

# TP Algorithmique

David SALLÉ ([d4v1d.s4ll3@gmail.com](mailto:d4v1d.s4ll3@gmail.com))

Ce document est mis à disposition selon les termes de la licence

[Creative Commons BY-NC-SA 3.0](https://creativecommons.org/licenses/by-nc-sa/3.0/)



Version du document : v0.2  
Date : 17/01/2022

# Table des matières

<b>1 - Introduction.....</b>	<b>3</b>
<b>2 - Outils et organisation.....</b>	<b>4</b>
2.1 - Organisation.....	4
2.2 - Mini-projet IA.....	4
2.2.1 - Plateforme codingame.....	4
2.2.2 - Présentation de l'IDE.....	5
2.2.3 - Classement.....	6
2.2.4 - Participants.....	7
2.3 - IDE C++ CLion.....	8
2.3.1 - Installation compilateur C++.....	8
2.3.2 - Activation CLion.....	8
2.3.3 - Configuration CLion.....	9
2.3.4 - Premier projet : Hello, CNAM!.....	10
<b>3 - Mini-projet Broomstick Flyers.....</b>	<b>11</b>
3.1 - Présentation.....	11
3.2 - Code de départ.....	12
3.3 - Erreurs courantes.....	13
3.4 - IA.....	14
3.5 - Travail à faire et évaluation.....	15
<b>4 - Exercices structures de données.....</b>	<b>16</b>
4.1 - Broomstick flyers (tableaux, tri).....	16
4.2 - Jeu de la Bataille (pile, file).....	17
4.2.1 - Présentation.....	17
4.2.2 - Représentation des données.....	18
4.2.3 - Règles pour la simulation.....	18
4.2.4 - Travail à faire.....	21
4.3 - Nuage de mots (dictionnaire, tableau, tri).....	22
4.3.1 - Présentation.....	22
4.3.2 - Transformations des données.....	22
4.3.3 - Travail à faire.....	24
4.4 - Learderboard (arbres, récursivité).....	24
4.4.1 - Présentation.....	24
4.4.2 - Structure de données.....	24
4.4.3 - Fonctionnalités.....	26
4.4.4 - Visualisation avec Graphviz.....	27
4.4.5 - Travail à faire.....	27
4.5 - Navigation routière (graphes).....	28
4.5.1 - Présentation.....	28
4.5.2 - Représentation des données.....	28
4.5.3 - Travail à faire.....	29

# 1 - Introduction

Ce document contient les activités liées au cours **Systemes d'exploitation**.

Quand une commande devra être entrée dans un terminal, elle sera présentée comme ci-dessous :

```
$ ls
```

Les fichiers à télécharger depuis Moodle apparaîtront en couleur vert : **logisim-generic-2.7.1.jar**

A chaque fois que vous verrez apparaître ce pictogramme, c'est qu'il faudra joindre votre travail au compte rendu de TP :



**A rendre : ....**

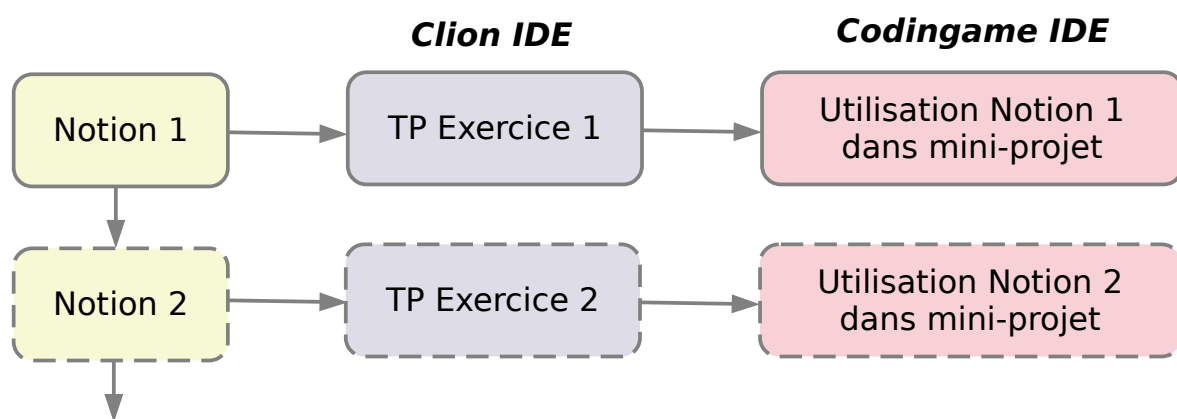
## 2 - Outils et organisation

### 2.1 - Organisation

Nous travaillerons tout au long de ces travaux pratiques avec 2 outils :

- **CLion** : un IDE C++ pour les exercices de TP sur les structures de données
- **Codingame** : une plateforme proposant des challenges de programmation/IA

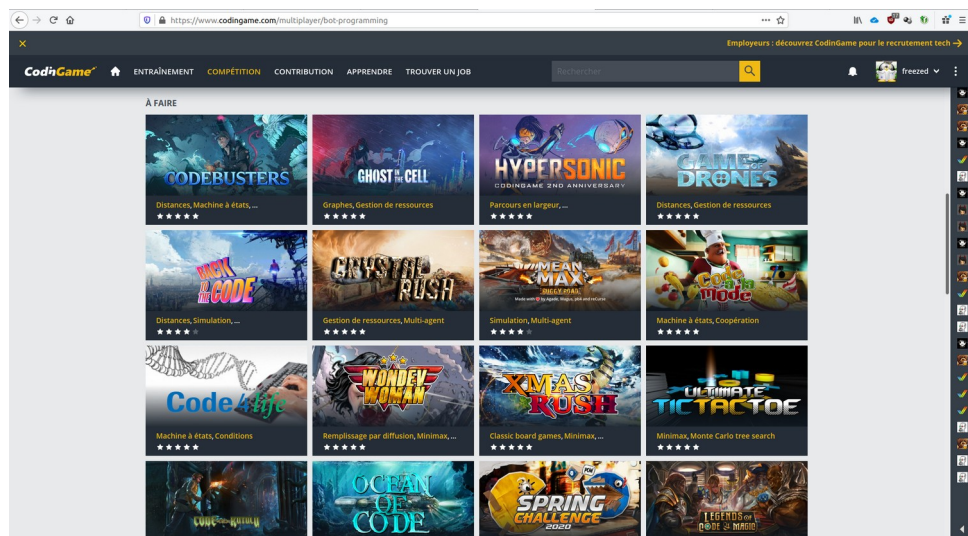
L'idée globalement sera de réinvestir quelques unes des notions algorithmiques/C++ étudiées (POO, vector...) dans une perspective IA sous forme de mini-projet "fil rouge".



### 2.2 - Mini-projet IA

#### 2.2.1 - Plateforme codingame

Le mini-projet IA sur lequel nous allons travailler est un challenge proposé par la plateforme **codingame** et accessible ici : <https://www.codingame.com/start>



C'est une plate-forme dédiée à la programmation informatique proposant aux participants différents jeux sous forme de casse-tête ou de compétition d'intelligence artificielle multi-joueurs. Des entreprises utilisent également cette plateforme pour recruter des développeurs.

Chaque joueur devra programmer un bot qui affrontera ensuite les bots des autres joueurs. La plate-forme se charge ensuite d'organiser les matchs et gérer le classement. Les programmes ne disposent que de 100ms par tour pour proposer une réponse.

### 2.2.2 - Présentation de l'IDE

La plate-forme intègre un IDE web pour coder directement en ligne dans votre navigateur sans avoir installer quoi que ce soit d'autre. Le résultat ainsi que le détails des règles sont également intégrés dans l'IDE.

The screenshot shows the CodingGame IDE interface with several annotations pointing to different parts of the UI:

- Menu**: Points to the left sidebar containing navigation links like RETOUR, FORUM, LAST BATTLES, HISTORIQUE, LEADERBOARD, and PARAMETRES.
- Résultat**: Points to the central game area showing a match between 'Freezed' and 'Bossgonagall'.
- Règles du challenge**: Points to the 'Récapitulatif des nouveautés' (Summary of new features) section below the game area.
- Zone pour coder**: Points to the Python 3 code editor on the right side of the interface.
- Résultat du match**: Points to the 'Sortie console' (Console output) section at the bottom left.
- Console d'informations et d'erreurs**: Points to the 'Informations de jeu' (Game information) section at the bottom left.
- Joueurs du match**: Points to the 'Joueurs' (Players) section at the bottom center, showing the avatars of 'Freezed' and 'Bossgonagall'.
- Soumet le code à la plateforme**: Points to the 'Actions' section at the bottom right, specifically the 'LANCER MON CODE' button.
- Test le code sans le soumettre**: Points to the 'TESTER DANS L'ARÈNE' button in the 'Actions' section.

