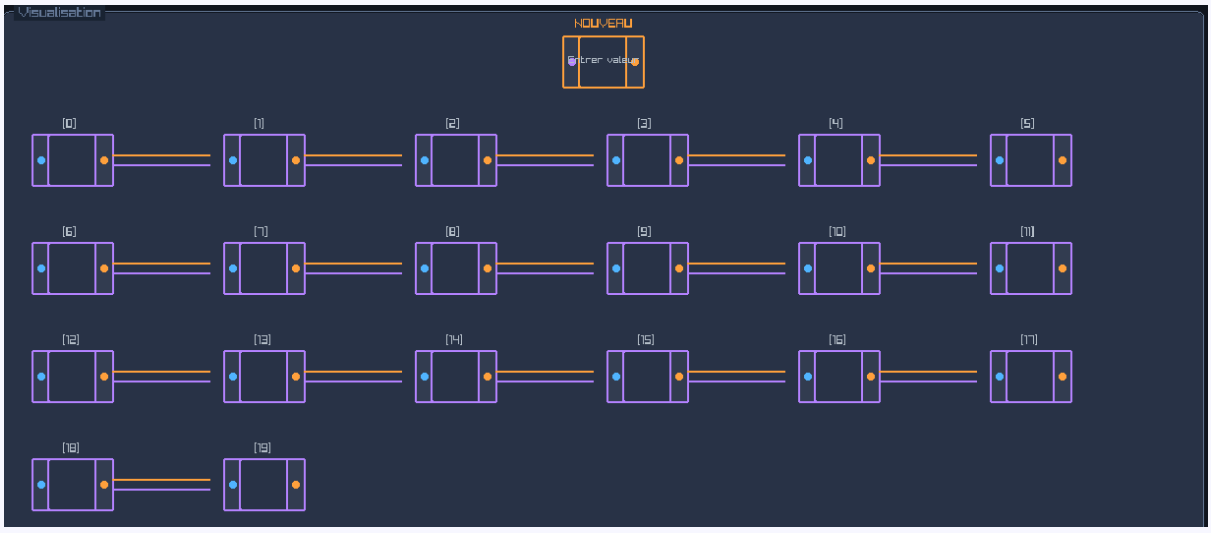
Python - Liste Simple



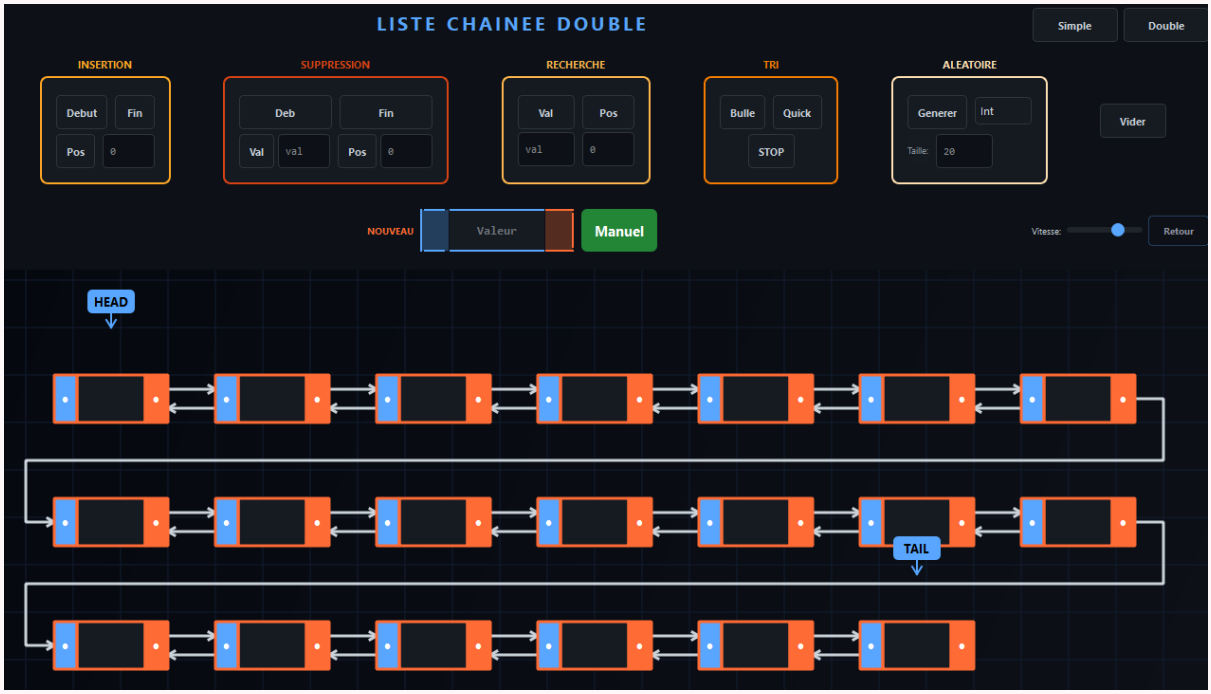
3.2.2 Liste Doublement Chaînée

Dans une liste doublement chaînée, chaque nœud contient une donnée et **deux pointeurs** : un vers le nœud précédent (prev) et un vers le suivant (next). Cette structure permet le parcours dans les deux sens, visualisé par des flèches bidirectionnelles. Elle facilite la suppression d'un nœud quelconque en O(1) si on a déjà la référence au nœud.

C - Liste Double



Python - Liste Double



3.3 Structures de Données

C

```c
typedef struct NodeSimple {
  char data[32];
  struct NodeSimple *next;
} NodeSimple;

typedef struct NodeDouble {
  char data[32];
  struct NodeDouble *prev, *next;
} NodeDouble;

typedef struct { NodeSimple *head; int count; } ListeSimple;
typedef struct { NodeDouble *head, *tail; int count; } ListeDouble;
```

Python

```python
class NodeSimple:
  def __init__(self, data):
    self.data, self.next = data, None

class NodeDouble:
  def __init__(self, data):
    self.data, self.prev, self.next = data, None, None
```

3.4 Cadre INSERTION

Le cadre d'insertion contient trois boutons permettant d'ajouter un élément :
— **Début** : Insère au début de la liste
— **Fin** : Insère à la fin de la liste
— **Position** : Insère à une position spécifique (demande l'index)

C

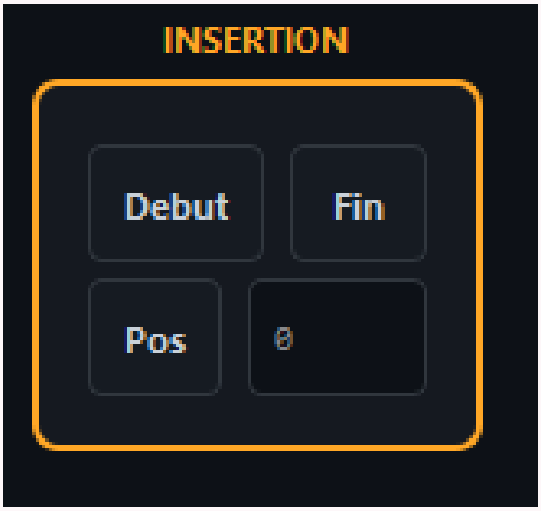
```c
void InsererDebut(ListeSimple *l, const char *v) {
  NodeSimple *n = malloc(sizeof(NodeSimple));
  strncpy(n->data, v, 31); n->next = l->head;
  l->head = n; l->count++;
}
void InsererFin(ListeSimple *l, const char *v) {
  NodeSimple *n = malloc(sizeof(NodeSimple));
  strncpy(n->data, v, 31); n->next = NULL;
  if (!l->head) l->head = n;
  else { NodeSimple *c = l->head; while(c->next) c=c->next; c->next = n
      ; }
  l->count++;
}
void InsererPosition(ListeSimple *l, const char *v, int pos) {
  if (pos <= 0) { InsererDebut(l, v); return; }
  NodeSimple *n = malloc(sizeof(NodeSimple));
  strncpy(n->data, v, 31);
  NodeSimple *c = l->head; int i = 0;
  while (c && i < pos-1) { c = c->next; i++; }
  if (c) { n->next = c->next; c->next = n; l->count++; }
}
```

Python

```python
def inserer_debut(self, value):
  n = NodeSimple(value)
  n.next, self.head = self.head, n
  self.count += 1

def inserer_fin(self, value):
  n = NodeSimple(value)
  if not self.head: self.head = n
  else:
    c = self.head
    while c.next: c = c.next
    c.next = n
  self.count += 1

def inserer_position(self, value, pos):
  if pos <= 0: return self.inserer_debut(value)
  n, c, i = NodeSimple(value), self.head, 0
  while c and i < pos-1: c, i = c.next, i+1
  if c: n.next, c.next = c.next, n; self.count += 1
```

C - Cadre Insertion

Python - Cadre Insertion



3.5 Cadre SUPPRESSION

Le cadre de suppression contient trois boutons :
— **Début** : Supprime le premier élément
— **Fin** : Supprime le dernier élément
— **Valeur** : Supprime un élément par sa valeur (demande la valeur)

C

```c
bool SupprimerDebut(ListeSimple *l) {
  if (!l->head) return false;
  NodeSimple *t = l->head; l->head = l->head->next;
  free(t); l->count--; return true;
}
bool SupprimerFin(ListeSimple *l) {
  if (!l->head) return false;
  if (!l->head->next) { free(l->head); l->head = NULL; }
  else {
    NodeSimple *c = l->head;
    while (c->next->next) c = c->next;
    free(c->next); c->next = NULL;
  }
  l->count--; return true;
}
bool SupprimerValeur(ListeSimple *l, const char *v) {
  NodeSimple *c = l->head, *p = NULL;
  while (c) {
    if (strcmp(c->data, v) == 0) {
      if (!p) l->head = c->next; else p->next = c->next;
      free(c); l->count--; return true;
    }
    p = c; c = c->next;
  }
  return false;
}
```

Python

```python
def supprimer_debut(self):
  if not self.head: return False
  self.head = self.head.next
  self.count -= 1; return True

def supprimer_fin(self):
  if not self.head: return False
  if not self.head.next: self.head = None
  else:
    c = self.head
    while c.next.next: c = c.next
    c.next = None
  self.count -= 1; return True

def supprimer_valeur(self, value):
  c, p = self.head, None
  while c:
    if c.data == value:
      if not p: self.head = c.next
      else: p.next = c.next
      self.count -= 1; return True
    p, c = c, c.next
  return False
```

Animation : Lors de la suppression, une animation montre l'élément qui disparaît et les liens qui se reconnectent.

C - Cadre Suppression

Python - Cadre Suppression

3.6 Cadre RECHERCHE

Le cadre de recherche propose deux modes :
— **Val** : Recherche par valeur - affiche la position de l'élément
— **Pos** : Recherche par position - affiche l'élément à cet index

C

```c
int ListeSimple_GetPosition(ListeSimple *l, const char *v) {
  NodeSimple *c = l->head; int i = 0;
  while (c) {
    if (strcmp(c->data, v) == 0) return i;
    c = c->next; i++;
  }
  return -1;
}
NodeSimple* ListeSimple_GetAt(ListeSimple *l, int pos) {
  NodeSimple *c = l->head; int i = 0;
  while (c && i < pos) { c = c->next; i++; }
  return c;
}
```

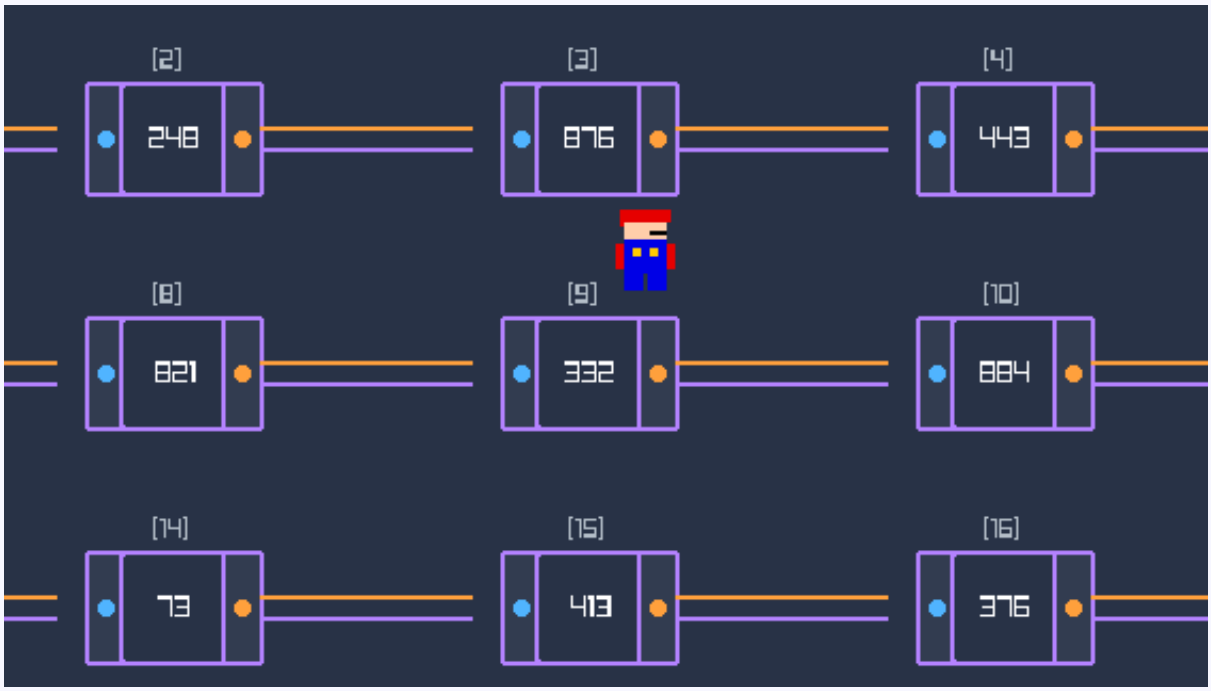
Python

```python
def rechercher_valeur(self, value):
  c, i = self.head, 0
  while c:
    if c.data == value: return i
    c, i = c.next, i + 1
  return -1

def rechercher_position(self, pos):
  c, i = self.head, 0
  while c and i < pos:
    c, i = c.next, i + 1
  return c.data if c else None
```

