

# Rapport de projet C++

Mathieu Nguyen et Tania Mendes Dias

January 27, 2023



# Contents

<b>1 Description de l'application développée</b>	<b>1</b>
1.1 Mise en contexte . . . . .	1
1.2 Déroulement du jeu . . . . .	1
<b>2 Mise en valeur des contraintes utilisées</b>	<b>1</b>
2.1 Classes utilisées et Hiérarchie . . . . .	1
2.2 Méthodes virtuelles . . . . .	1
2.3 Surcharges d'opérateurs . . . . .	1
2.4 Conteneurs utilisés . . . . .	2
2.5 la Bibliothèque Simple and Fast Multimedia Library (SFML) . . . . .	2
<b>3 Diagramme UML de l'application</b>	<b>3</b>
<b>4 Procédures d'installation et exécution</b>	<b>5</b>
4.1 Procédures d'installation . . . . .	5
4.2 Exécution de l'application . . . . .	5
<b>5 Gameplay et Utilisation</b>	<b>7</b>
5.1 Manipuler le menu . . . . .	7
5.2 Manipuler les options . . . . .	7
5.3 Manipuler le jeu . . . . .	8
5.4 Règles du jeu . . . . .	8
<b>6 Nos plus grandes fiertés</b>	<b>10</b>
6.1 Le rendu esthétique . . . . .	10
6.2 Implémentation d'une musique . . . . .	11
6.3 Sélection de musique . . . . .	11
6.4 Basculer d'un écran à l'autre . . . . .	11
6.5 Le gameplay en tour par tour . . . . .	12
6.6 Cacher l'interface graphique . . . . .	12
<b>7 Aller plus loin</b>	<b>12</b>
7.1 Ajouter plus d'équipes . . . . .	12
7.2 Choisir ses joueurs . . . . .	12
7.3 Mécanique de passe . . . . .	12
7.4 Savoir à qui on fait la passe . . . . .	12
7.5 Méthodes de calcul de probabilités . . . . .	12
7.6 Intelligence de l'adversaire . . . . .	12
7.7 Mécanique de défense . . . . .	12

# 1 Description de l'application développée

Notre application est un jeu basé sur le football, mais dont le gameplay est considérée comme **tour par tour**. C'est un jeu de football classique où nous avons une équipe de 11 joueurs contre 11 joueurs et l'objectif est de marquer plus de buts que l'équipe adverse. Cependant, au lieu d'un gameplay en temps réel, nous avons à chaque instant une décision à prendre, **sans se préoccuper du temps de reflexion**.

## 1.1 Mise en contexte

Notre application est directement inspirée de la coupe du monde de Football. En effet, deux équipes de nationalités différentes se confrontent dans un match de football et le gagnant remporte une **bénédiction pour son pays**, ainsi que la coupe du championnat.

Ce dernier match oppose la France au Portugal. Le Portugal veut pouvoir officiellement acclamer **Christiano Ronaldo comme meilleur joueur de tous les temps**, tandis que la France cherche à **récupérer l'Alsace-Lorraine**.

Le souhait du pays gagnant le championnat sera réalisé par l'organisateur. La victoire de ce dernier est donc primordial ! Pour cela, la France nécessitera votre aide.

## 1.2 Déroulement du jeu

Pour aider l'équipe de France, c'est simple! Il suffit de jouer et gagner le match. Pour cela il est primordiale de comprendre le déroulement du match.

Tout d'abord, un joueur sera attribué comme ayant la balle à chaque tour. À chaque instant, si l'utilisateur a la balle, celui-ci va jouer le joueur possédant la balle et va décider de l'action à réaliser : passe à un allié, dribbler (garder la balle tout en avançant) ou tirer directement dans les cages.

Ces trois actions ont une **probabilité de réussite dépendant de plusieurs facteurs** : le potentiel du joueur, le potentiel de l'adversaire défendant la balle, la distance avec l'objectif (allié ou cage), le nombre de joueurs qui pressurisent la balle.