### 2.2.3 - Classement

Une fois votre code soumis, la plate-forme organise des matchs contre les autres bots. Selon vos résultats votre classement sera ensuite mis à jour. Vous pouvez soumettre votre code autant de fois que vous le souhaitez.

https://www.codinggame.com/ide/puzzle/fantastic-bits

**Ligue Bronze**

Rechercher un ami, une école, un pays, ...

CodinGamer Langage Score Pays

Ceux meilleurs que le Boss seront promus en ligue Argent à 16 H 07 **CLASSEMENT COMPLET**

Rang	Nom	Langage	Score	Pays
360	TheGreenaz	C++	-3,05	Bulgarie
361	Jivraj_Grewal	Java	-3,07	Canada
362	Sheenbatori	C++	-3,08	France
363	Joueur sans nom	Java	-3,09	
364	Joueur sans nom	C++	-3,12	
365	tarahuma	C++	-3,14	Turquie
366	freezed	Java	-3,16	France
367	Vince31	C++	-3,17	France
368	Fedingo	Scala	-3,20	Italie
369	Arminas_Valtorius	C++	-3,25	Bulgarie

Les joueurs sont regroupés par niveaux/ligues :

#### NIVEAUX DE LIGUES



Bois



Bronze



Argent



Or



Légende

Division 2 à 1

## 2.2.4 - Participants

→ Créez un **compte** sur la plateforme en utilisant votre adresse électronique [prenom.nom@lecnam.net](mailto:prenom.nom@lecnam.net)

Choisir alors un **pseudo**, une **école** et une **entreprise**. Afin de pouvoir faciliter le classement chacun d'entre vous choisira la même école/entreprise.



→ [https://mensuel.framapad.org/p/codinggame\\_players\\_names-9rvk?lang=fr](https://mensuel.framapad.org/p/codinggame_players_names-9rvk?lang=fr)

<b>Nom</b>	<b>Prénom</b>	<b>Pseudo</b>
AUVRAY	Marc-Antoine	MarcAirlines
BEN MEZIANE	Manale	Mnlbs
BORDENAVE	Baptiste	Baptiste_Bordenave_Robotique_Pro_Of_Java
DELAMARRE	Paul	Paul_Dlm
DREAN	Arno	DArno
GABORIEAU	Florian	Floriancnam
GARNIER	Maxime	Maximecnam
GREGOIRE	Corentin	corentin_grg
GROS	Baptiste	Bapti92
JOHN MOUHAMAD	Subhane	subhanej
KHEDER	Ammar	Ammar99
KOESSLER	Théo	Theo_Koessler
LE GALL	Matthieu	Matt56
LESAGE	Max	max7984
MAINARD	Léo	Tiki
NGAKA	Djessy	
PAPET	Nathan	Nathan_Papet
QUEFFELEC	Timothée	CryptoManiganceur
RAMKALIA	Leroy	leroy971
RAVAUD	Damien	Damsitoo
RIGAULT	Enzo	enzo_rigl
ROLLAND	Gwendal	Tanki93_2A
SYOUD	Walid	
VAN DEN BOOM	Pierre	Piteur_le_Destructeur

## 2.3 - IDE C++ CLion

Nous utiliserons également en parallèle l'IDE C++ **CLion** de JetBrains. Il nous à la fois à faire les exercices hors mini-projet comme les blocs de code à intégrer et tester dans l'IDE de la plateforme codingame.

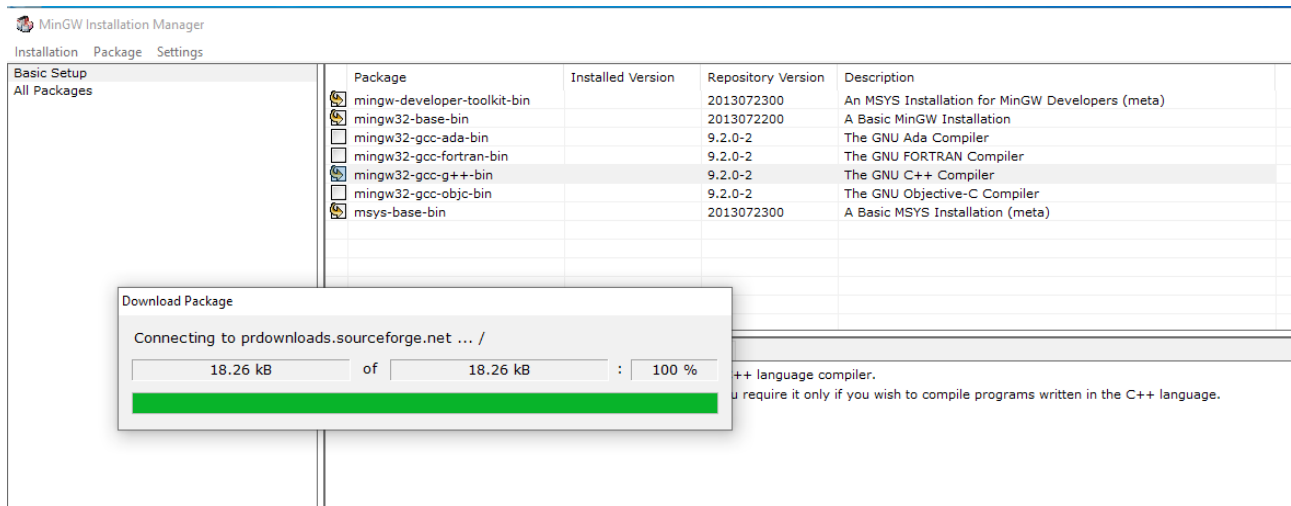
L'IDE CLion est déjà installé sur les machines de TP mais il faudra compléter et configurer l'outil en :

- installant un compilateur C++ (MinGW)
- activant la licence via un compte JetBrains
- configurant CLion pour MinGW

### 2.3.1 - Installation compilateur C++

L'installateur MinGW est déjà installé sur les machines de TP dans le dossier suivant : **C:\MinGW\bin\mingw-get.exe**

→ Après avoir lancé l'installateur cochez les outils à télécharger et installer comme ci-dessous :

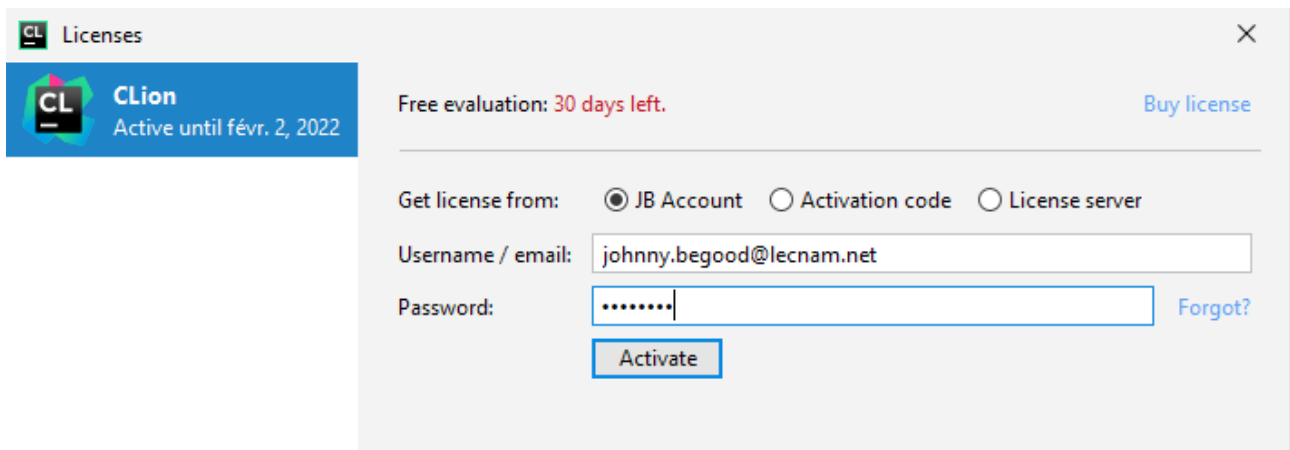


→ Puis depuis le menu **"Installation"**, sélectionnez **"Apply changes"**. Ces opérations de téléchargement et installation risquent de prendre un certain temps...