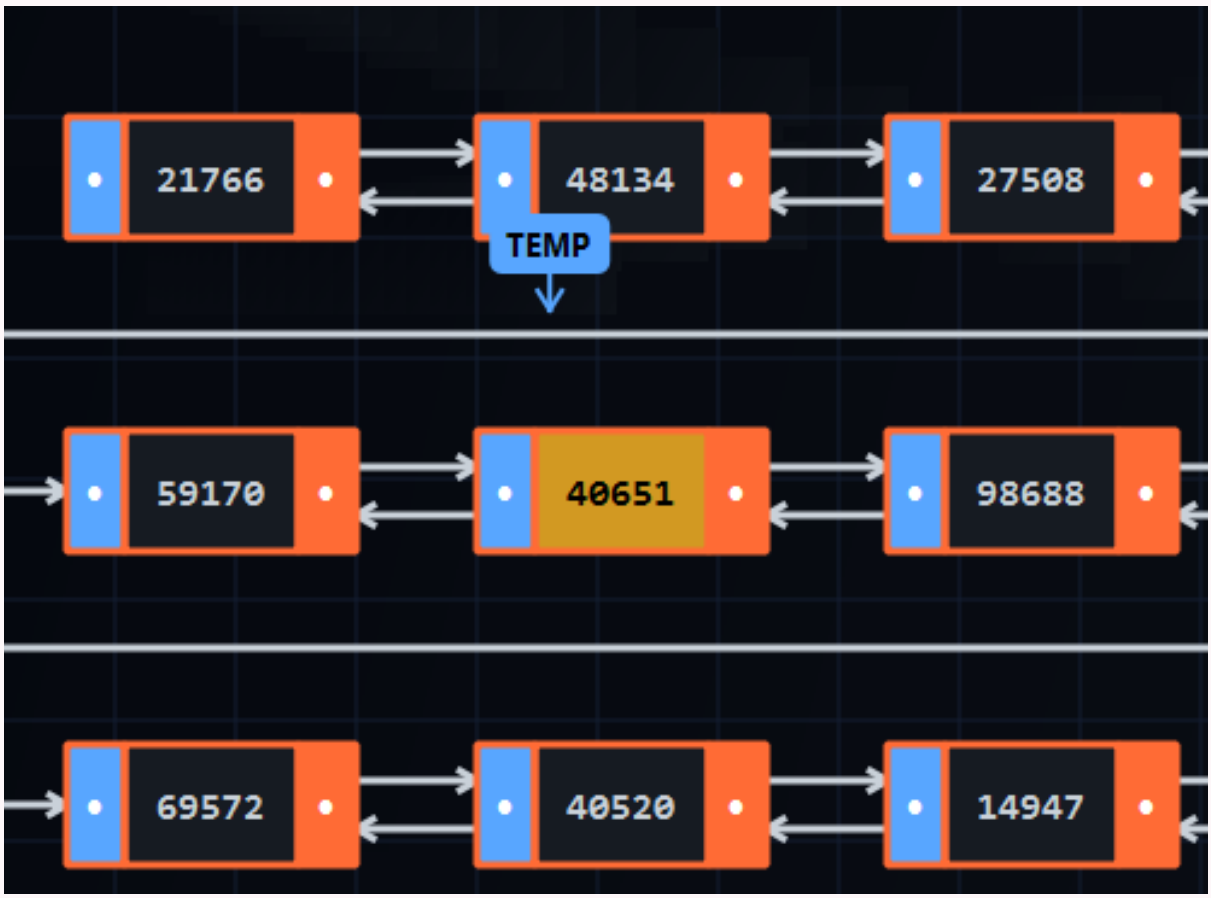
3.6.1 Recherche en cours

Lorsqu'une recherche est lancée, le programme parcourt la liste élément par élément avec une **animation visuelle**. Dans la version C/Raylib, le personnage **Mario** se déplace le long de la liste, passant de nœud en nœud jusqu'à trouver l'élément recherché. Chaque nœud visité est temporairement mis en surbrillance. Un **compteur de temps** affiche la durée écoulée depuis le début de la recherche en temps réel.

C - Recherche En Cours



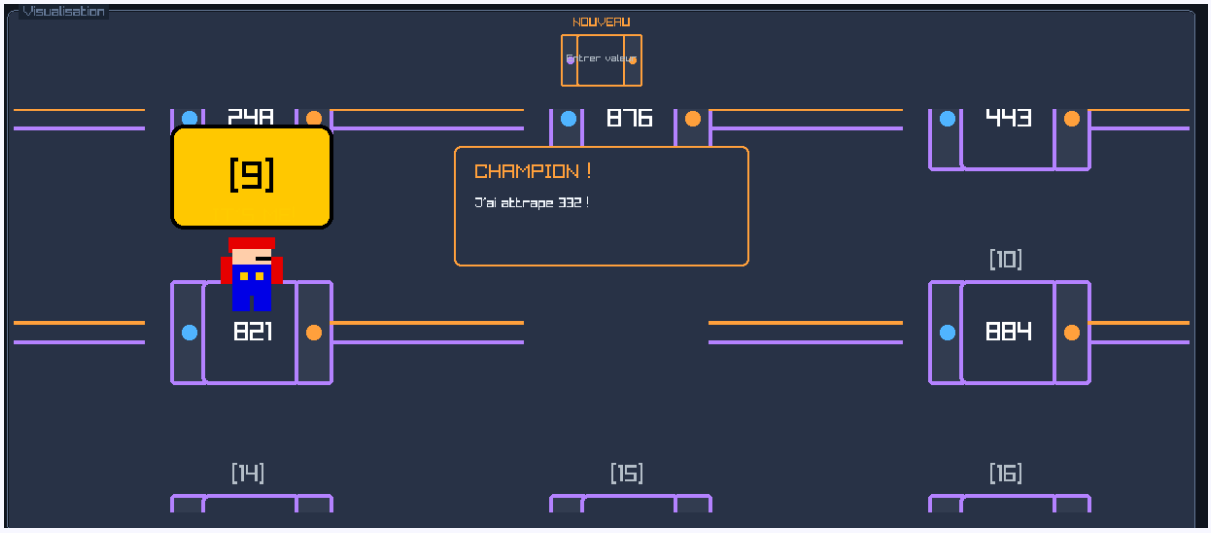
Python - Recherche En Cours



3.6.2 Valeur Trouvée

Quand l'élément est trouvé, le résultat s'affiche avec plusieurs informations : la **valeur trouvée**, sa **position** dans la liste (index), et le **temps de recherche** (affiché en microsecondes, millisecondes ou secondes selon la durée). L'élément correspondant reste en surbrillance verte dans la visualisation. Si l'élément n'est pas trouvé, un message d'erreur s'affiche.

C - Recherche Trouvée



Python - Recherche Trouvée

3.7 Génération de Données

3.7.1 Génération Aléatoire

Permet de générer automatiquement une liste avec un nombre spécifié d'éléments aléatoires.

C - Génération Aléatoire

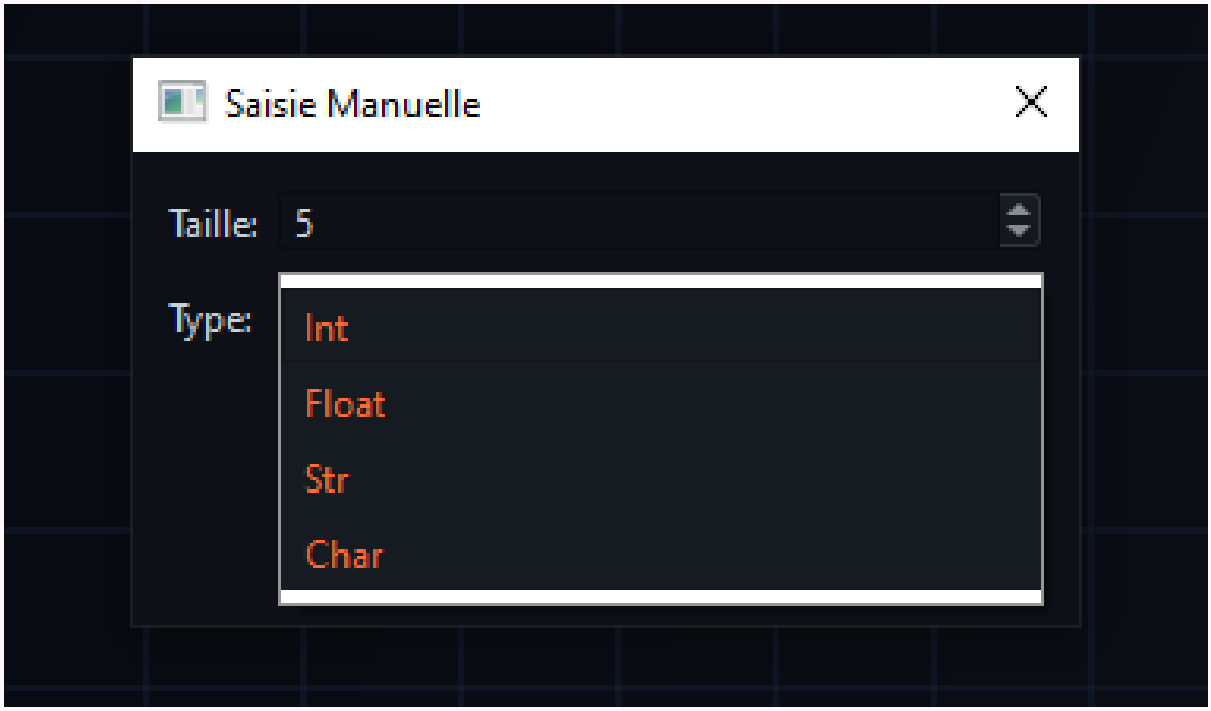
Python - Génération Aléatoire

3.7.2 Génération Manuelle (Popup)

Un clic sur le bouton **Manuel** ouvre un menu popup permettant de configurer la génération : type de données et nombre d'éléments. L'utilisateur saisit ensuite chaque valeur une par une.

C - Génération Manuelle

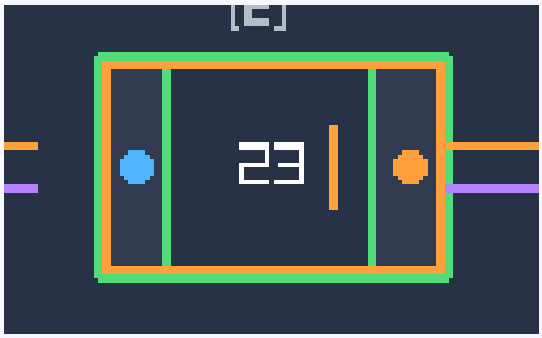
Python - Génération Manuelle



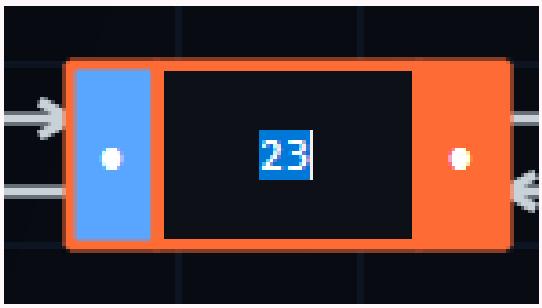
3.7.3 Saisie Directe dans la Liste

L'utilisateur peut saisir directement une valeur en cliquant sur un emplacement dans la liste. Un champ de saisie apparaît alors à cet endroit, permettant d'entrer la valeur souhaitée.

