

Projet Jeux Olympiques



Ce projet veut s'articuler dans le cadre des Jeux Olympiques et a pour but de proposer une façon originale de présenter les classements des joueurs utilisant un cadre simple pour la lecture de données : le format texte, nous savons que nous aurions pu utiliser le format Json ou xml afin d'utiliser un format normalisé potentiellement utilisé partout dans les API actuelles, toutefois le format texte a semblé bien plus simple à mettre en œuvre, tout d'abord car C++ ne parse pas naturellement le json, ensuite parce que les opérateurs de stream fonctionnent très bien avec les espaces et les retours à la ligne, de

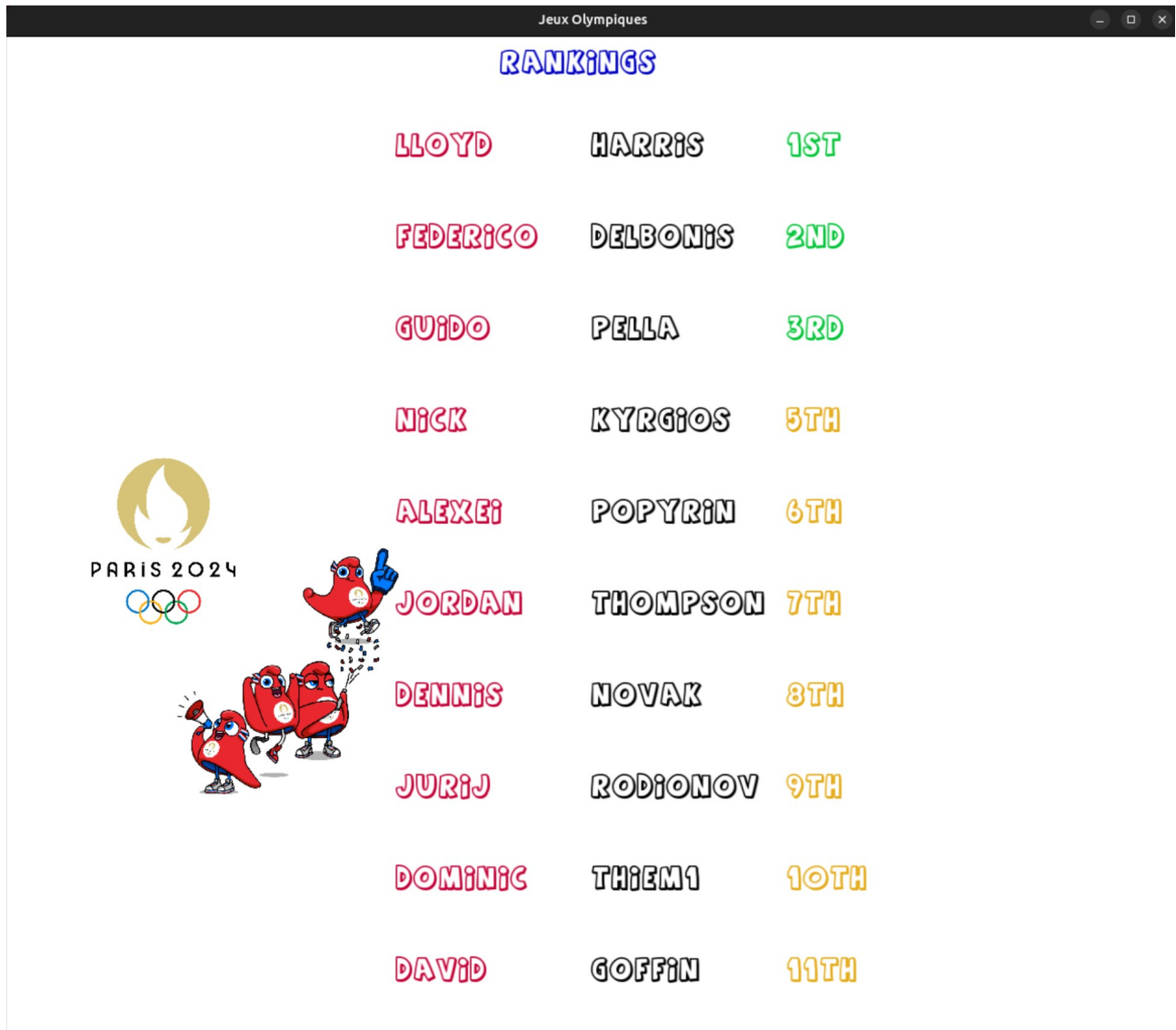
l'autre côté le fichier texte reste lisible pour l'humain. Par conséquent le fichier texte a été choisi. C'est évidemment une question centrale que de décider comment représenter la donnée, néanmoins, même si Json semble populaire, il n'est pas si lisible que cela par l'humain, pour ce qui est de XML, c'est la même problématique, finalement au regard de la relative faible complexité des données manipulées, le fichier texte standard est suffisant.