### 2.3.2 - Activation CLion

CLion comme tous les outils JetBrains sont activables avec une licence "éducation" via un compte JetBrains basé sur une adresse électronique du CNAM : [prenom.nom@lecnam.net](mailto:prenom.nom@lecnam.net)

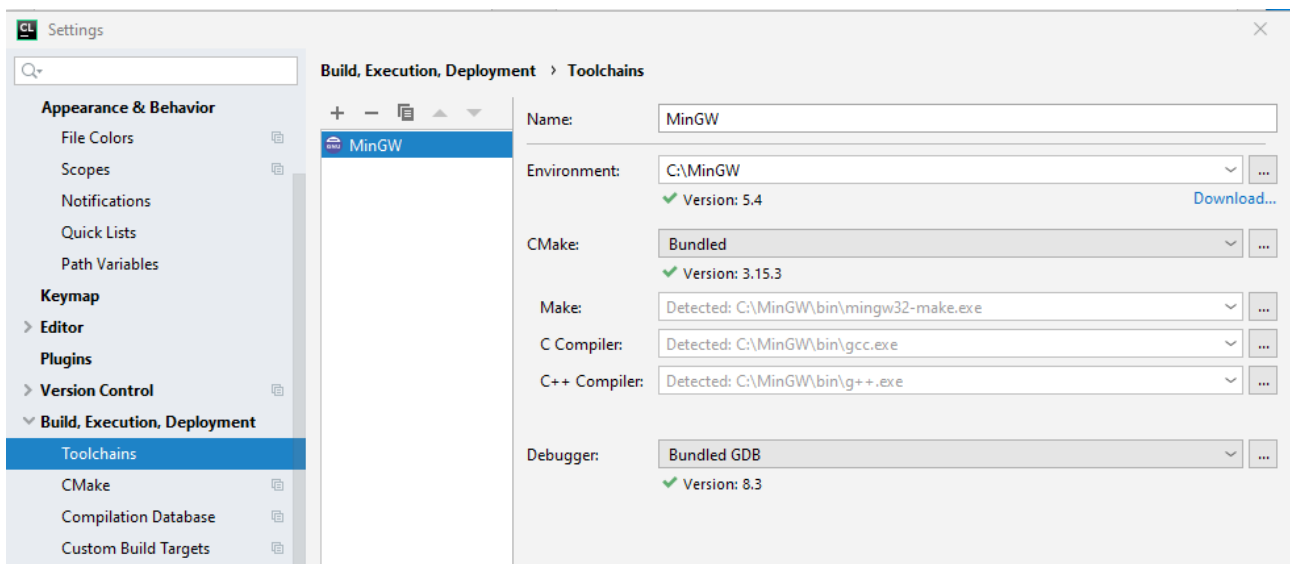




### 2.3.3 - Configuration CLion

La dernière étape consiste à configurer l'IDE CLion pour qu'il utilise le compilateur C++ installé avec MinGW.

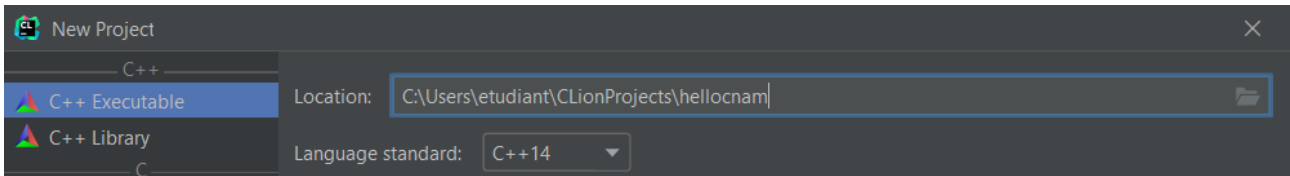
- Depuis le menu **"File"**, sélectionnez **"Settings..."**.
- Dans le menu de gauche déployez la rubrique **"Build, Execution, Deployment"** et sélectionnez l'entrée **"Toolchains"**
- Cliquez sur le bouton **"[+]"** pour ajouter un nouveau toolchain et choisissez **"MinGW"** dans le menu.
- Après quelques secondes, CLion devrait détecté tout seul les outils MinGW. Reste à confirmer avec le bouton **"Ok"**



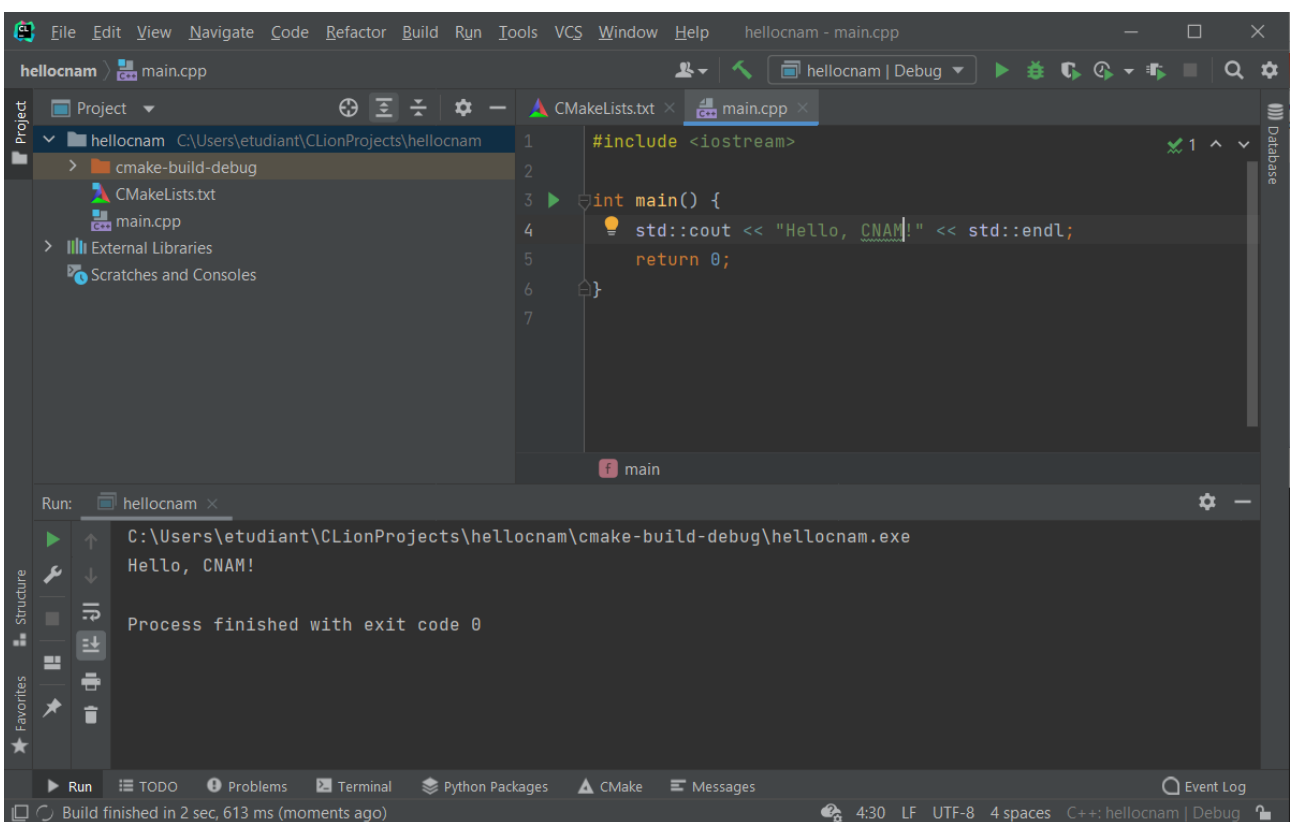
### 2.3.4 - Premier projet : Hello, CNAM!

Afin de valider la bonne installation et configuration de l'ensemble, nous allons créer un premier projet.

→ Depuis le menu **"File"**, sélectionnez **"New Project..."**. Choisissez le type de projet (C++ Executable) et un nom/dossier pour ce projet (hellocnam). Validez avec le bouton **"Create"**



→ Après chargement du projet vous devriez obtenir l'IDE comme ci-dessous. Modifiez le texte affiché et cliquez sur le bouton **"Run"** (triangle vert)