C - Saisie Nœud



Python - Saisie Nœud



3.8 Cadre TRI

Deux algorithmes de tri sont disponibles :
— **Tri à Bulles** : Recommandé pour les petites listes ($<$ 100 éléments)
— **Tri Rapide (QuickSort)** : Recommandé pour les grandes listes ($>$ 100 éléments)

3.8.1 Tri à Bulles (Bubble Sort)

C

```c
void TriBulle(ListeSimple *l) {
  if (!l->head) return;
  bool swapped;
  do {
    swapped = false;
    NodeSimple *c = l->head;
    while (c->next) {
      if (strcmp(c->data, c->next->data) > 0) {
        char temp[32]; strcpy(temp, c->data);
        strcpy(c->data, c->next->data);
        strcpy(c->next->data, temp);
        swapped = true;
      }
      c = c->next;
    }
  } while (swapped);
}
```

Python

```python
def tri_bulle(self):
  if not self.head: return
  swapped = True
  while swapped:
    swapped = False
    c = self.head
    while c.next:
      if c.data > c.next.data:
        c.data, c.next.data = c.next.data, c.data
        swapped = True
      c = c.next
```

3.8.2 Tri Rapide (Quick Sort)

C

```c
NodeSimple* Partition(NodeSimple *head, NodeSimple *end) {
  NodeSimple *pivot = end, *prev = NULL, *cur = head;
  while (cur != pivot) {
    if (strcmp(cur->data, pivot->data) < 0) {
      if (!prev) head = cur;
      prev = cur;
    } else {
      if (prev) prev->next = cur->next;
      NodeSimple *tmp = cur->next;
      cur->next = NULL; pivot->next = cur;
      pivot = cur; cur = tmp; continue;
    }
    cur = cur->next;
  }
  return pivot;
}
```

Python

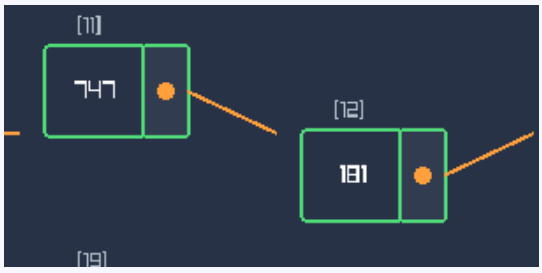
```python
def tri_rapide(self):
  if not self.head: return
  arr = []
  c = self.head
  while c: arr.append(c.data); c = c.next
  arr.sort()  # Python's Timsort (optimized)
  c, i = self.head, 0
  while c: c.data = arr[i]; c, i = c.next, i+1
```

Animation du Tri : Le processus de tri est entièrement **animé en temps réel**. Les éléments en cours de comparaison sont mis en surbrillance, et lors d'un échange, les nœuds concernés changent visuellement de position avec une animation fluide. Cela permet de visualiser concrètement le fonctionnement de chaque algorithme : les "bulles" qui remontent pour le tri à bulles, ou les partitions qui se forment pour le tri rapide.

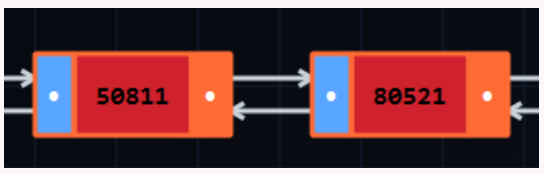
Barre de Progression : Une barre de progression néon indique l'avancement du tri (pourcentage de comparaisons effectuées). Le **temps écoulé** est également affiché en temps réel.

Bouton Stop : Un bouton **Arrêter** permet d'interrompre le tri en cours à tout moment, utile pour les très grandes listes où le tri peut prendre plusieurs secondes.

C - Animation Tri



Python - Animation Tri



3.9 Bouton VIDER

Le bouton **Vider** supprime tous les éléments de la liste en une seule action, réinitialisant la liste à vide.

Chapitre 4

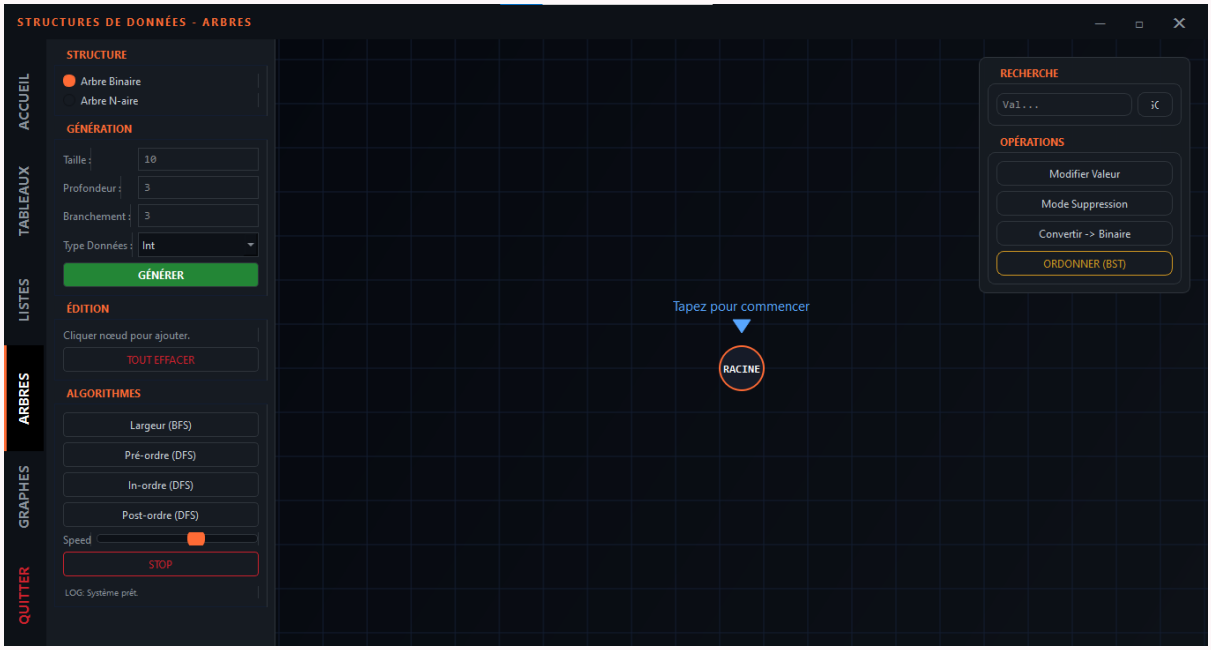
Module Arbres

4.1 Interface Générale et Effet Map

L'interface du module Arbres offre une zone de visualisation interactive de type "map" permettant de naviguer librement dans l'arbre. Cette zone occupe la majeure partie de l'écran et affiche une grille dynamique en arrière-plan qui se déplace avec la caméra, créant un effet de profondeur visuelle. L'utilisateur peut zoomer sur l'arbre en utilisant la molette de la souris, avec un facteur de zoom allant de 30% à 200%. Le zoom est centré sur la position du curseur, permettant un contrôle précis de la zone à agrandir. Pour se déplacer dans la scène, il suffit de maintenir le clic droit enfoncé et de faire glisser la souris dans la direction souhaitée. Le clic gauche sur une zone vide permet également de déplacer la vue. Cette navigation intuitive est essentielle pour explorer des arbres de grande taille qui dépassent les dimensions de l'écran.

C - Interface

Python - Interface

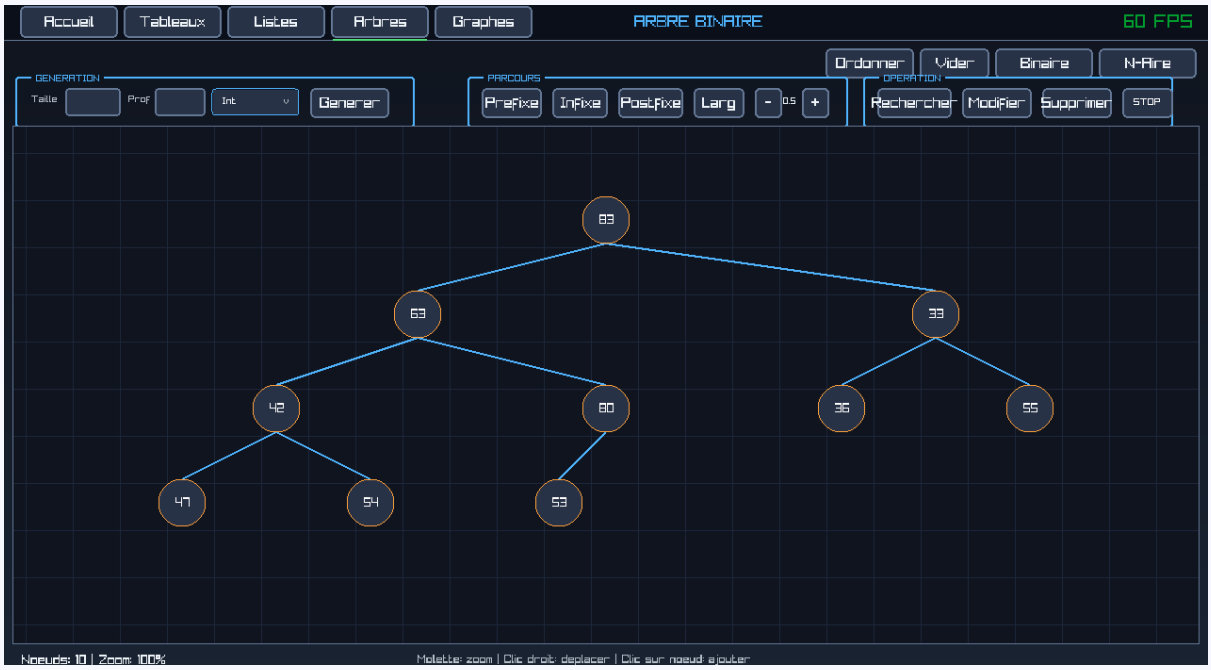


4.2 Modes Binaire et N-aire

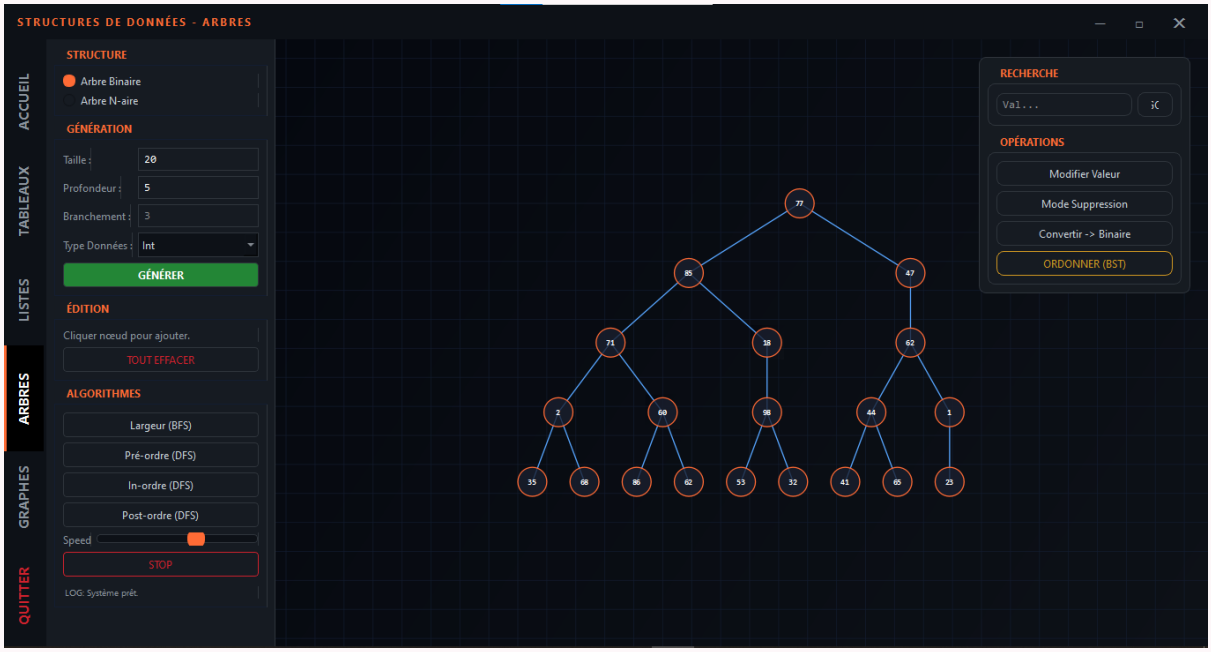
4.2.1 Mode Binaire

En mode binaire, chaque nœud de l'arbre peut avoir au maximum deux enfants : un enfant gauche et un enfant droite. Ce mode est particulièrement adapté pour représenter des arbres binaires de recherche (BST), des tas binaires ou des arbres d'expression. L'interface affiche les boutons “Binaire” et “N-Aire” en haut à droite de l'écran. Lorsque le mode binaire est actif, le bouton correspondant est mis en surbrillance avec une couleur bleue néon. Le titre dynamique “ARBRE BINAIRE” s'affiche dans la barre de navigation. Le parcours In-ordre est disponible uniquement en mode binaire car il n'a de sens que pour les arbres binaires où il produit un ordre trié pour les BST.