Si l'action échoue, un des joueurs adverse récupèrera la balle, et s'ensuit ensuite une **phase de défense**. L'utilisateur va devoir essayer de récupérer la balle.

La stratégie est donc de marquer le plus de buts tout en prenant en compte la **mécanique de probabilité**. La stratégie la plus efficace sera de s'approcher le plus possible des cages pour maximiser les chances de marquer.

# 2 Mise en valeur des contraintes utilisées

## 2.1 Classes utilisées et Hiérarchie

Nous disposons au total de **huit classes**, dont le plus haut niveau de hiérarchie est composé de **trois classes** (**Character**, mère de **Player**, elle-même mère de **Goal**). Certains fichiers sont en revanche des fichiers contenant des fonctions **utilisables par toutes les classes** (utility et constant) et ne sont donc pas des classes. (Voir Diagramme UML à la section 3).

## 2.2 Méthodes virtuelles

Nous avons instauré **trois méthodes virtuelles** : les fonctions *toInfo* et *move*.

*toInfo* : Une méthode qui renvoie, selon la classe (Character, Player, Goal) toute sa description (nom, role, stat, position).

*move* : La méthode fait bouger tous les Player aléatoirement, **excepté les Goal** qui, eux, restent sur place à leur position de départ (devant les cages).

## 2.3 Surcharges d'opérateurs

Nous avons au total **trois surcharge d'opérateur dans nos classes** :

- Opérateur << dans **Character** : Il nous permet d'afficher toutes les caractéristiques de notre Character (nom, poste, stat...) en effectuant **un simple cout de l'objet**.
- Opérateur = dans **Player** : On peut avoir un pointeur direct vers le Player que l'on désire.
- Opérateur () dans **Team** : Cet opérateur nous permet de retourner le Player correspondant à partir d'un nom.

## 2.4 Conteneurs utilisés

Nous utilisons principalement les conteneurs **vecteurs et listes**. En effet, la classe **Team** contient comme attribut une **liste de Player** qu'on accède à partir de commandes STL.

Dans la classe **gameInstance**, nous avons également un attribut qui est un **vecteur de string**, qui correspond à la liste de chansons disponibles dans l'application.

## 2.5 la Bibliothèque Simple and Fast Multimedia Library (SFML)

SFML est la bibliothèque graphique que nous utilisons pour notre application. Les principales fonctionnalités que nous implémentons viennent des modules **graphique et audio**.

La documentation peut se trouver sur leur site officiel (<https://www.sfml-dev.org/documentation/2.5.1/annotated.php>) mais les principales classes que nous utilisons sont :

- Le RenderWindow : c'est elle qui permet d'afficher une **fenêtre de jeu**
- L'InputManager : un peu modifiée (pour éviter certains bugs de la bibliothèque SFML), elle nous permet d'interagir avec le jeu avec les **touches du clavier**.
- Les Sprites : pour pouvoir implémenter une image
- Les Shapes : pour représenter des formes géométriques (sans utiliser d'image)
- La Musique : une simple classe qui permet de jouer ou non une musique d'ambiance.