## 3 - Mini-projet Broomstick Flyers

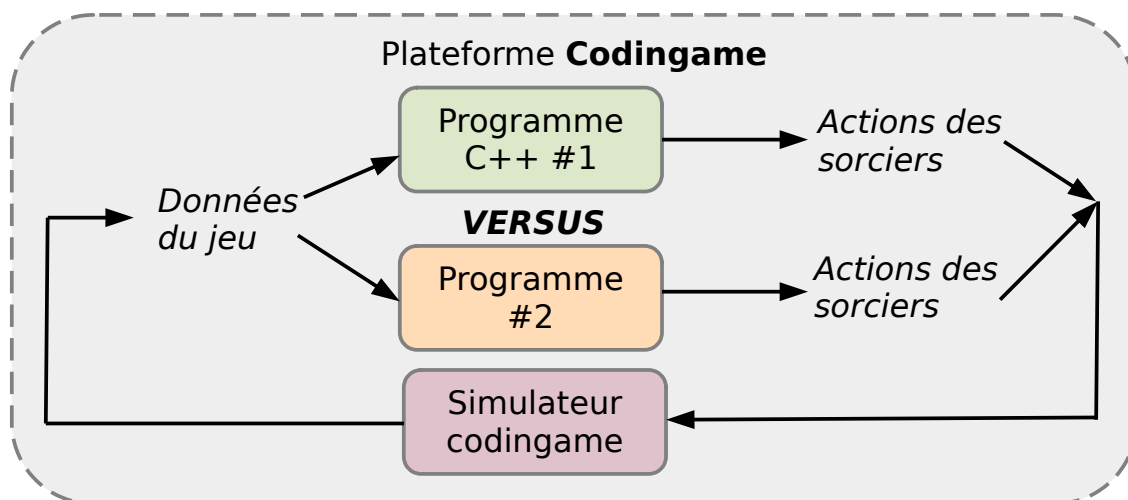
### 3.1 - Présentation

Le challenge auquel nous allons participer se nomme **Broomstick Flyers**. Ce challenge fera office de mini-projet "fil rouge" tout au long de ce dernier thème. Il est accessible ici : <https://www.codingame.com/ide/puzzle/fantastic-bits>

Ce challenge inspiré du roman Harry Potter propose de jouer au **Quidditch**. Vous devez "piloter" 2 sorciers, attraper les "snaffles" et tirer dans le but adverse pour marquer plus de buts que votre adversaire.



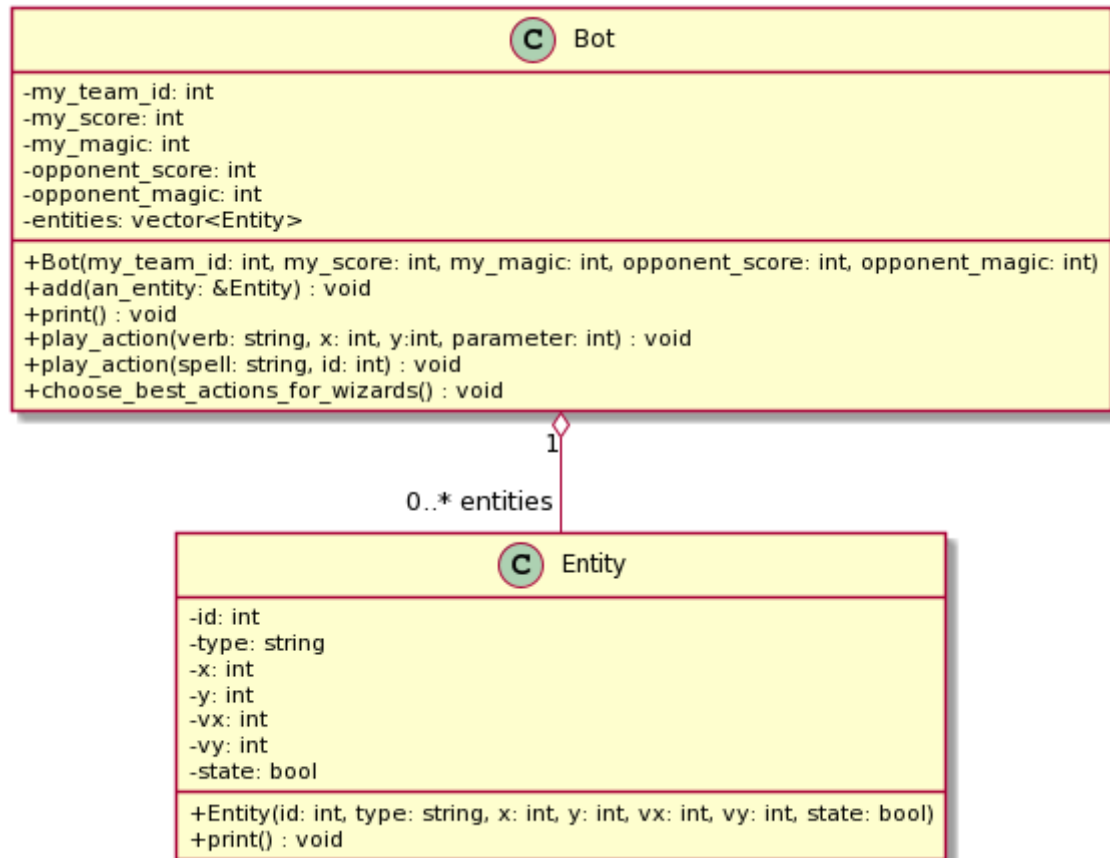
Bien entendu, ce n'est pas vous qui jouez directement mais un programme en C++ de votre conception. Toutes les données de la partie vous sont transmises sous forme de variables/objets et vous devez décider à chaque tour quelle action choisir pour chacun de vos sorciers (MOVE, THROW, SORT MAGIQUE...).



## 3.2 - Code de départ

→ Vous utiliserez le code de départ fourni sur Moodle dans le fichier **broomstick\_flyers.cpp**. Il vous suffira de copier/coller son contenu dans l'IDE de la plateforme codingame.

Ci-dessous le diagramme de classes UML représentant les différentes classes modélisant le jeu.



La classe **Entity** modélise toutes les entités mouvantes du jeu avec leurs caractéristiques à savoir :

- Wizard : les 2 sorciers à gérer et les 2 sorciers adverses
- Snaffles : les objets à attraper et lancer dans le but adverse
- Bludgers : les enquiquineurs

La classe **Bot** modélise votre IA qui va devoir choisir les meilleures actions possibles pour vos 2 sorciers en fonction des données du jeu en entrée. C'est principalement dans cette classe que vous programmerez votre IA.

### 3.3 - Erreurs courantes

Quelques erreurs courantes dans l'IDE codingame et les causes possibles.

#### Erreur #1

```
Informations :  
Timeout: the program did not provide 2 input lines in due time... freezed will no  
longer be active in this game.
```

Signifie que votre programme n'a pas répondu dans le délai de 100ms fixé. La cause la plus probable est une erreur de syntaxe dans votre code. Remontez la fenêtre pour le vérifier :

```
Sortie d'erreur :  
/tmp/Answer.cpp: In member function 'void Bot::choose_best_actions_for_wizards()':  
/tmp/Answer.cpp:279:46: error: expected ';' before ')' token  
279 |         play_action("MOVE", 16000, 7500, 100)  
    |                                     ^  
    |                                     ;  
280 |  
281 |     }  
    |     ~  
  
Initialisation :  
freezed Score: 0 | Magic: 0  
DumbleBoss Score: 0 | Magic: 0  
  
Informations :  
Timeout: the program did not provide 2 input lines in due time... freezed will no  
longer be active in this game.
```

Ici c'est bien le

cas, un point-virgule manque ligne 279. Si aucune erreur n'apparaît, c'est que votre programme a vraiment mis trop longtemps pour répondre

#### Erreur #2

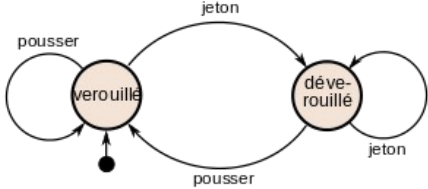
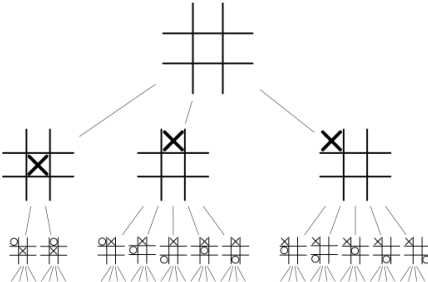
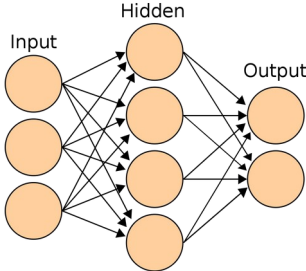
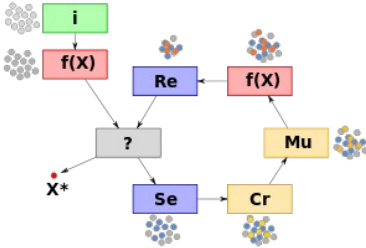
```
Attention : votre code n'a pas lu toutes les données disponibles  
depuis l'entrée standard avant d'avoir écrit sur la sortie standard.  
Ceci cause une désynchronisation qui peut entraîner des comportements  
inattendus.
```

Cette erreur signifie que votre programme a fourni trop d'ordre en sortie via les cout. Exemple, dans ce challenge, votre programme doit fournir 2 ordres, donc 2 cout. Si la plateforme en trouve 1 seul ou 3, ou plus, cela désynchronisera son fonctionnement.

Vérifiez donc le nombre de cout réalisé par votre programme.

### 3.4 - IA

Les IA pour ce genre de challenges vont consister à trouver la meilleure action possible en sortie en fonction des informations en entrée.