C - Mode Binaire



Python - Mode Binaire

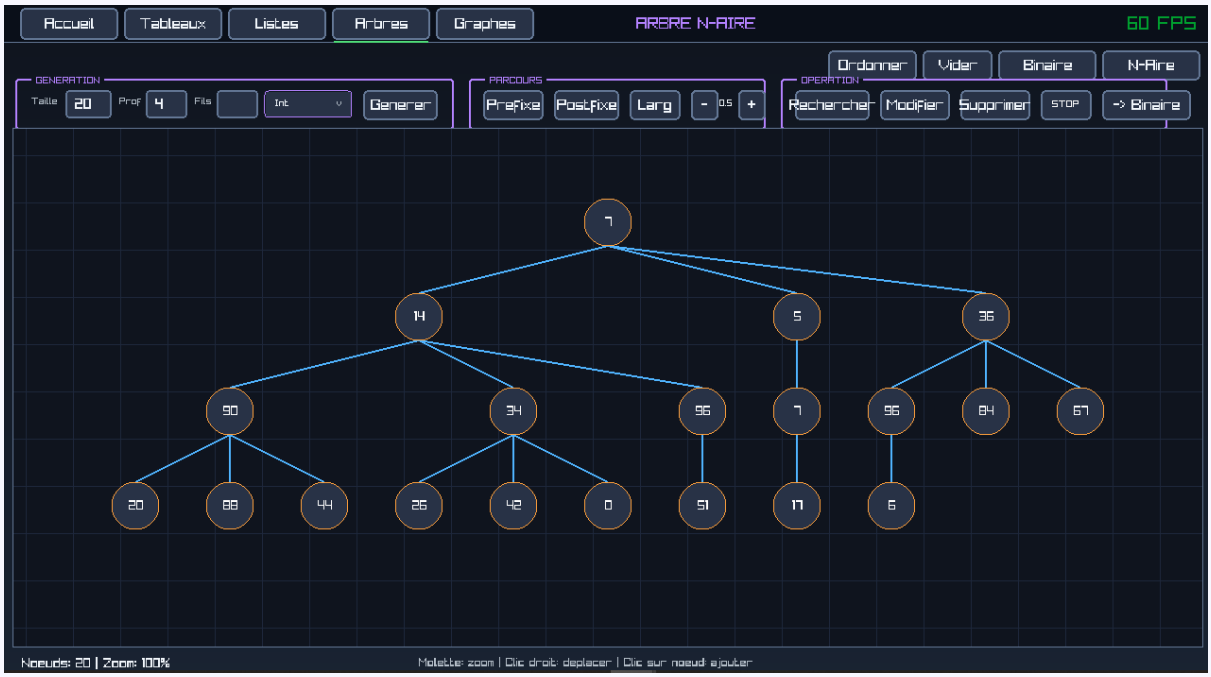


4.2.2 Mode N-aire

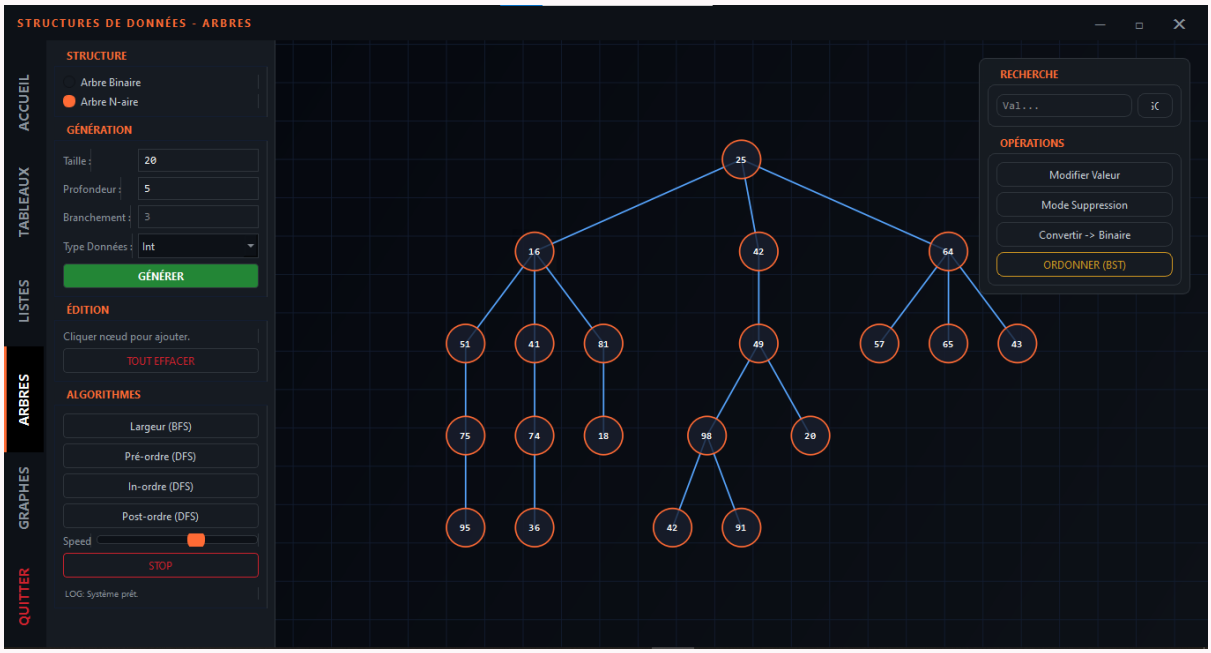
En mode N-aire, chaque nœud peut avoir un nombre variable d'enfants, limité par la constante MAX_CHILDREN fixée à 10. Ce mode offre plus de flexibilité pour représenter des structures hiérarchiques comme des organigrammes, des systèmes de fichiers ou des menus arborescents. Lorsque l'utilisateur bascule en mode N-aire, le titre change pour "ARBRE N-AIRE" avec une couleur violette, et un champ de saisie supplémentaire "Fils" apparaît dans le cadre de génération pour spécifier le nombre maximum d'enfants par nœud. Le bouton de conversion "→ Binaire" devient également visible, permettant de transformer l'arbre N-aire en arbre binaire via la représentation LCRS. À noter que le basculement entre les modes vide

l'arbre actuel pour éviter les incohérences structurelles.

C - Mode N-aire



Python - Mode N-aire



4.3 Structures de Données

C

```c
#define MAX_CHILDREN 10
typedef struct NoeudGenerique {
  char data[32];
  struct NoeudGenerique *children[MAX_CHILDREN];
  int child_count;
  float x, y, subtree_width;
} NoeudGenerique;

typedef struct { NoeudGenerique *racine; int taille; } ArbreGenerique;
```

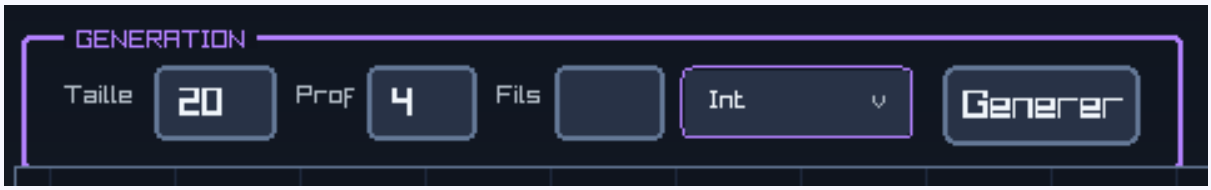
Python

```python
class Node:
  def __init__(self, value):
    self.value = str(value)
    self.children = []
    self.x, self.y = 0.0, 0.0

  def add_child(self, child):
    self.children.append(child)
```

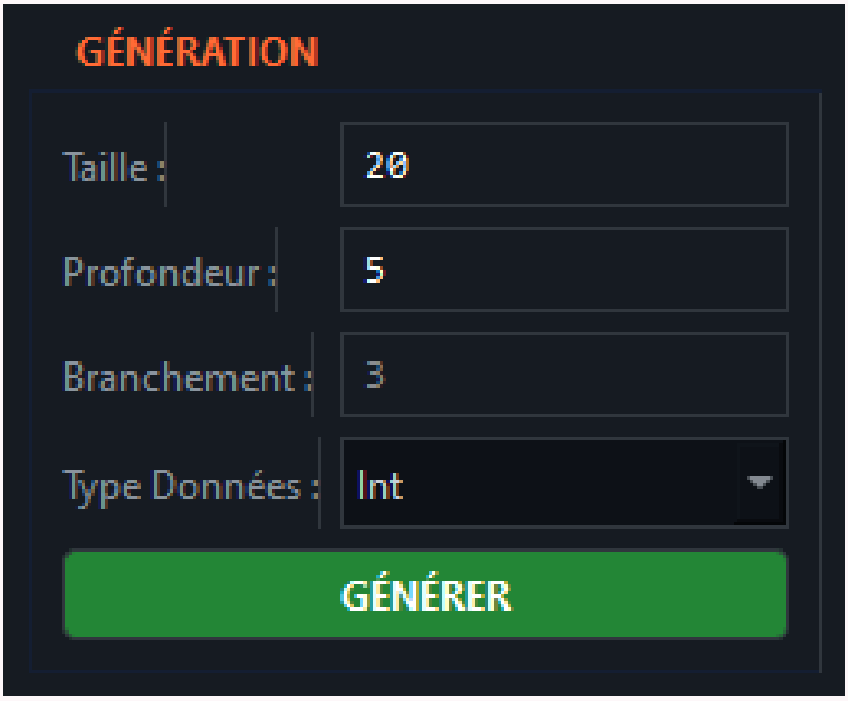
4.4 Génération Aléatoire

La génération aléatoire permet de créer rapidement un arbre complet avec des valeurs générées automatiquement. Le cadre “GENERATION” contient les champs suivants : **Taille** pour spécifier le nombre exact de nœuds souhaités (l’algorithme garantit ce nombre), **Prof** pour définir la profondeur maximale de l’arbre, et un menu déroulant pour choisir le type de données (Int, Float, Char ou String). En mode N-aire, un champ supplémentaire **Fils** permet de limiter le nombre maximum d’enfants par nœud. L’algorithme de génération utilise une approche par “candidats” : il maintient une liste de tous les emplacements disponibles où un nouveau nœud peut être attaché, puis en choisit un aléatoirement à chaque itération. Si les contraintes de profondeur empêchent d’atteindre le nombre demandé, un mécanisme de secours parcourt l’arbre en largeur pour trouver des emplacements disponibles, ignorant temporairement la limite de profondeur.

C - Cadre Génération



Python - Cadre Génération

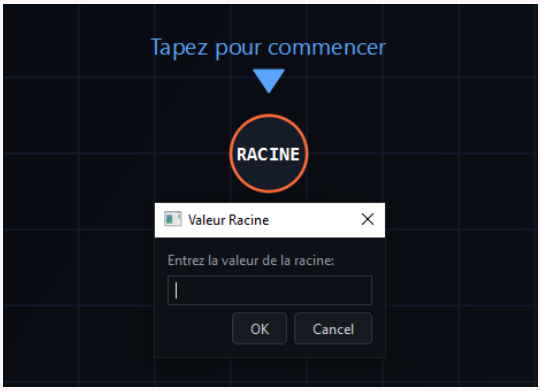


4.5 Génération Manuelle

La génération manuelle offre un contrôle total sur la construction de l'arbre. Au démarrage ou après avoir vidé l'arbre, un nœud spécial "RACINE" s'affiche au centre de la zone de visualisation, accompagné d'une flèche animée qui pointe vers lui et du message "Cliquez pour commencer". L'utilisateur doit cliquer sur ce nœud pour ouvrir une fenêtre popup lui demandant de saisir la valeur de la racine. Une fois la racine créée, l'arbre devient interactif : **il suffit de cliquer sur n'importe quel nœud existant** pour ouvrir le popup d'ajout et créer un nouveau nœud enfant. Cette approche intuitive permet de construire l'arbre nœud par nœud, en visualisant immédiatement le résultat de chaque ajout. La popup affiche un champ de saisie de texte avec un curseur clignotant, et les boutons d'action varient selon le mode (binaire ou N-aire) et le contexte (création de racine ou ajout d'enfant).

C - Création Racine

Python - Création Racine



4.6 Différence d'Ajout entre Binaire et N-aire

La différence majeure entre les deux modes se manifeste lors de l'ajout de nœuds enfants. En **mode binaire**, lorsque l'utilisateur clique sur un nœud parent, la popup affiche trois boutons : "GAUCHE", "DROITE" et "ANNULER". L'utilisateur doit explicitement choisir la position (gauche ou droite) où le nouveau nœud sera inséré. Les boutons correspondant aux positions déjà occupées sont désactivés (grisés) pour éviter l'écrasement de sous-arbres existants. Cette distinction gauche/droite est fondamentale pour les arbres binaires de recherche où la position détermine la relation d'ordre.