Notre programme lit d'abord le nom de famille de l'athlète, puis le prénom et enfin la position. Il affiche enfin dans l'ordre à l'écran le classement de manière conviviale pour l'utilisateur.

Voici le sommaire :

- I) Description de l'Application développée
- II) Utilisation des contraintes
- III) Procédure d'installation et d'exécution du code
- IV) Parties de l'implémentation dont je suis le plus fier

I) Description de l'application développée



L'application est développée est un système de classement des joueurs après une épreuve. Imaginons une épreuve des jeux Olympiques comme le tir à l'arc où l'on veut pouvoir, après la prestation des athlètes présenter le classement des joueurs, c'est un écran qui se retrouve dans de nombreuses disciplines et qui permet au spectateur de savoir où en est la partie. Par conséquent l'application s'adresse clairement au spectateur et il faut pouvoir présenter les informations de classement de manière lisible au spectateur tout en préservant une certaine convivialité et l'esprit festif des jeux Olympiques, par conséquent il a été décidé de mettre en avant les personnages mascottes, les *phryges* à l'écran afin de répondre au besoin de convivialité et afin de pouvoir attirer l'attention du spectateur. Ensuite il s'agit donc surtout de présenter les différents joueurs avec le classement. Pour le moment les athlètes sont affichés dans l'ordre les uns en dessous des autres. Chaque ligne contient d'abord le nom de famille de l'athlète, puis le prénom et enfin la position. Le but est d'offrir la plus grande lisibilité possible et nous avons opté pour des couleurs avec un contraste suffisamment fort.

II) Utilisation des contraintes,

tout d'abord nous avons bien vérifié à l'aide de l'outil valgrind l'absence de fuite mémoire, ce qui peut être problématique lorsque lancé en continu(on peut imaginer une fin de partie où l'application reste affichée pendant un long moment afin d'attendre la prochaine manche ou la prochaine partie).

```
Score: 0
Nom: Jordan
Prenom: Thompson
Score: 7

Nom: Dennis
Prenom: Novak
Score: 8

Nom: Jurij
Prenom: Rodionov
Score: 9

Nom: Dominic
Prenom: Thiem1
Score: 0

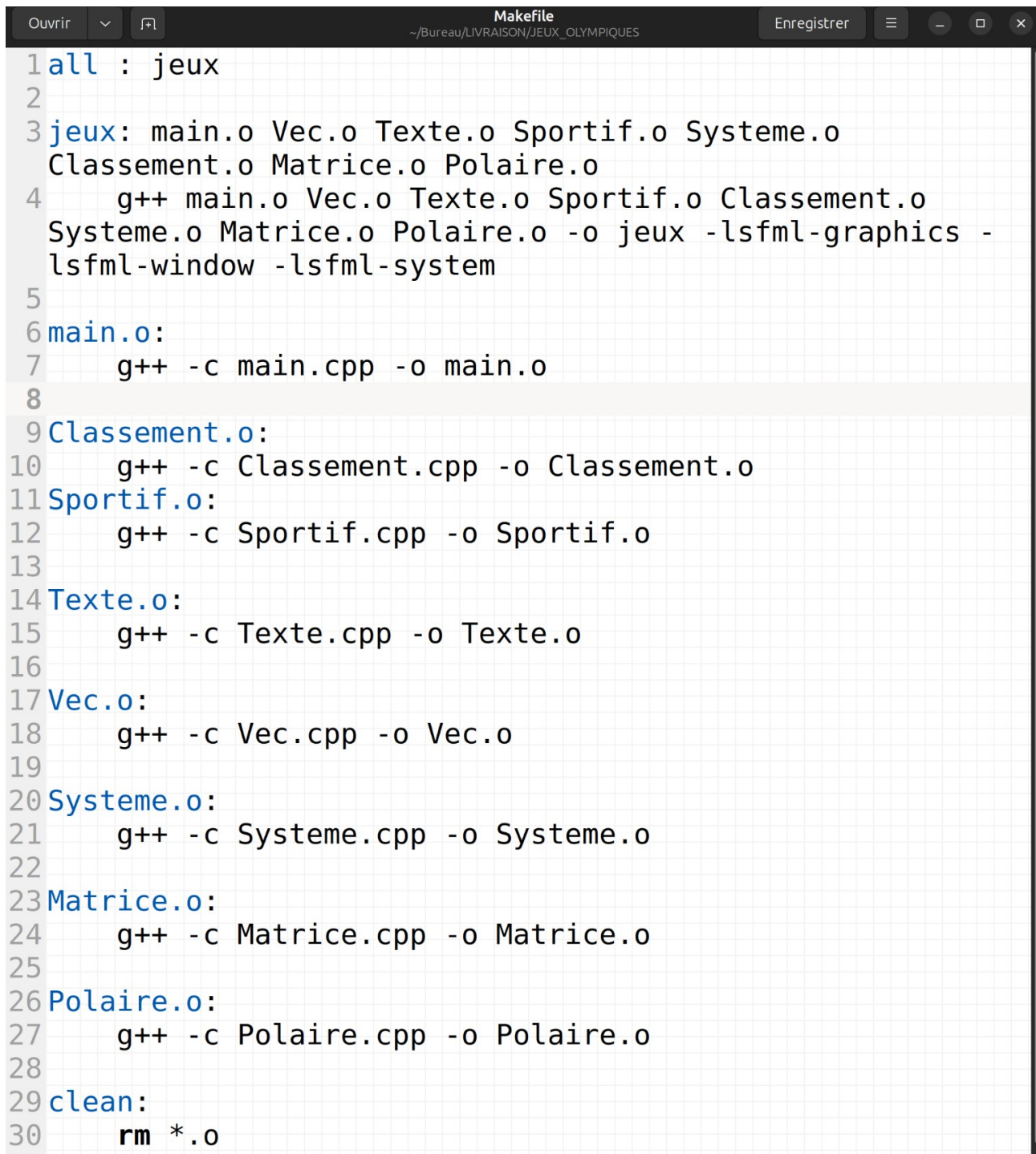
Nom: David
Prenom: Goffin
Score: 11

Nom: David
Prenom: Goffin
Score: 11

PROJET JEUX OLYMPIQUES
==66257==
==66257== HEAP SUMMARY:
==66257==   in use at exit: 226,140 bytes in 2,671 blocks
==66257==   total heap usage: 202,126 allocs, 199,455 frees, 179,910,056 bytes allocated
==66257==
==66257== LEAK SUMMARY:
==66257==   definitely lost: 0 bytes in 0 blocks
==66257==   indirectly lost: 0 bytes in 0 blocks
==66257==   possibly lost: 0 bytes in 0 blocks
==66257==   still reachable: 226,140 bytes in 2,671 blocks
==66257==     suppressed: 0 bytes in 0 blocks
==66257== Rerun with --leak-check=full to see details of leaked memory
==66257==
==66257== For lists of detected and suppressed errors, rerun with: -s
==66257== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Figure 2) Trace de valgrind sur l'application

Comme on peut le voir sur la figure 2, il n'y pas de fuite mémoire, à aucun moment ce qui garantit une utilisation optimale des ressources. Après avoir tenté diverses méthodes, il est apparu qu'il était possible d'éviter l'utilisation de pointeurs dans plusieurs situations, on pouvait tout d'abord passer les paramètres en référence, pour finalement bien veiller à effectuer les delete nécessaires à la fin du programme. Etant donné que chercher les fuites mémoires peut prendre énormément de temps, nous avons fait attention à tester progressivement l'application avec valgrind. Ensuite pour compiler le projet, un Makefile a été utilisé et il possède bien les règles clean et all, comme visible ci-dessous :



The image shows a screenshot of a text editor window titled "Makefile". The window has a dark theme and a grid background. The file path in the title bar is "~/Bureau/LIVRAISON/JEUX_OLYMPIQUES". The editor contains the following Makefile content:

```
1 all : jeux
2
3 jeux: main.o Vec.o Texte.o Sportif.o Systeme.o
  Classement.o Matrice.o Polaire.o
4     g++ main.o Vec.o Texte.o Sportif.o Classement.o
  Systeme.o Matrice.o Polaire.o -o jeux -lsfml-graphics -
  lsFML-window -lsfml-system
5
6 main.o:
7     g++ -c main.cpp -o main.o
8
9 Classement.o:
10    g++ -c Classement.cpp -o Classement.o
11 Sportif.o:
12    g++ -c Sportif.cpp -o Sportif.o
13
14 Texte.o:
15    g++ -c Texte.cpp -o Texte.o
16
17 Vec.o:
18    g++ -c Vec.cpp -o Vec.o
19
20 Systeme.o:
21    g++ -c Systeme.cpp -o Systeme.o
22
23 Matrice.o:
24    g++ -c Matrice.cpp -o Matrice.o
25
26 Polaire.o:
27    g++ -c Polaire.cpp -o Polaire.o
28
29 clean:
30     rm *.o
```

Figure 3 – le Makefile utilisé pour compiler le projet

III) Procédure d'installation

La procédure d'installation est relativement simple sur une machine Linux, elle consiste en l'ajout d'une seule bibliothèque : la SFML, et en le linking de la librairie lors de la compilation, voici les étapes à suivre pour pouvoir lancer l'application

- 1) Ouvrir un terminal
- 2) Taper `sudo apt get install libsfml-dev`
- 3) valider les étapes sur le terminal et attendre la fin de l'installation
- 4) exécuter la commande `make`

IV) Partie dont nous serions le plus fier

La partie qui consiste à mettre en place toute la géométrie, à travers la classe Vec qui permet de se repérer dans le plan en deux dimensions, puis ensuite la classe Polaire, permettant de se repérer en coordonnées polaires afin d'effectuer facilement des rotations ainsi que l'animation de la scène sont la partie où nous présentons le plus de satisfaction. La force de C++ est de permettre l'encapsulation

Figure 4 - Diagramme UML de l'application