Nom	Description
<p>Machines à états finis</p> 	<p>Votre bot change d'état selon certaines conditions. Il effectue des actions différentes selon l'état dans lequel il est.</p> <p>Assez simple à mettre en œuvre avec des structures conditionnelles if/else</p>
<p>Arbres</p> 	<p>Votre bot va essayer toutes les actions possibles en sortie sous forme d'arbre. Chaque branche représente alors un scénario du jeu dans le futur. Il faut pouvoir simuler l'environnement du jeu pour appliquer les conséquences d'une action. Une fonction d'évaluation de la situation (heuristique) est aussi utilisée pour évaluer la meilleure feuille et ensuite remonter la branche.</p> <p>Différentes variations : DFS, BFS, minimax, alphabéta, beamsearch, MCTS</p>
<p>Réseaux de neurones</p> 	<p>Votre bot va entraîner un réseaux de neurones artificiels pour "apprendre" la meilleure action en sortie en fonction des entrées.</p> <p>Cela suppose de pouvoir simuler intégralement l'environnement de jeu et</p>
<p>Algorithme génétique</p> 	<p>Votre bot va faire évoluer itérativement les différentes actions possibles pour ne garder que la meilleure :</p> <ul style="list-style-type: none"> <li>- <math>f(X)</math> : fonction d'évaluation</li> <li>- ? : critère d'arrêt (temps, erreur...)</li> <li>- Re : remplacement</li> <li>- Se : sélection</li> <li>- Cr : croisement</li> <li>- Mu : mutation</li> </ul>

Source Images : wikipédia.

## 3.5 - Travail à faire et évaluation

→ A partir du code de départ fourni, programmez la meilleure IA possible. Le challenge s'arrêtera le 7 février prochain à 20h00. Le classement sera alors stabilisé et figé.



**A rendre** : le code source de votre IA **broomstick\_flyers.cpp** ainsi qu'une présentation de la stratégie que vous avez mis en œuvre dans le compte-rendu.

Au niveau de l'évaluation de ce mini-projet 2 critères seront pris en compte :

- qualité **algorithmique** de la solution : la plateforme grâce à son système de classement permettra d'évaluer les 4 points évoqués en cours :
  - ✓ terminaison :
  - ✓ correction :
  - ✓ complétude :
  - ✓ complexité :
- qualité du **code C++** : même si aucune convention de codage ne vous est imposée, il vous faudra rester attentif à différents items garantissant une lecture aisée par un autre intervenant :
  - ✓ indentation du code
  - ✓ nommage des variables et fonctions
  - ✓ commentaires

## 4 - Exercices structures de données

Ces exercices pourront être réalisés avec l'IDE CLion. En cas de problème avec cet outil, vous pourrez utiliser un IDE C++ en ligne parmi les suivants :

- <http://cpp.sh/>
- [https://www.onlinegdb.com/online\\_c++\\_compiler](https://www.onlinegdb.com/online_c++_compiler)
- <https://www.programiz.com/cpp-programming/online-compiler/>

### 4.1 - Broomstick flyers (tableaux, tri)

Comme vu au chapitre 3.2 avec le code de départ, l'essentiel des données en entrée du problème à traiter correspond à une collection d'entités mouvantes dans le jeu.

Le choix dans les structures de données se portera donc tout naturellement sur les tableaux et notamment le tableau dynamique **vector** car le nombre d'entités n'est pas connu et varie peut varier d'un tour à l'autre (ex: un snaffle marqué, disparaîtra du tableau).

0	1	2	3	4	5
<pre>_id=0 _type=WIZARD _x=123 _y=4444 _vx=0 _vy=0 _state=false</pre>	<pre>_id=1 _type=WIZARD _x=555 _y=234 _vx=0 _vy=0 _state=false</pre>	<pre>_id=2 _type=OPPONE _x=4687 _y=12 _vx=0 _vy=0 _state=false</pre>	<pre>_id=3 _type=OPPONE _x=42 _y=9999 _vx=0 _vy=0 _state=false</pre>	<pre>_id=4 _type=SNAFFLE _x=8000 _y=2341 _vx=0 _vy=0 _state=false</pre>	<pre>_id=5 _type=SNAFFLE _x=12000 _y=7499 _vx=0 _vy=0 _state=false</pre>

Quelques exemples d'utilisation :

```
// id de mon premier second sorcier
cout << _entities[0].get_id() << endl;    // 0

// coordonnées en X du second sorcier adverse
cout << _entities[3].get_x() << endl;    // 42

// type de l'entité dans la 5ième case (d'indice 4 donc)
cout << _entities[4].get_type() << endl; // "SNAFFLE"
```



→ Complétez les 4 **TODO** du code de **broomstick\_flyers.cpp**. Ils vous permettront d'avoir toutes les entités du jeu dans un tableau dynamique vector.

Le résultat attendu est de voir la liste des Entity s'afficher dans la console de debug par ordre d'id comme ci-dessous :

```
Informations de jeu, ...
Sortie d'erreur :
> === scores and magics ===
> my_team_id=0
> my_score=0, my_magic=0
> op_score=0, op_magic=0
> === entities ===
> id=0, type=WIZARD, x=1777, y=5243, vx=480, vy=-132, s=0
> id=1, type=WIZARD, x=4557, y=3015, vx=284, vy=61, s=0
> id=2, type=OPPONENT_WIZARD, x=14072, y=5644, vx=-27, vy=227, s=0
> id=3, type=OPPONENT_WIZARD, x=9698, y=5271, vx=-323, vy=2, s=0
> id=4, type=SNAFFLE, x=5771, y=2231, vx=0, vy=0, s=0
> id=5, type=SNAFFLE, x=8713, y=5099, vx=-1061, vy=-127, s=0
> id=6, type=SNAFFLE, x=6712, y=958, vx=0, vy=0, s=0
> id=7, type=SNAFFLE, x=9288, y=6542, vx=0, vy=0, s=0
> id=8, type=SNAFFLE, x=8000, y=3750, vx=0, vy=0, s=0
> id=9, type=BLUDGER, x=2354, y=4329, vx=-203, vy=-189, s=0
```



**A rendre** : le code source **broomstick\_flyers\_v1.cpp** complété et modifié pour gérer les entités dans un vector.

## 4.2 - Jeu de la Bataille (pile, file)

### 4.2.1 - Présentation

La bataille est l'un des plus anciens jeu de carte, datant environ du XIVe siècle.

Votre objectif dans cet exercice sera de créer un simulateur qui a partir d'une configuration de cartes donnée pour les 2 joueurs, donnera :

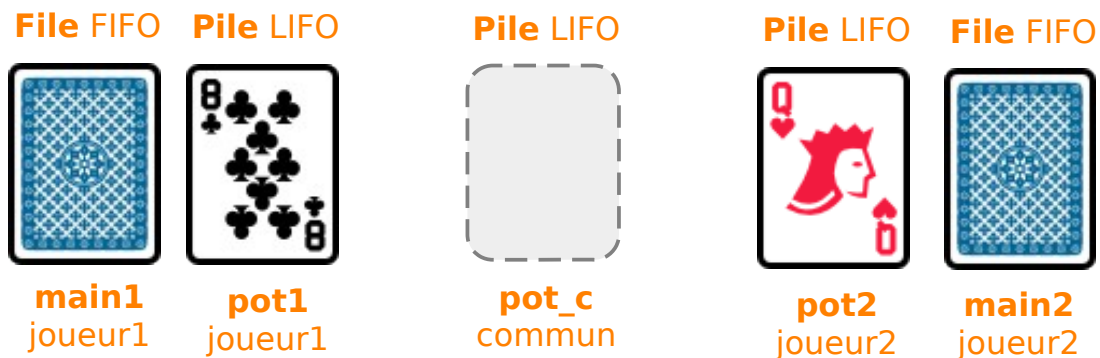
- le vainqueur (joueur1, joueur2 ou pat)
- le nombre de manches jouées

Un code source de départ **bataille.cpp** est disponible sur Moodle. Il contient plusieurs énumérations et classes servant à modéliser le jeu :







- **Valeur** : TWO=0, THREE=1, FOUR=2, FIVE=3... KING=11, ACE=12
- **Couleur** : SPADE=0, HEART=1, DIAMOND=2, CLUB=3
- **Carte** : représente une carte dans le jeu
- **Partie** : représente une partie à simuler

### 4.2.2 - Représentation des données

Représentation du jeu et structures de données retenues :









Format du fichier texte contenant la distribution des cartes en début de partie :

Fichier texte	Description
3	Nombre de cartes distribuées au <b>joueur1</b>
AD	  
KC	
QC	
3	Nombre de cartes distribuées au <b>joueur2</b>
KH	  
QS	
7H	