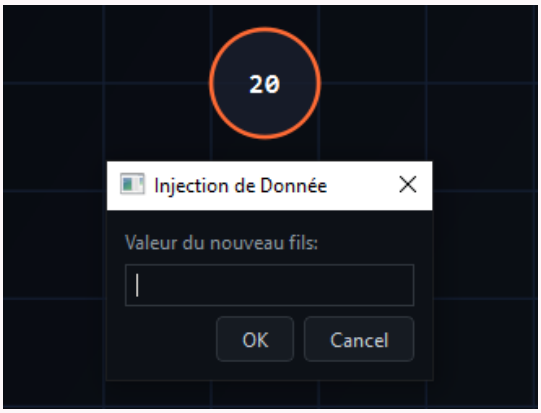
En **mode N-aire**, la popup est simplifiée et n'affiche que deux boutons : "AJOUTER" et "ANNULER". Chaque nouveau nœud est automatiquement ajouté comme dernier enfant du parent sélectionné, sans choix de position. Cette approche séquentielle est cohérente avec la nature des arbres N-aires où les enfants forment une liste ordonnée sans notion de "gauche" ou "droite". Le système empêche l'ajout si le nœud parent a déjà atteint la limite de MAX_CHILDREN enfants.

C - Popup Binaire

C - Popup N-aire

Python - Popup Binaire

Python - Popup N-aire



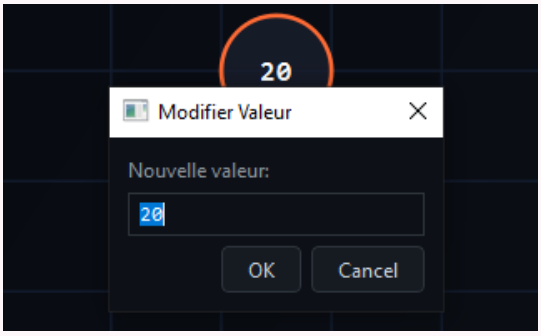
4.7 Modification de Nœuds

La modification permet de changer la valeur d'un nœud existant sans affecter sa position dans l'arbre ni ses enfants. Pour activer cette fonctionnalité, l'utilisateur clique sur le bouton "Modifier" dans le cadre "OPERATION". Le bouton s'illumine alors avec une couleur orange vif, indiquant que le mode modification est actif. Dans ce mode, cliquer sur n'importe quel nœud de l'arbre ouvre une popup intitulée "MODIFIER NOEUD" avec un champ de saisie pré-rempli contenant la valeur actuelle du nœud. L'utilisateur peut éditer cette valeur puis confirmer avec le bouton "MODIFIER" ou la touche Entrée. Le mode modification reste actif jusqu'à ce que l'utilisateur clique à nouveau sur le bouton pour le désactiver, ou jusqu'à ce qu'il active un autre

mode (suppression ou recherche). Cette fonctionnalité est particulièrement utile pour corriger des erreurs de saisie ou pour réorganiser les valeurs d'un arbre de recherche.

C - Modification

Python - Modification



4.8 Suppression de Nœuds

La suppression fonctionne de manière similaire à la modification. L'utilisateur active d'abord le mode suppression en cliquant sur le bouton "Supprimer" dans le cadre "OPERATION". Le bouton devient rouge vif pour signaler que le mode est actif et que les prochains clics sur des nœuds entraîneront leur suppression. Lorsqu'un nœud est cliqué en mode suppression, celui-ci ainsi que **tout son sous-arbre** (tous ses descendants) sont définitivement supprimés. En mode binaire, la suppression préserve la distinction gauche/droite : si le fils gauche est supprimé, l'emplacement reste vide et le fils droit conserve sa position. En mode N-aire, la liste des enfants est compactée : les enfants suivants "remontent" pour combler le trou créé par la suppression. Si la racine elle-même est supprimée, l'arbre entier est détruit et l'interface revient à l'état initial avec le nœud "RACINE" cliquable. Le mode suppression se désactive automatiquement lorsque l'arbre devient vide.

C

```c
void DetruireArbre(NoeudGenerique *node) {
  if (!node) return;
  for (int i = 0; i < node->child_count; i++)
    DetruireArbre(node->children[i]);
  free(node);
}
```

Python

```python
def delete_node(self, node):
  if node is None:
    return
  # Supprimer recursivement tous les enfants
  for child in node.children:
    self.delete_node(child)
  node.children.clear()
  # Retirer le noeud de son parent
  if node.parent:
    node.parent.children.remove(node)
```

4.9 Recherche

La fonctionnalité de recherche permet de localiser visuellement un nœud dans l'arbre en fonction de sa valeur. L'utilisateur clique sur le bouton "Rechercher" pour ouvrir une popup avec un champ de saisie où il entre la valeur recherchée. Après confirmation, une animation de recherche démarre : les nœuds sont visités un par un dans l'ordre du parcours, chaque nœud visité étant temporairement mis en surbrillance verte. L'animation suit le chemin depuis la racine jusqu'au nœud cible (s'il existe), permettant à l'utilisateur de visualiser le processus de recherche. La vitesse de l'animation peut être ajustée avec les boutons - et + du contrôle de vitesse. À la fin de l'animation, une popup de résultat s'affiche indiquant si la valeur "EST PRESENTE" ou "N'EST PAS PRESENTE" dans l'arbre. Si la valeur est trouvée, le nœud correspondant reste mis en surbrillance pour le repérer facilement.

C

```c
static bool CollecterRecherche(NoeudGenerique *node, const char *valeur
    ,
                               NoeudGenerique **result, int *idx) {
  if (!node) return false;
  result[(*idx)++] = node;  // Ajouter au chemin
  if (strcmp(node->data, valeur) == 0) return true;
  for (int i = 0; i < node->child_count; i++) {
    if (CollecterRecherche(node->children[i], valeur, result, idx))
      return true;
  }
  return false;
}
```

Python

```python
def search(self, node, value, path=[]):
  if node is None:
    return None, path
  path.append(node)  # Ajouter au chemin
  if node.value == value:
    return node, path
  for child in node.children:
    result, path = self.search(child, value, path)
    if result:
      return result, path
  return None, path
```

4.10 Parcours d'Arbres

Quatre types de parcours sont disponibles dans le cadre "PARCOURS", chacun représentant une stratégie différente pour visiter tous les nœuds de l'arbre :

— **Préfixe (Pré-ordre)** : Visite le nœud courant, puis ses enfants de gauche à droite. Ordre : Racine → Gauche → Droite. Utile pour copier un arbre ou obtenir une expression préfixée.

— **Infixe (In-ordre)** : Visite le fils gauche, puis le nœud courant, puis le fils droit. Ordre : Gauche → Racine → Droite. *Disponible uniquement en mode binaire.* Produit un ordre trié pour les arbres binaires de recherche.

— **Postfixe (Post-ordre)** : Visite tous les enfants avant le nœud courant. Ordre : Gauche → Droite → Racine. Utile pour la libération de mémoire ou l'évaluation d'expressions.
— **Largeur (BFS)** : Visite les nœuds niveau par niveau, de la racine vers les feuilles. Utile pour trouver le chemin le plus court ou explorer un arbre par niveaux.

Lors d'un parcours, une animation montre chaque nœud visité avec une surbrillance verte progressive. Les contrôles de vitesse (-/+) permettent d'accélérer ou ralentir l'animation. Le bouton "STOP" arrête immédiatement le parcours en cours. À la fin du parcours, une popup affiche la séquence complète des valeurs visitées, séparées par des flèches (ex : "5 → 3 → 8 → 2 → 4").

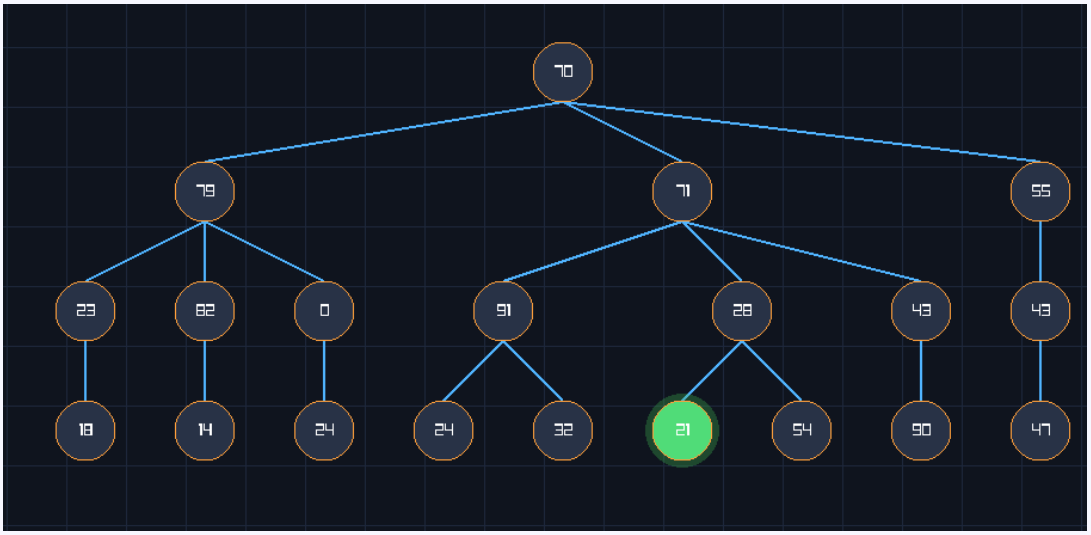
C

```c
// Pre-ordre: Racine -> Gauche -> Droite
void CollecterPreOrdre(NoeudGenerique *node, NoeudGenerique **res, int *idx) {
  if (!node) return;
  res[(*idx)++] = node;
  for (int i = 0; i < node->child_count; i++)
    CollecterPreOrdre(node->children[i], res, idx);
}
// In-ordre (binaire): Gauche -> Racine -> Droite
void CollecterInOrdre(NoeudGenerique *node, NoeudGenerique **res, int *idx) {
  if (!node) return;
  if (node->child_count > 0) CollecterInOrdre(node->children[0], res, idx);
  res[(*idx)++] = node;
  if (node->child_count > 1) CollecterInOrdre(node->children[1], res, idx);
}
// Post-ordre: Gauche -> Droite -> Racine
void CollecterPostOrdre(NoeudGenerique *node, NoeudGenerique **res, int *idx) {
  if (!node) return;
  for (int i = 0; i < node->child_count; i++)
    CollecterPostOrdre(node->children[i], res, idx);
  res[(*idx)++] = node;
}
// Largeur (BFS): Niveau par niveau
void CollecterLargeur(NoeudGenerique *root, NoeudGenerique **res, int *idx) {
  if (!root) return;
  NoeudGenerique *queue[500]; int front = 0, rear = 0;
  queue[rear++] = root;
  while (front < rear) {
    NoeudGenerique *node = queue[front++];
    res[(*idx)++] = node;
    for (int i = 0; i < node->child_count; i++)
      if (node->children[i]) queue[rear++] = node->children[i];
  }
}
```

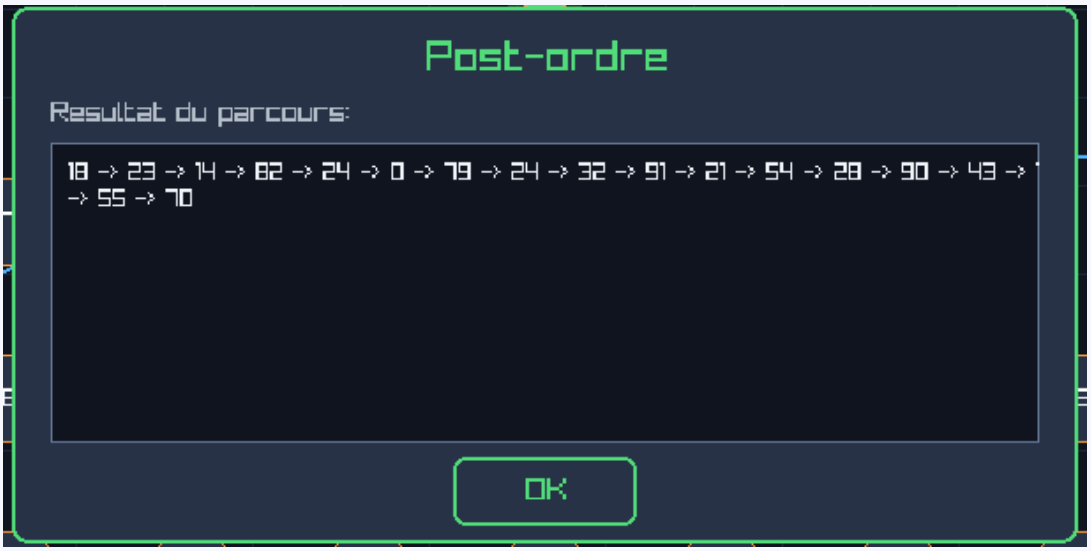
Python

```python
# Pre-ordre: Racine -> Enfants
def preorder(self, node, result=[]):
  if node: result.append(node)
  for child in node.children:
    self.preorder(child, result)
  return result

# In-ordre (binaire): Gauche -> Racine -> Droite
def inorder(self, node, result=[]):
  if len(node.children) > 0: self.inorder(node.children[0], result)
  result.append(node)
  if len(node.children) > 1: self.inorder(node.children[1], result)
  return result

# Post-ordre: Enfants -> Racine
def postorder(self, node, result=[]):
  for child in node.children:
    self.postorder(child, result)
  result.append(node)
  return result

# Largeur (BFS): Niveau par niveau
def bfs(self, root):
  if not root: return []
  result, queue = [], [root]
  while queue:
    node = queue.pop(0)
    result.append(node)
    queue.extend(node.children)
  return result
```

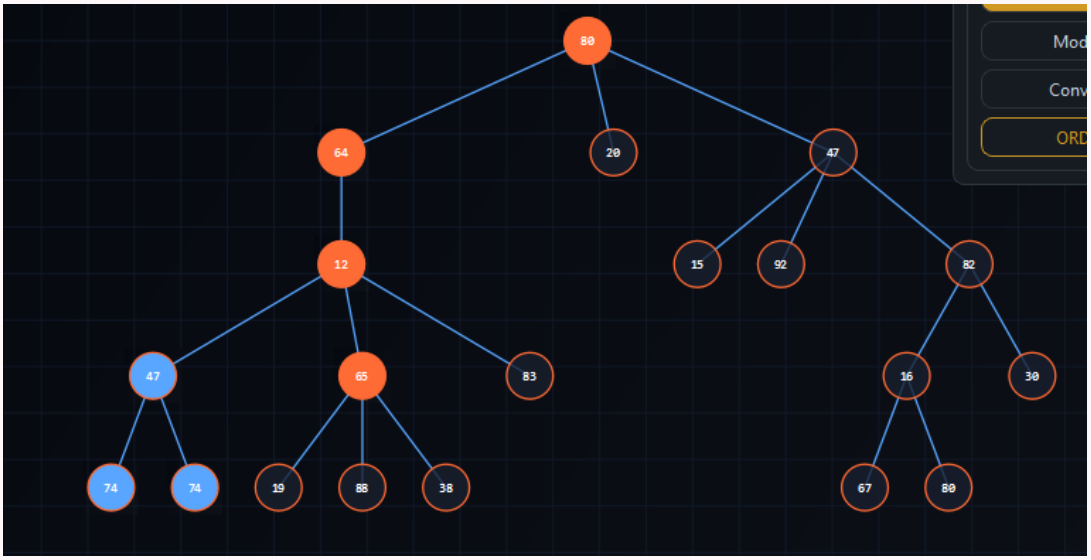
C - Animation Parcours



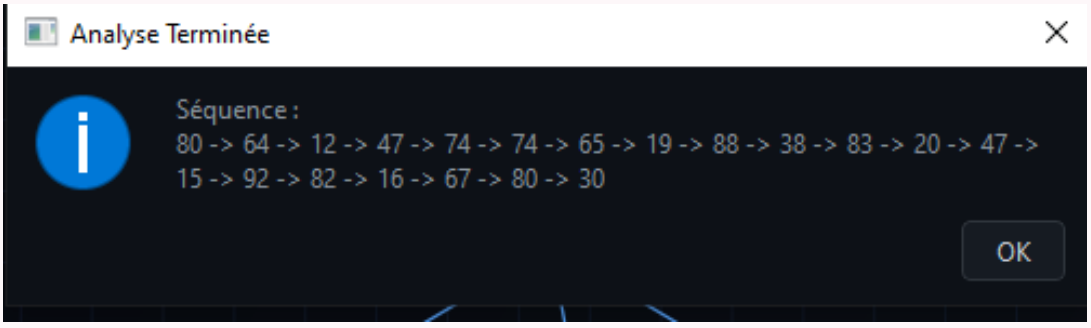
C - Résultat Parcours



Python - Animation Parcours



Python - Résultat Parcours



4.11 Bouton Ordonner (BST)

Le bouton “Ordonner” permet de réorganiser automatiquement l’arbre actuel en un arbre binaire de recherche (BST) équilibré. Cette opération collecte toutes les valeurs des nœuds de l’arbre, les trie selon le type de données (comparaison numérique pour Int/Float, alphabétique pour Char/String), puis reconstruit un arbre binaire parfaitement équilibré où chaque nœud est correctement placé selon les règles du BST (fils gauche $<$ parent $<$ fils droit). Le résultat est un arbre avec une hauteur minimale, optimisant les temps de recherche à O(log n). Ce bouton est visible uniquement lorsqu’un arbre existe et force automatiquement le passage en mode binaire puisqu’un BST est par définition un arbre binaire. Un message de confirmation “Arbre reordonne (BST equilibre)” s’affiche après l’opération.

4.12 Bouton Vider

Le bouton “Vider” situé en haut à droite de l’interface permet de supprimer intégralement l’arbre actuel et de revenir à l’état initial. Lorsque l’utilisateur clique sur ce bouton, tous les nœuds sont libérés de la mémoire, le compteur de taille revient à zéro, et le nœud “RACINE” cliquable réapparaît au centre de la zone de visualisation avec le message “Cliquez pour commencer”. Cette fonctionnalité est utile pour recommencer la construction d’un arbre depuis zéro ou pour libérer la mémoire avant de générer un nouvel arbre. Les modes actifs (modification, suppression) sont automatiquement désactivés et les animations en cours sont arrêtées.

4.13 Conversion N-aire vers Binaire (LCRS)

La conversion d’un arbre N-aire en arbre binaire utilise la représentation **LCRS** (Left-Child Right-Sibling), une technique élégante qui permet de représenter n’importe quel arbre N-aire sous forme binaire sans perte d’information. Le principe est le suivant :

— Le **premier enfant** d’un nœud N-aire devient son **enfant gauche** dans l’arbre binaire

— Les **frères suivants** deviennent une chaîne d’**enfants droits** successifs

Cette transformation préserve la structure hiérarchique tout en la rendant compatible avec les algorithmes d’arbres binaires. Le bouton “→ Binaire” n’est visible qu’en mode N-aire lorsqu’un arbre existe. Lors du clic, une animation de progression démarre avec une barre de progression et un pourcentage. L’arbre original “glisse” vers la gauche pendant que le pourcentage augmente. Une fois la conversion terminée, l’interface bascule automatiquement en mode binaire et affiche le nouvel arbre avec sa structure LCRS. Le message “Arbre converti en binaire (LCRS)” confirme la réussite de l’opération.

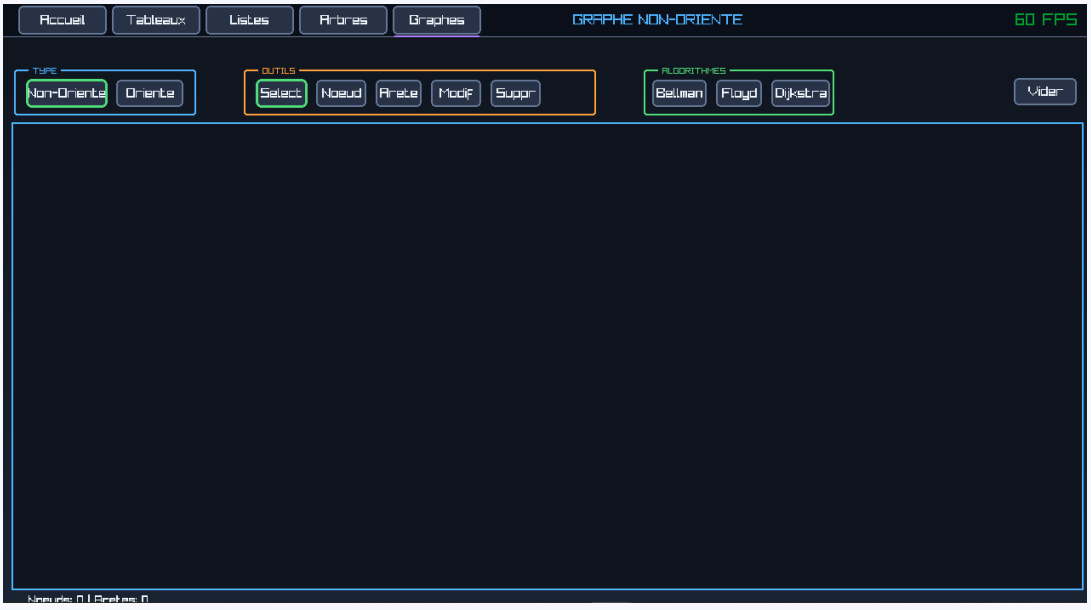
Chapitre 5

Module Graphes

5.1 Interface Générale

L'interface du module Graphes offre une zone de visualisation interactive où les nœuds peuvent être positionnés librement par glisser-déposer. La barre supérieure affiche le titre dynamique "GRAPHE ORIENTE" ou "GRAPHE NON-ORIENTE" selon le mode actif, avec une couleur distinctive (cyan pour orienté, violet pour non-orienté). Sur le côté gauche, un panneau d'outils permet de sélectionner l'action courante : **Sélection** (déplacer les nœuds), **Nœud** (ajouter un nœud), **Arête** (créer une connexion), **Supprimer** (retirer un nœud ou une arête), et **Modifier** (changer le label d'un nœud). Le panneau de droite contient les boutons des algorithmes de parcours et de chemin le plus court. Les nœuds sont représentés par des cercles avec leur label au centre, et les arêtes sont dessinées avec des lignes (avec flèches pour les graphes orientés) affichant le poids au milieu.

C - Interface



Python - Interface

5.2 Structures de Données

Le graphe est représenté par trois structures principales. **GraphNode** stocke l'identifiant unique, un label textuel personnalisable (jusqu'à 32 caractères), les coordonnées (x, y) pour l'affichage, et un booléen "active" pour la suppression logique. **GraphEdge** représente une arête avec les identifiants des nœuds source et destination, le poids (entier), et un booléen "active". La structure **Graph** contient les tableaux de nœuds et d'arêtes (limités par MAX_NODES=50 et MAX_EDGES=200), les compteurs, et un booléen "directed" indiquant si le graphe est orienté. Cette représentation par liste d'arêtes permet une manipulation flexible des graphes pondérés.

C

```c
typedef struct { int id; char label[32]; float x, y; bool active; }
    GraphNode;
typedef struct { int from, to, weight; bool active; } GraphEdge;
typedef struct {
  GraphNode nodes[MAX_NODES]; GraphEdge edges[MAX_EDGES];
  int node_count, edge_count; bool directed;
} Graph;
```

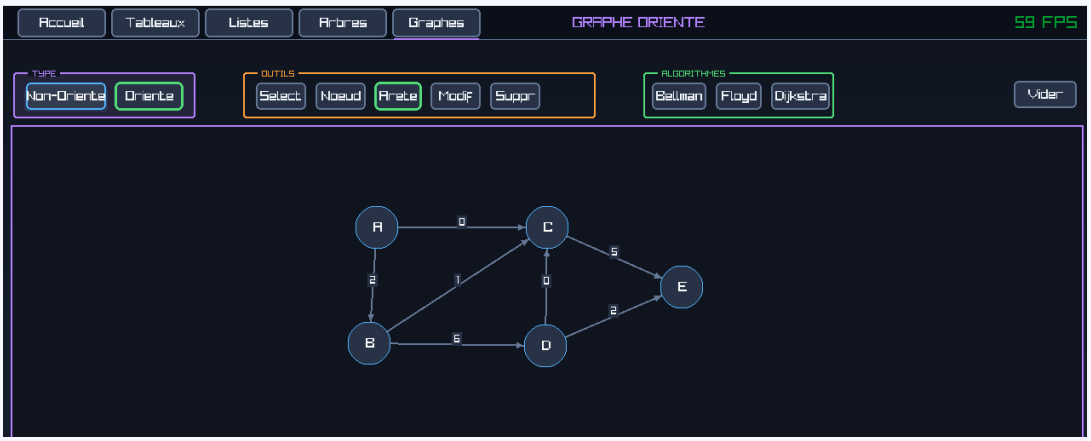
Python

```python
class GraphNode:
  def __init__(self, node_id, x=0, y=0):
    self.id, self.x, self.y = node_id, x, y
    self.neighbors = {}

class Graph:
  def __init__(self, directed=False):
    self.nodes, self.directed = {}, directed
```