### 4.2.3 - Règles pour la simulation

Étapes de la simulation/partie. Une manche correspond aux étapes 2+3+4 :

Étape	Description
<b>#1</b> Distribution	<p>La distribution des cartes sera chargée depuis un fichier texte répondant à un format particulier (voir précédemment)</p> <div style="display: flex; justify-content: space-around; align-items: center;">   </div>

Étape	Description
<b>#2</b> Défausse	<p>Chaque joueur met la première carte de sa main dans son pot</p> 
<b>#3</b> Bataille	<p>Les cartes en haut de chacun des pots sont comparées avec 3 cas de figure :</p> <ul style="list-style-type: none"> <li>● si la carte du pot1 est plus petite que celle du pot2, alors le joueur2 remporte les cartes des 2 pots (voir étape #4)</li> <li>● si la carte du pot1 est plus grande que celle du pot2, alors le joueur1 remporte les cartes des 2 pots (voir étape #4)</li> <li>● si la carte du pot1 est de même valeur que celle du pot2, alors il y a "bataille" et les 2 joueurs doivent mettre 3 cartes de plus dans leur pot et revenir à l'étape #2</li> </ul> 
<b>#4</b> Levée	<p>Les cartes des 2 pots sont empilées dans le pot central en commençant par le joueur2</p>  <p>Le joueur gagnant récupère les cartes du pot central en les dépilant. Les cartes du joueur1 seront donc enfilées en premier.</p> 
<b>#5</b> Rebouclage	<p>Tant que les 2 joueurs ont encore des cartes dans leurs mains ont reboucle à l'étape #2</p>
<b>#6</b> Résultats	<p>Affichage des résultats de la simulation</p>

Exemple concret du déroulement d'une manche (étape 2+3+4)

<b>Distribution</b>	<b>Déroulement</b>
7 10D 9S 8D KH 7D 5H 6S 7 10H 7H 5C QC 2C 4H 6D	<p><b>Étape #2 : défausse</b></p> <ul style="list-style-type: none"> <li>- main1 : 10D 9S 8D KH 7D 5H 6S</li> <li>- pot1 :</li> <li>- pot_c :</li> <li>- pot2 :</li> <li>- main2 : 10H 7H 5C QC 2C 4H 6D</li> </ul> <p><b>Étape #3 : bataille</b></p> <ul style="list-style-type: none"> <li>- main1 : 9S 8D KH 7D 5H 6S</li> <li>- pot1 : 10D</li> <li>- pot_c :</li> <li>- pot2 : 10H</li> <li>- main2 : 7H 5C QC 2C 4H 6D</li> </ul> <p><b>Étape #3 : bataille + défausse</b></p> <ul style="list-style-type: none"> <li>- main1 : 5H 6S</li> <li>- pot1 : 7D KH 8D 9S 10D</li> <li>- pot_c :</li> <li>- pot2 : 2C QC 5C 7H 10H</li> <li>- main2 : 4H 6D</li> </ul> <p><b>Étape #4 : empilement dans pot_c, j2 avant j1</b></p> <ul style="list-style-type: none"> <li>- main1 : 5H 6S</li> <li>- pot1 :</li> <li>- pot_c : 2C QC 5C 7H 10H 7D KH 8D 9S 10D</li> <li>- pot2 :</li> <li>- main2 : 4H 6D</li> </ul> <p><b>Étape #4 : dépilement de pot_c dans main1</b></p> <ul style="list-style-type: none"> <li>- main1 : 5H 6S 10D 9S 8D KH 7D 10H 7H 5C QC 2C</li> <li>- pot1 :</li> <li>- pot_c :</li> <li>- pot2 :</li> <li>- main2 : 4H 6D</li> </ul>

Et si on déroule les autres manches, on doit obtenir le joueur1 gagnant en 3 manches.

#### 4.2.4 - Travail à faire

→ Complétez la méthode **simuler()** de la classe **Partie** afin de pouvoir simuler l'intégralité d'une partie du jeu de la bataille à partir d'une distribution de cartes donnée dans un fichier texte.

Différentes distributions (**test00.txt**, **test01.txt...**) de cartes sont disponibles sur Moodle dans le fichier **distributions.zip**.

<b>Fichier</b>	<b>Description</b>	<b>Résultat attendu</b>
test00.txt	Exemple ci-dessus (4.2.3)	1 3
test01.txt	Mini-partie (3 cartes/joueur)	1 3
test02.txt	Partie complète et courte (26 cartes/joueur)	2 26
test03.txt	Partie complète et moyenne (26 cartes/joueur)	2 56
test04.txt	Bataille (5 cartes/joueur)	2 1
test05.txt	Partie complète avec bataille (26 cartes/joueur)	1 52
test06.txt	Deux batailles enchaînées (9 cartes/joueur)	2 1
test07.txt	Partie complète et longue (26 cartes/joueur)	2 1262
test08.txt	Égalité PAT (26 cartes/joueur)	PAT
test09.txt	Autre égalité PAT (26 cartes/joueur)	PAT



**A rendre** : le code source **bataille.cpp** complété et modifié pour simuler une partie

## 4.3 - Nuage de mots (dictionnaire, tableau, tri)

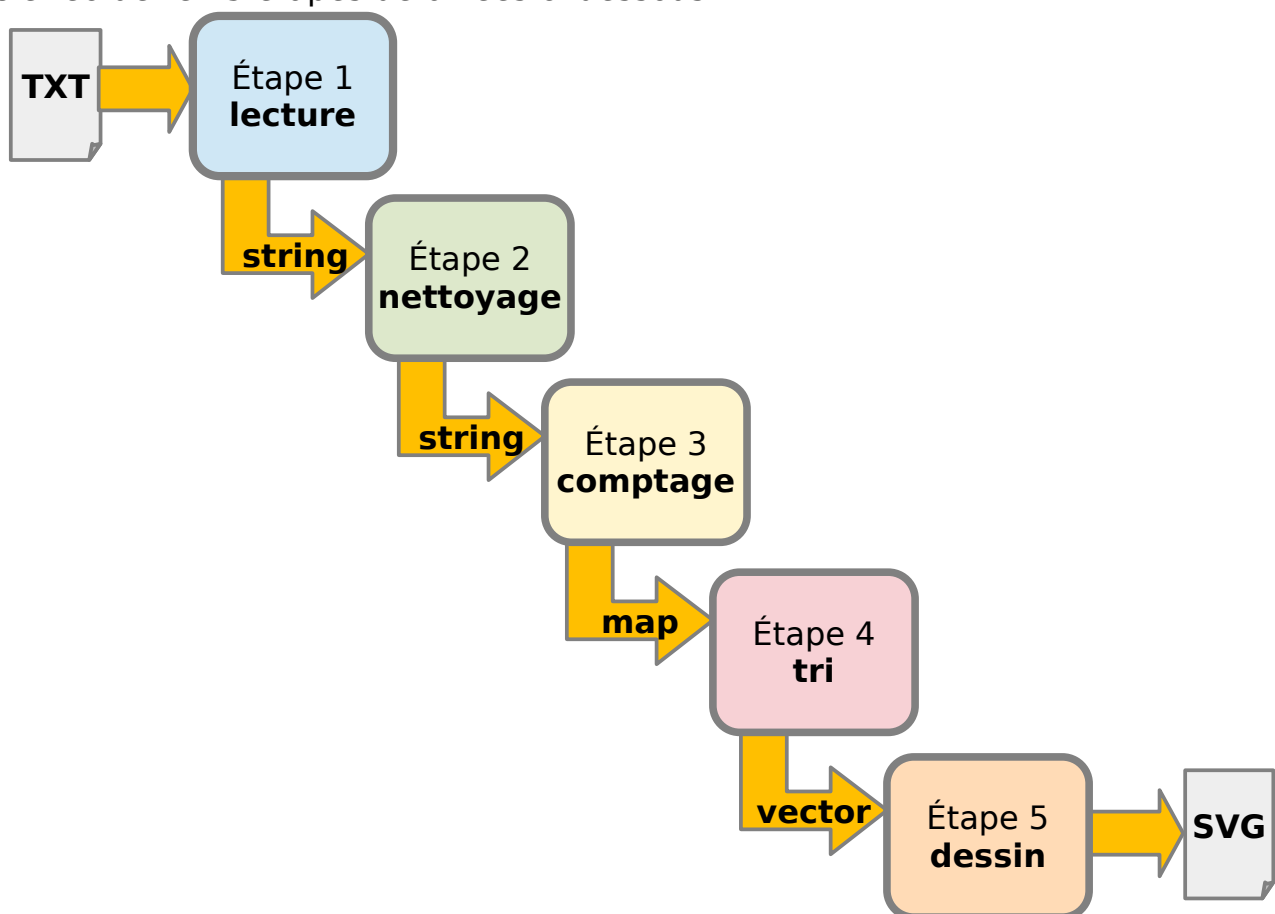
### 4.3.1 - Présentation

Un nuage de mot (ou tag cloud) est une représentation visuelle des mots les plus utilisés dans un texte. C'est une sorte de résumé graphique d'un texte. La taille d'un mot est alors fonction du nombre de fois où il est présent dans le texte.