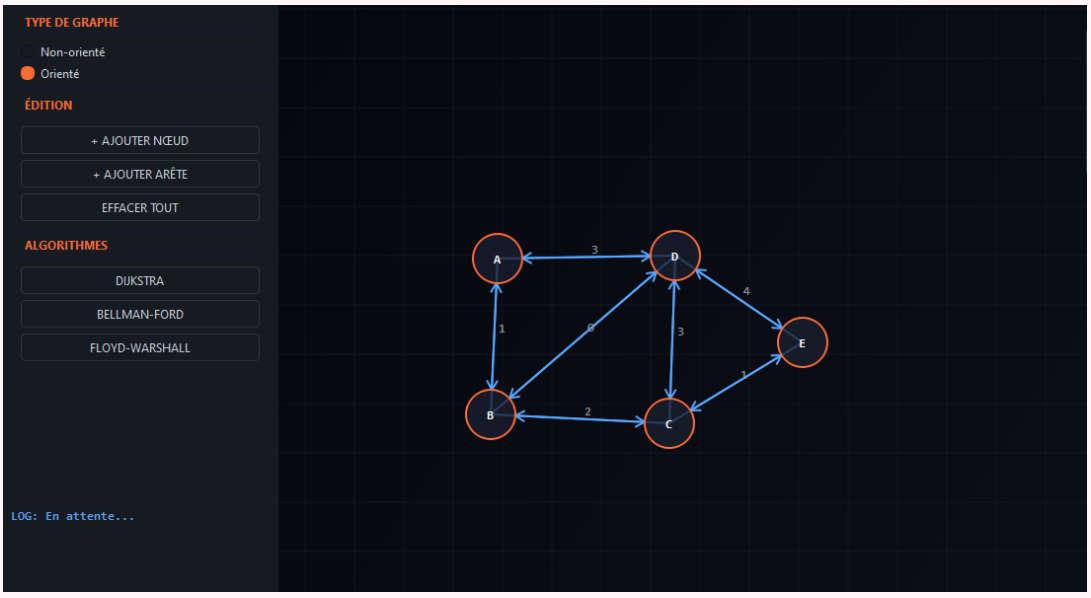
5.3 Modes Orienté et Non-Orienté

Le module supporte deux modes de graphes fondamentalement différents. En **mode orienté**, chaque arête a une direction : elle part d'un nœud source vers un nœud destination, visualisée par une flèche. Cela permet de modéliser des relations asymétriques (ex : réseau routier à sens unique, dépendances). En **mode non-orienté**, chaque arête représente une connexion bidirectionnelle sans notion de direction, visualisée par une simple ligne. Lors de l'ajout d'une arête en mode non-orienté, le système crée automatiquement l'arête inverse avec le même poids. Les boutons "Orienté" et "Non-Orienté" en haut de l'interface permettent de basculer entre les modes. Le changement de mode conserve le graphe existant.

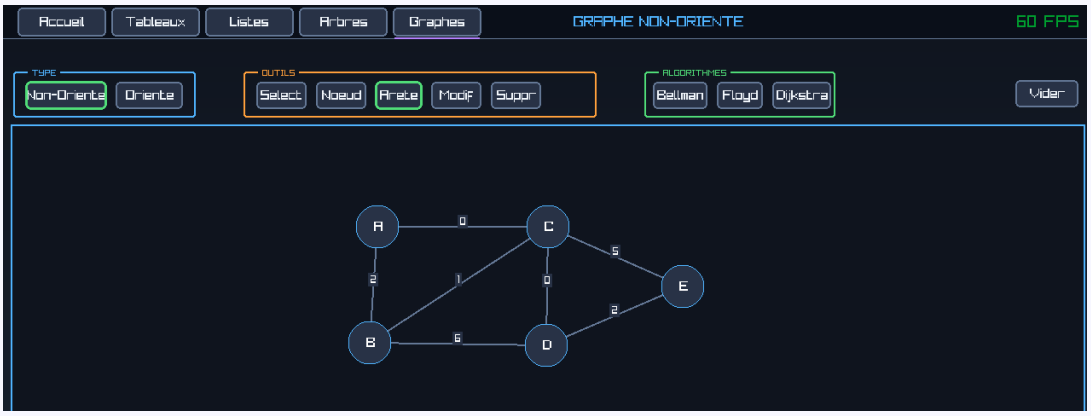
C - Mode Orienté



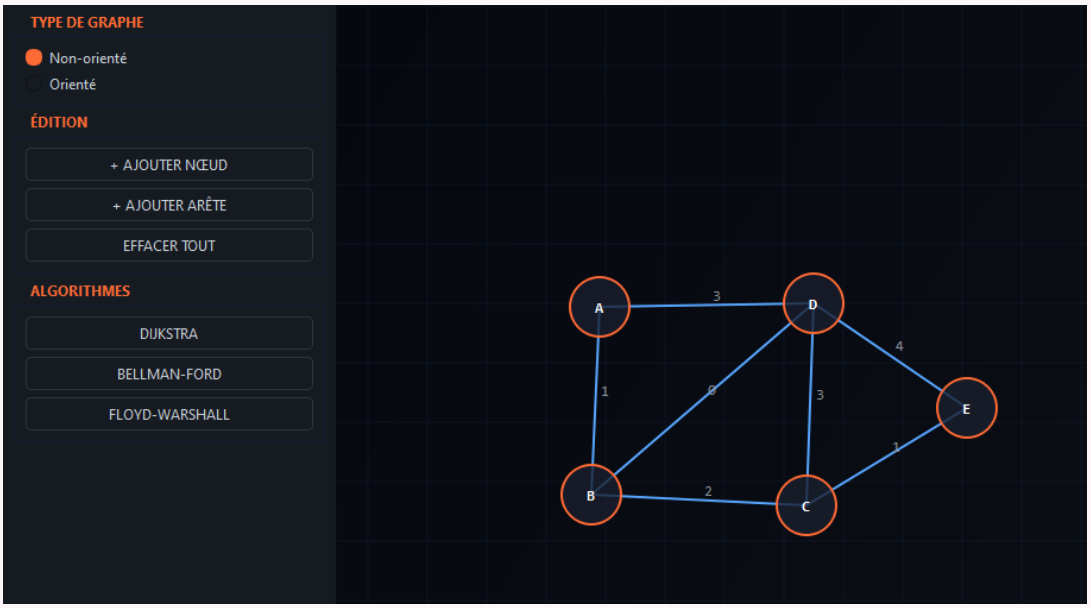
Python - Mode Orienté



C - Mode Non-Orienté



Python - Mode Non-Orienté



5.4 Ajout de Nœuds

Pour ajouter un nœud, l'utilisateur sélectionne l'outil "Nœud" dans le panneau de gauche, puis clique sur la zone de visualisation à l'emplacement souhaité. Une popup s'ouvre alors pour saisir le **label** du nœud (texte libre, contrairement à la limitation A-Z des anciens systèmes). Ce label peut être un nom, un numéro, ou tout texte descriptif. Après validation, le nœud apparaît à la position du clic avec son label centré. Chaque nœud reçoit un identifiant unique interne (entier auto-incrémenté) qui est indépendant du label affiché. Les nœuds peuvent ensuite être déplacés en utilisant l'outil Sélection et le glisser-déposer.

C

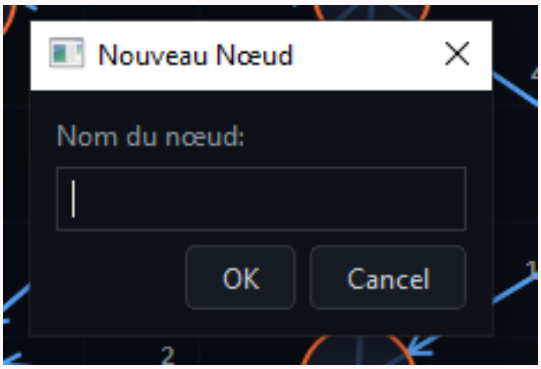
```c
int Graph_AddNode(Graph *g, float x, float y, const char *label) {
  if (g->node_count >= MAX_NODES) return -1;
  int id = g->node_count++;
  g->nodes[id].id = id; g->nodes[id].x = x; g->nodes[id].y = y;
  strncpy(g->nodes[id].label, label, 31); g->nodes[id].active = true;
  return id;
}
```

Python

```python
def add_node(self, node_id, x=0, y=0, label=""):
  if node_id not in self.nodes:
    self.nodes[node_id] = GraphNode(node_id, x, y)
```

C - Popup Ajout Noeud

Python - Popup Ajout Noeud



5.5 Ajout d'Arêtes

L'ajout d'arêtes diffère entre les versions C et Python. En **version C (Raylib)**, l'utilisateur sélectionne l'outil "Arête" puis **trace manuellement l'arête** en maintenant le clic enfoncé depuis le nœud source et en tirant jusqu'au nœud destination. Une ligne de prévisualisation suit la souris pendant le tracé. Lorsque l'utilisateur relâche le clic sur un nœud valide, une popup s'ouvre pour saisir le poids. En **version Python (PySide6)**, l'ajout se fait en deux clics : un premier clic sur le nœud source (qui se met en surbrillance), puis un second clic sur le nœud destination for ouvrir la popup de poids. Dans les deux cas, après validation du poids (valeur entière, positive ou négative), l'arête est créée et affichée avec son poids au milieu du segment. En mode non-orienté, l'arête inverse est automatiquement créée.

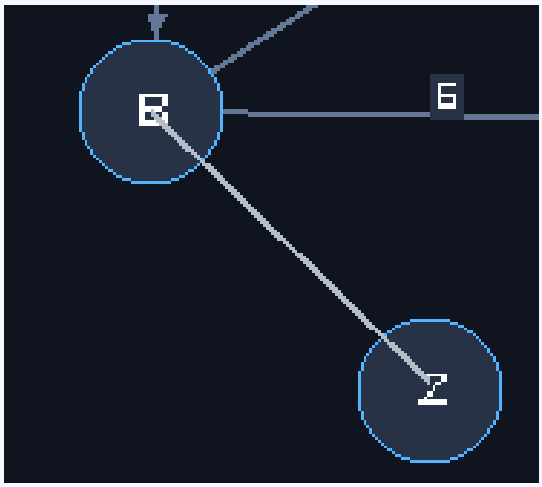
C

```c
bool Graph_AddEdge(Graph *g, int from, int to, int weight) {
  if (g->edge_count >= MAX_EDGES) return false;
  g->edges[g->edge_count].from = from;
  g->edges[g->edge_count].to = to;
  g->edges[g->edge_count].weight = weight;
  g->edges[g->edge_count].active = true;
  g->edge_count++;
  if (!g->directed) {
    g->edges[g->edge_count] = (GraphEdge){to, from, weight, true};
    g->edge_count++;
  }
  return true;
}
```

Python

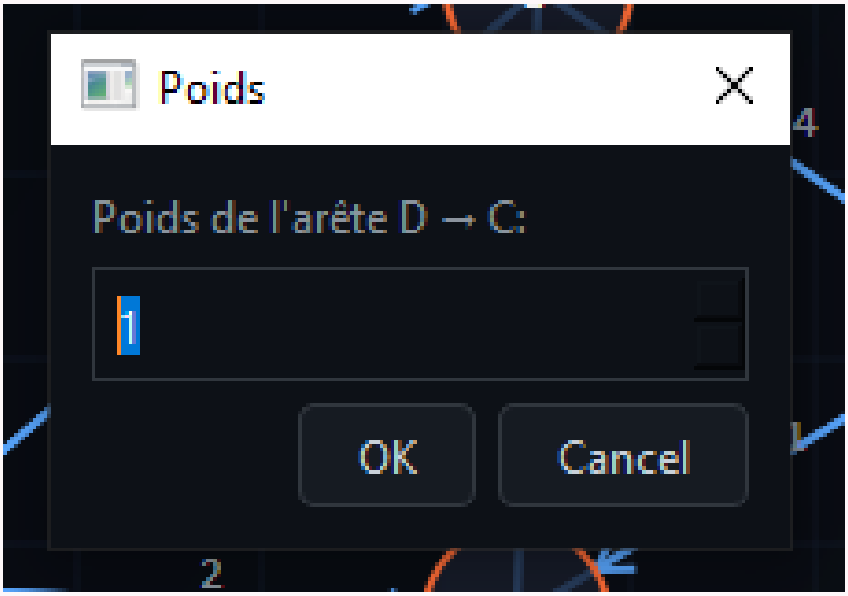
```python
def add_edge(self, from_id, to_id, weight=1):
  if from_id in self.nodes and to_id in self.nodes:
    self.nodes[from_id].neighbors[to_id] = weight
    if not self.directed:
```

C - Tracé Arête et Popup Poids (C)



Python - Tracé Arête et Popup Poids (C)

Python - Popup Poids Arête



5.6 Modification et Suppression

L'outil **Modifier** permet de changer le label d'un nœud existant. En sélectionnant cet outil puis en cliquant sur un nœud, une popup s'ouvre avec le label actuel pré-rempli. L'utilisateur peut éditer le texte et valider. L'outil **Supprimer** fonctionne de manière similaire : un clic sur un nœud le supprime ainsi que **toutes les arêtes connectées** (entrantes et sortantes). Un clic sur une arête (près de son centre/poids) ne supprime que cette arête. Le bouton "Vider" en haut de l'interface supprime l'intégralité du graphe et réinitialise l'état.

5.7 Algorithme de Dijkstra

L'algorithme de Dijkstra trouve le **chemin le plus court** entre deux nœuds dans un graphe pondéré avec des poids **positifs uniquement**. L'utilisateur saisit le nœud de départ et le nœud d'arrivée. L'algorithme utilise une approche gloutonne : il maintient pour chaque nœud la distance minimale connue depuis le départ, et explore toujours le nœud non visité ayant la plus petite distance. **Important :** le bouton Dijkstra est désactivé (grisé) si le graphe contient des poids négatifs.

C

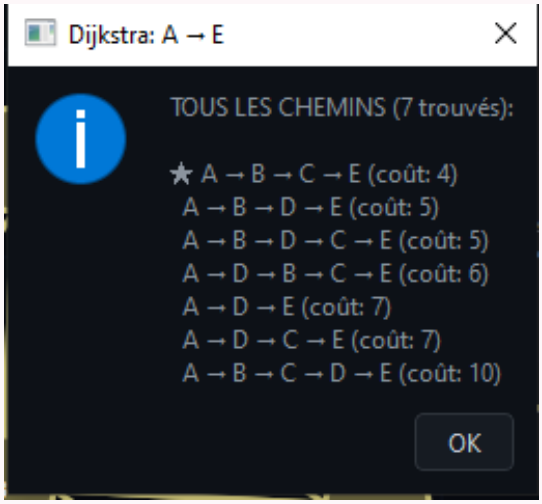
```c
int *Graph_Dijkstra(Graph *g, int start, int end, int *len, int *cost)
    {
  int dist[MAX_NODES], parent[MAX_NODES]; bool vis[MAX_NODES] = {0};
  for (int i = 0; i < MAX_NODES; i++) { dist[i] = INT_MAX; parent[i] =
      -1; }
  dist[start] = 0;
  for (int c = 0; c < g->node_count; c++) {
    int u = -1, min = INT_MAX;
    for (int v = 0; v < MAX_NODES; v++)
      if (!vis[v] && dist[v] < min) { min = dist[v]; u = v; }
    if (u == -1) break; vis[u] = true;
    for (int i = 0; i < g->edge_count; i++)
      if (g->edges[i].from == u) {
        int v = g->edges[i].to, w = g->edges[i].weight;
        if (dist[u] + w < dist[v]) { dist[v] = dist[u] + w; parent[v] =
            u; }
      }
  }
  *cost = dist[end];
  // Reconstruction du chemin via parent[]
}
```

Python

```python
def dijkstra(self, start_id, end_id):
  import heapq
  dist = {n: float('inf') for n in self.nodes}; dist[start_id] = 0
  parent = {n: None for n in self.nodes}; pq = [(0, start_id)]
  while pq:
    d, u = heapq.heappop(pq)
    if d > dist[u]: continue
    for v, w in self.nodes[u].neighbors.items():
      if dist[u] + w < dist[v]:
        dist[v] = dist[u] + w; parent[v] = u
        heapq.heappush(pq, (dist[v], v))
  path = []; cur = end_id
  while cur: path.append(cur); cur = parent[cur]
```

C - Resultat Dijkstra

Python - Resultat Dijkstra



5.8 Algorithme de Bellman-Ford

L'algorithme de Bellman-Ford résout le même problème que Dijkstra (chemin le plus court) mais il supporte les **poids négatifs**. Il fonctionne en effectuant V-1 passes de relaxation sur toutes les arêtes, où V est le nombre de nœuds. À chaque passe, pour chaque arête (u,v) de poids w, si dist[u] + w < dist[v], alors dist[v] est mis à jour. L'algorithme peut également détecter les cycles de poids négatif. La complexité est O(V×E).

C

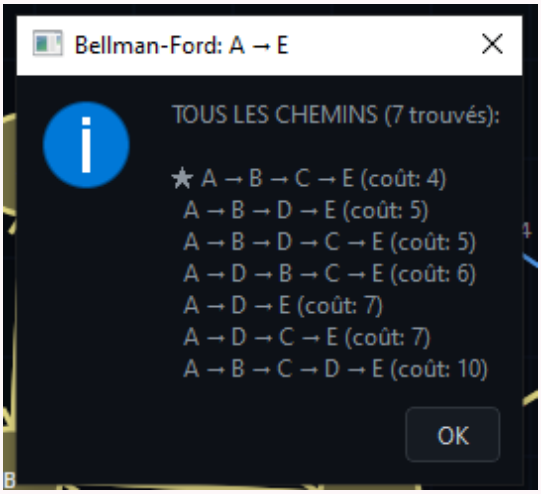
```c
int *Graph_BellmanFord(Graph *g, int start, int end, int *len, int *
    cost) {
  int dist[MAX_NODES], parent[MAX_NODES];
  for (int i = 0; i < MAX_NODES; i++) { dist[i] = INT_MAX; parent[i] =
      -1; }
  dist[start] = 0;
  for (int i = 0; i < g->node_count - 1; i++) {
    for (int j = 0; j < g->edge_count; j++) {
      int u = g->edges[j].from, v = g->edges[j].to, w = g->edges[j].
          weight;
      if (dist[u] != INT_MAX && dist[u] + w < dist[v]) {
        dist[v] = dist[u] + w; parent[v] = u;
      }
    }
  }
  *cost = dist[end];
  // Reconstruction du chemin
}
```

Python

```python
def bellman_ford(self, start_id, end_id):
  dist = {n: float('inf') for n in self.nodes}; dist[start_id] = 0
  parent = {n: None for n in self.nodes}
  for _ in range(len(self.nodes) - 1):
    for from_id, node in self.nodes.items():
      for to_id, w in node.neighbors.items():
        if dist[from_id] + w < dist[to_id]:
          dist[to_id] = dist[from_id] + w; parent[to_id] = from_id
  return self._reconstruct(parent, end_id), dist[end_id]
```

C - Resultat Bellman-Ford

Python - Resultat Bellman-Ford



5.9 Algorithme de Floyd-Warshall

L'algorithme de Floyd-Warshall calcule les **plus courts chemins entre toutes les paires** de nœuds en une seule exécution. Il construit une matrice de distances où l'entrée (i,j) contient le coût minimal pour aller de i à j. L'algorithme utilise la programmation dynamique avec trois boucles imbriquées : pour chaque nœud intermédiaire k, il vérifie si passer par k améliore le chemin entre i et j. La complexité est $O(V^3)$. L'utilisateur clique sur "Floyd-Warshall" sans saisir de nœuds spécifiques. Le résultat affiche une matrice des distances minimales entre toutes les paires de nœuds. Cet algorithme est particulièrement utile pour les graphes denses où l'on a besoin de connaître tous les chemins.

C

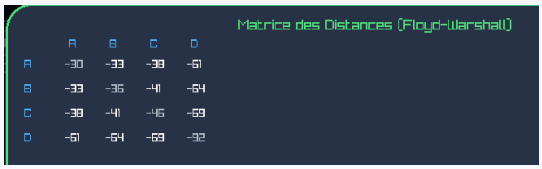
```c
FloydResult Graph_FloydWarshall(Graph *g) {
  FloydResult res; res.count = g->node_count;
  // Initialiser matrice avec INF
  for (int i = 0; i < MAX_NODES; i++)
    for (int j = 0; j < MAX_NODES; j++)
      res.costs[i][j] = (i == j) ? 0 : INT_MAX;
  // Ajouter les aretes
  for (int e = 0; e < g->edge_count; e++)
    res.costs[g->edges[e].from][g->edges[e].to] = g->edges[e].weight;
  // Relaxation
  for (int k = 0; k < g->node_count; k++)
    for (int i = 0; i < g->node_count; i++)
      for (int j = 0; j < g->node_count; j++)
        if (res.costs[i][k] + res.costs[k][j] < res.costs[i][j])
          res.costs[i][j] = res.costs[i][k] + res.costs[k][j];
  return res;
}
```

Python

```python
def floyd_warshall(self):
  ids = list(self.nodes.keys()); n = len(ids); INF = float('inf')
  dist = [[INF]*n for _ in range(n)]
  for i in range(n): dist[i][i] = 0
  for i, nid in enumerate(ids):
    for jid, w in self.nodes[nid].neighbors.items():
      j = ids.index(jid); dist[i][j] = w
  for k in range(n):
    for i in range(n):
      for j in range(n):
        if dist[i][k] + dist[k][j] < dist[i][j]:
          dist[i][j] = dist[i][k] + dist[k][j]
  return dist
```

Python - Resultat Floyd-Warshall

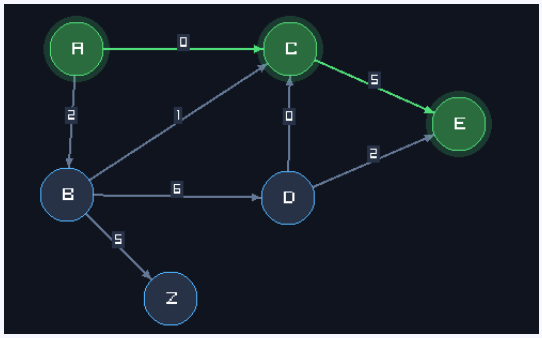
C - Resultat Floyd-Warshall



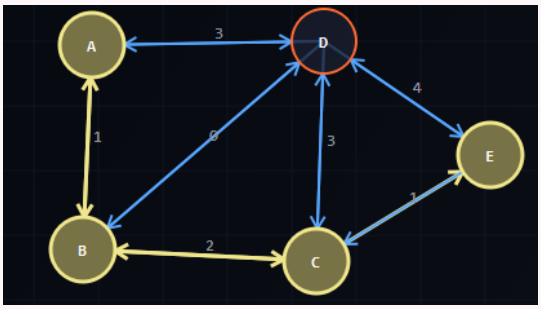
5.10 Affichage des Résultats

Pour tous les algorithmes de plus court chemin (Dijkstra, Bellman-Ford), le programme affiche **tous les chemins possibles** entre les deux nœuds sélectionnés dans une popup de résultat. Le **chemin le plus court** est mis en évidence en vert, tandis que les autres chemins sont affichés normalement. De plus, une **animation visuelle** parcourt le chemin optimal sur le graphe : les nœuds et arêtes du chemin sont surlignés successivement, permettant de visualiser clairement le trajet calculé. Le coût total du chemin le plus court est également affiché.

C - Animation Chemin



Python - Animation Chemin



Chapitre 6

Conclusion

Ce projet de fin de module a permis de concevoir et développer une application éducative complète dédiée à la visualisation et à la manipulation des structures de données fondamentales en informatique.

6.1 Bilan du Projet

L'application développée couvre les quatre grandes familles de structures de données :

— **Tableaux** : Génération, manipulation et comparaison visuelle des algorithmes de tri (Bubble Sort, Insertion Sort, Shell Sort, Quick Sort) avec mesure des performances en temps réel.

— **Listes Chaînées** : Implémentation complète des listes simplement et doublement chaînées avec opérations CRUD, recherche animée et tri visualisé.

— **Arbres** : Visualisation interactive des arbres binaires et N-aires avec parcours (Préfixe, Infixe, Postfixe), conversion entre types et algorithmes de recherche.

— **Graphes** : Manipulation de graphes orientés et non-orientés avec implémentation des algorithmes de plus court chemin (Dijkstra, Bellman-Ford, Floyd-Warshall).

6.2 Double Implémentation

Une particularité majeure de ce projet réside dans sa **double implémentation** :

— **Version C/Raylib** : Programmation système bas niveau avec gestion manuelle de la mémoire et rendu graphique performant.

— **Version Python/PySide6** : Développement orienté objet avec interface Qt moderne et architecture modulaire.

Cette approche comparative a permis d'appréhender les forces et contraintes de chaque paradigme de programmation.

6.3 Compétences Acquises

La réalisation de ce projet a permis de développer des compétences variées :

— Maîtrise approfondie des structures de données et de leurs algorithmes associés

— Programmation graphique et création d'interfaces utilisateur modernes

— Gestion de projet et organisation du code source

— Débogage et optimisation des performances

— Rédaction de documentation technique avec LaTeX

6.4 Perspectives

Plusieurs améliorations pourraient être envisagées :
— Ajout de nouvelles structures (Piles, Files, Tables de Hachage)
— Implémentation d'algorithmes supplémentaires (A*, Kruskal, Prim)
— Mode tutoriel interactif pour l'apprentissage
— Export des visualisations en format image ou vidéo

En conclusion, ce projet représente une expérience formatrice significative qui démontre que la maîtrise des structures de données constitue un pilier essentiel de la formation en informatique. L'approche visuelle et interactive adoptée facilite la compréhension de concepts parfois abstraits, confirmant ainsi l'intérêt pédagogique d'outils de ce type.