### 4.3.2 - Transformations des données

La transformation d'un texte en image contenant le nuage de mots peut s'effectuer en 5 étapes détaillées ci-dessous :



→ Étape 1 (**lecture**) : méthode **NuageDeMots::lire\_fichier()** :

Cette méthode vous est fournie intégralement dans le code source de départ **nuage\_de\_mots.cpp** disponible sur Moodle. Vous pouvez tester cette méthode avec l'un des fichiers exemples fournis sur Moodle également (**textes.zip**).

→ Étape 2 (**nettoyage**) : méthode **NuageDeMots::nettoyer()** :

Il s'agit de transformer une chaîne de caractères string en une autre après avoir nettoyé les caractères de ponctuation et saut de ligne.

→ Étape 3 (**comptage**) : méthode **NuageDeMots::compter\_mots()** :

Cette étape regroupe en fait 2 étapes, puisque avant de pouvoir compter les mots, il faut au préalable segmenter le texte en mots.

En parcourant chaque caractère de la chaîne de caractères (string) fournie en paramètre, dès qu'on tombe sur le caractère espace, c'est qu'un mot vient de se terminer et qu'un nouveau commence.

Si ce mot existe déjà dans le dictionnaire, il faut incrémenter son nombre d'occurrences, sinon il faut ajouter ce nouveau mot au dictionnaire avec un nombre d'occurrences à 1.


→ Étape 4 (**tri**) : méthode **NuageDeMots::trier\_mots()** :

Il n'est pas possible de trier directement un dictionnaire (map) en C++ comme c'est le cas en Python. Il faut donc ajouter tous les mots du dictionnaire (map) dans un tableau (vector) et ensuite appliquer l'algorithme sort pour trier les mots en fonction de leur nombre d'occurrences dans le texte.

→ Étape 5 (**dessin**) : méthode **NuageDeMots::lire\_fichier()** :

La méthode est déjà partiellement fournie avec le début du code SVG de l'image finale. SVG est un format d'image vectorielle où les éléments sont dessinés à partir de balises un peu comme en HTML.

Voici un exemple :

Code SVG	Image résultante
<pre>&lt;?xml version="1.0" encoding="utf-8"?&gt; &lt;svg width="800" height="600"&gt; &lt;rect width="800" height="600" fill="white" /&gt; &lt;text x="100" y="250" fill="rgb(255,0,0)" fill- opacity="0.8" font-size="100"&gt;Hello SVG&lt;/text&gt; &lt;/svg&gt;</pre>	

### 4.3.3 - Travail à faire

→ Complétez les méthodes de la classe **NuageDeMots** décrites précédemment. Le code source de départ **nuage\_de\_mots.cpp** est disponible sur Moodle.



**A rendre** : le code source **nuage\_de\_mots.cpp** complété et modifié pour générer une image SVG contenant le nuage de mots

## 4.4 - Learderboard (arbres, récursivité)

### 4.4.1 - Présentation

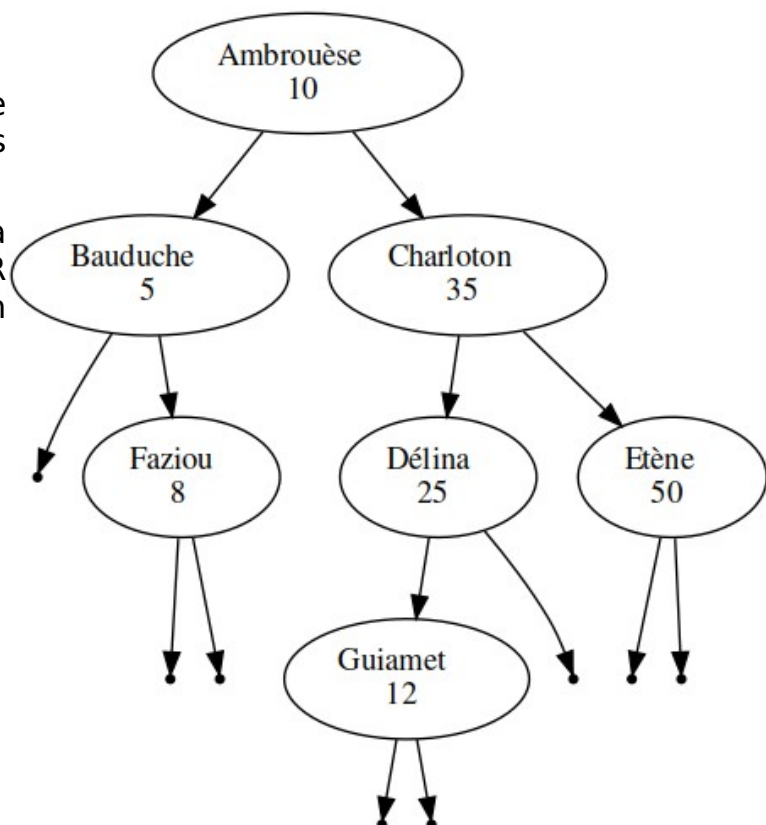
Les jeux de type MOBA (Multiplayer Online Battle Arena) attirent de nombreux joueurs. Un des composants majeurs de ces jeux est la "leaderboard" qui classe les joueurs entre eux et entretient la motivation des participants.

### 4.4.2 - Structure de données

Si les bases de données traditionnelles restent efficaces pour un nombre de joueurs raisonnable, d'autres structures de données deviennent plus intéressantes en termes de performances quand le nombre de joueurs augmente.

L'arbre binaire de recherche (ABR) fait partie de ces structures de données.

L'objectif de cet exercice sera de mettre en œuvre un ABR pour gérer la leaderboard d'un jeu.



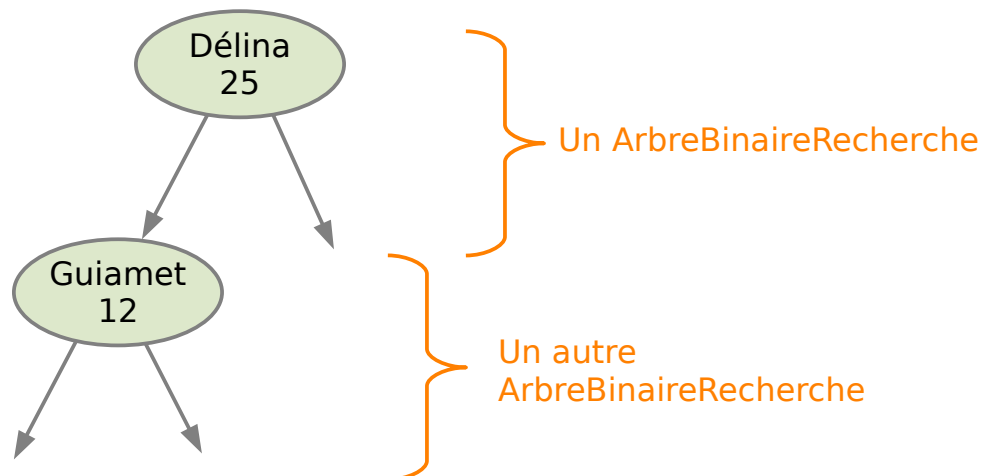


Le diagramme de classes UML ci-dessous dessine les contours de la modélisation retenue :



La classe **Joueur** représentera le joueur et ses données (pseudo + score).

La classe **ArbreBinaireRecherche** représentera la leaderboard. Comme on peut le voir avec la liaison auto-réflexive, il s'agira d'une structure de données récursive, faisant appel à elle-même pour ses nœuds enfants gauche et droite.



### 4.4.3 - Fonctionnalités

On vous demande de réaliser plusieurs fonctionnalités à l'aide de cette structure de données ABR. A chacune de ces fonctionnalités correspond une méthode dans la classe **ArbreBinaireRecherche**.

→ FS1 : Ajouter un nouveau joueur (avec pseudo et score) dans la leaderboard.

Méthode **ArbreBinaireRecherche::ajouter\_nouveau()**

→ FS2 : Compter les joueurs dans la leaderboard

Méthode **ArbreBinaireRecherche::compter\_joueurs()**

→ FS3 : Rechercher le meilleur joueur (avec le score le plus élevé)

Méthode **ArbreBinaireRecherche::rechercher\_meilleur\_joueur()**

→ FS4 : Rechercher un joueur à partir de son pseudo

Méthode **ArbreBinaireRecherche::rechercher\_joueur\_avec\_pseudo()**

→ FS5 : Rechercher un joueur à partir de son score

Méthode **ArbreBinaireRecherche::rechercher\_joueur\_avec\_score()**

→ FS6 : Calculer le classement d'un joueur à partir de son pseudo

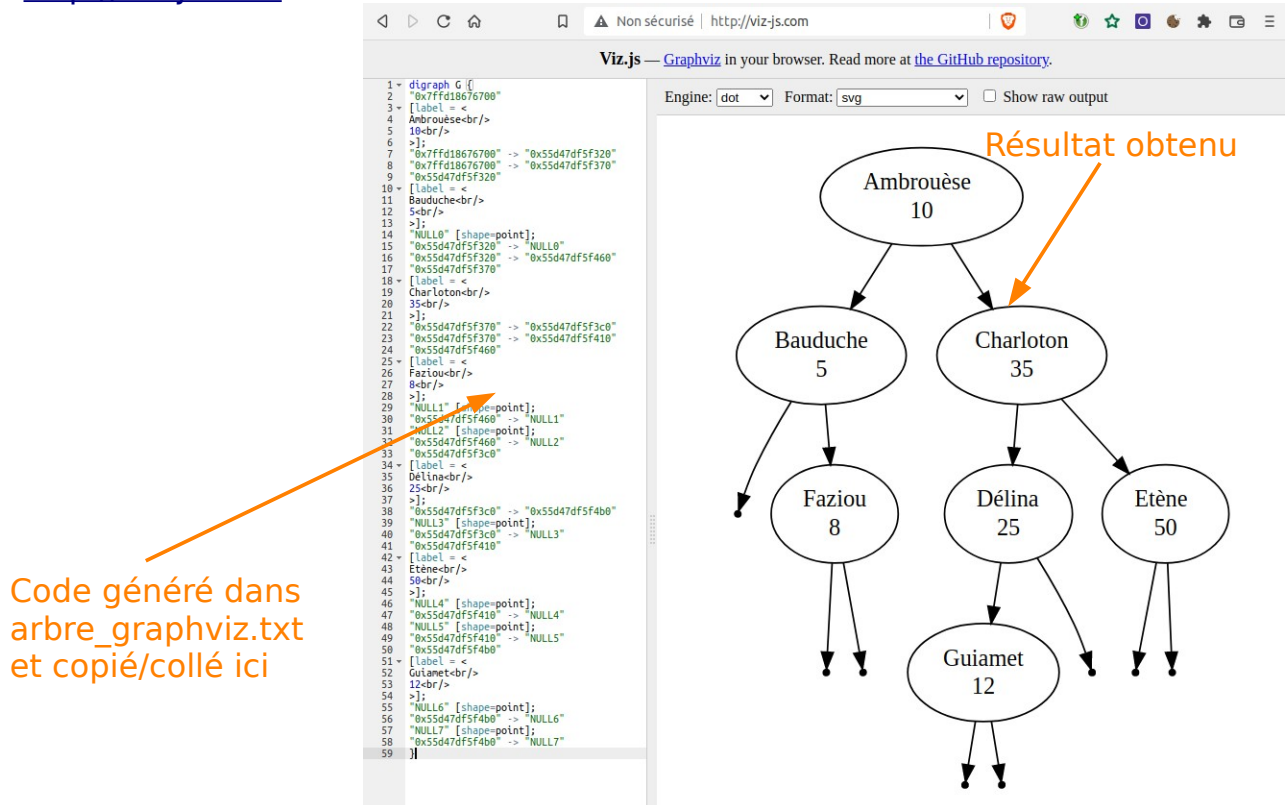
Méthodes **ArbreBinaireRecherche::calculer\_classement\_joueur()** et **ArbreBinaireRecherche::parcourir\_pour\_classement()**

→ FS7 : Sérialiser la leaderboard afin de pouvoir la sauvegarder

Méthode **ArbreBinaireRecherche::serialiser()**

#### 4.4.4 - Visualisation avec Graphviz

Afin de faciliter votre travail, il vous sera sans doute nécessaire de visualiser l'arbre construit. La méthode **ArbreBinaireRecherche::exporter()** sauvegarde votre leaderboard dans un fichier nommé `arbre_graphviz.txt`. Il suffit alors de copier/coller le code généré au format Graphviz dans le site <http://viz-js.com>



#### 4.4.5 - Travail à faire

→ Complétez les méthodes de la classe **ArbreBinaireRecherche** décrites précédemment pour réaliser les fonctionnalités FSx.

Le code source de départ **leaderboard.cpp** est disponible sur Moodle.



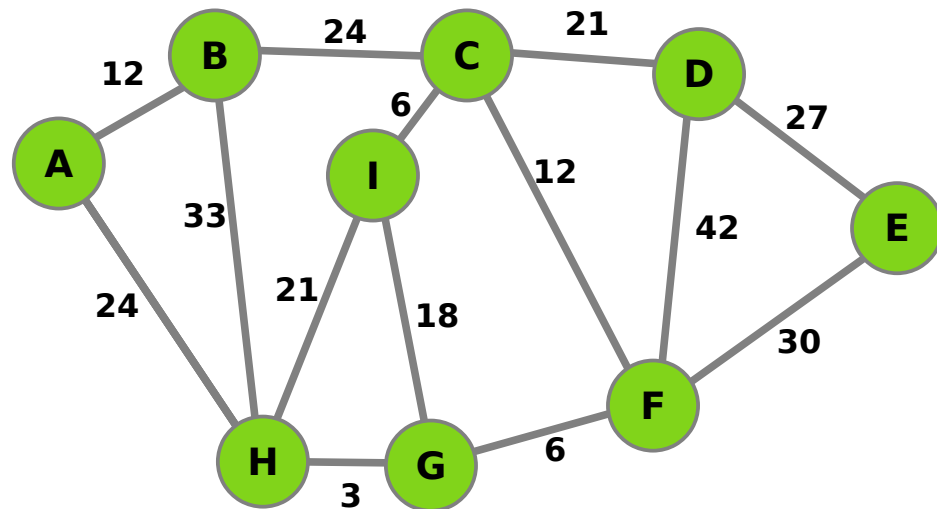
**A rendre** : le code source **leaderboard.cpp** complété et modifié pour gérer la leaderboard

## 4.5 - Navigation routière (graphes)

### 4.5.1 - Présentation

Soit le réseau routier suivant reliant les 9 villes et modélisé par la graphe ci-dessous :

Arantelle  
Bardou  
Chaline  
Drigail  
Encoinson  
Fumorge  
Garouil  
Hucher  
Itaux



### 4.5.2 - Représentation des données

Le graphe ci-dessus sera représenté par une matrice d'adjacence et un tableau de ville comme ci-dessous :

	<i>Aran.</i>	<i>Bard.</i>	<i>Chal.</i>	<i>Drig.</i>	<i>Enco.</i>	<i>Fumo.</i>	<i>Gar.</i>	<i>Huch.</i>	<i>Itau.</i>
<b>Arantelle</b>	0	12	0	0	0	0	0	24	0
<b>Bardou</b>	12	0	24	0	0	0	0	33	0
<b>Chaline</b>	0	24	0	21	0	12	0	0	6
<b>Drigail</b>	0	0	21	0	27	42	0	0	0
<b>Encoinson</b>	0	0	0	27	0	30	0	0	0
<b>Fumorge</b>	0	0	12	42	30	0	6	0	0
<b>Garouil</b>	0	0	0	0	0	6	0	3	18
<b>Hucher</b>	24	33	0	0	0	0	3	0	21
<b>Itaux</b>	0	0	6	0	0	0	18	21	0

Exemple : la distance entre Chaline et Drigail est de 21 kms

Comme les routes permettent de circuler dans les 2 sens, la matrice est symétrique autour de sa diagonale.

Une valeur de 0 indique qu'il n'y a pas de route praticable entre 2 villes.

### 4.5.3 - Travail à faire

→ En vous aidant du code source de départ **itineraire.cpp** disponible sur Moodle, complétez la fonction **chercher\_le\_plus\_court\_chemin\_entre()** et en utilisant l'algorithme de recherche du plus court chemin de Dijkstra afficher à la fois :

- la distance de ce chemin
- les étapes du début à la fin

Ainsi...

```
chercher_chemin_le_plus_court_entre("Arantelle", "Fumorge");
```

...devrait afficher :

```
Distance = 27  
Etapes   = Arantelle > Hucher > Garouil > Fumorge
```

→ Comment faudrait-il modifier la matrice d'adjacence pour traduire le fait qu'une chute d'arbre a rendu la route entre Hucher et Garouil impraticable ?

Quel sera alors le plus court chemin entre Arantelle et Fumorge ?



**A rendre** : le code source **itineraire.cpp** complété et modifié qui permet de trouver le plus court chemin