### 3 Diagramme UML de l'application

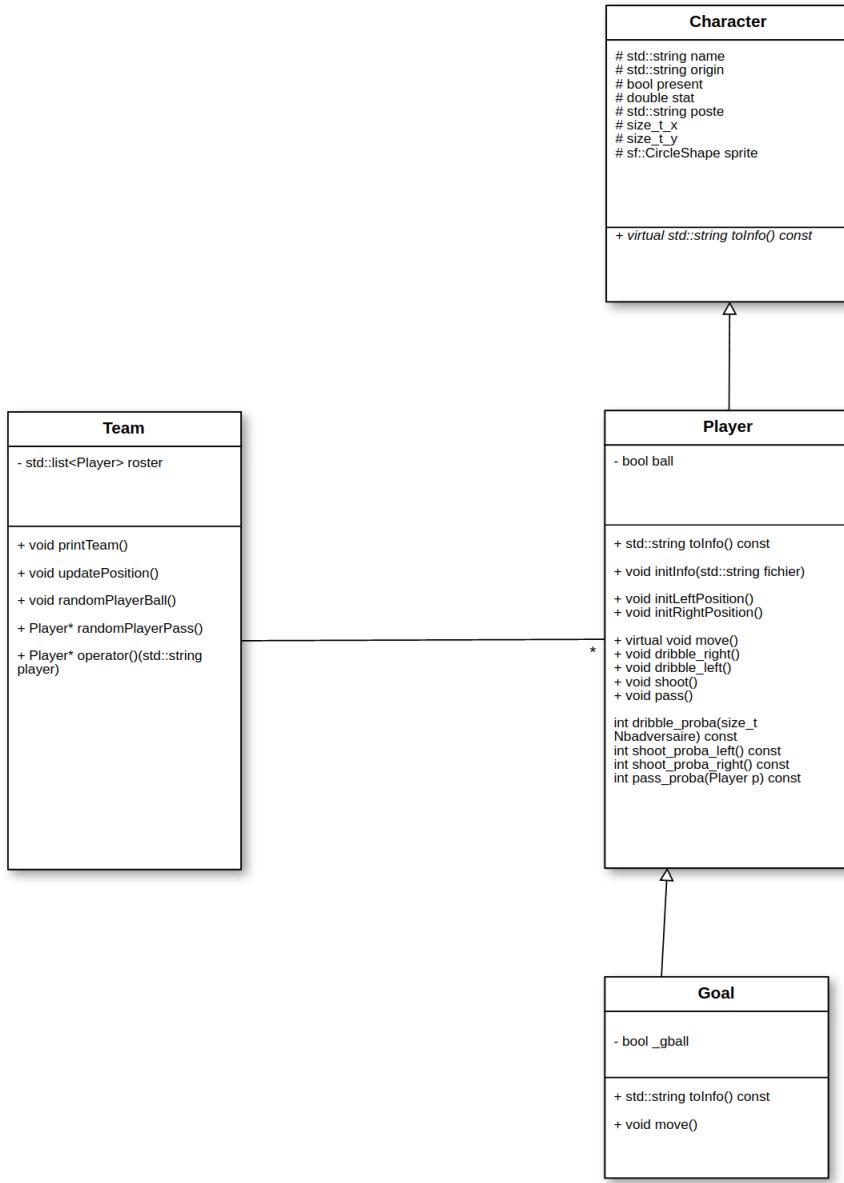


Figure 1: Diagramme du jeu

Nous avons décidé de ne représenter que les méthodes intéressantes pour notre jeu, les **constructeurs** ainsi que les **getters et setters** ne sont **pas représentés** dans les diagrammes UML suivants.

Pour faire notre jeu, nous avons besoin de plusieurs personnages qui ont des rôles différents. Comme nous pouvons le voir dans la figure 1, nos classes sont hiérarchisées et très explicites sur leur rôles. Nous avons tout d'abord une classe **Character** qui correspond à la classe mère de **Player**.

La classe **Player** est la classe qui permet d'effectuer les changements nécessaires sur chaque joueur de notre jeu. En effet, elle a des méthodes importantes pour le bon déroulement du jeu comme les méthodes d'actions du jeu (bouger, dribbler, tirer, passer).

Il était nécessaire de faire un sous-classe **Goal** car elle permet d'ajouter des méthodes spécifiques au goal qui lui ne peut bouger partout sur le terrain comme les autres joueurs. Cette classe hérite directement de **Player** car le goal fait partie des joueurs présents sur le terrain.

La classe **Team** permet de gérer l'équipe composée de plusieurs joueurs, c'est donc une association entre la classe

**Team** et la classe **Player**. Cette classe nous permet de gérer l'ensemble de l'équipe lors du déroulement du jeu. Elle permet notamment de mettre à jour les positions des joueurs.

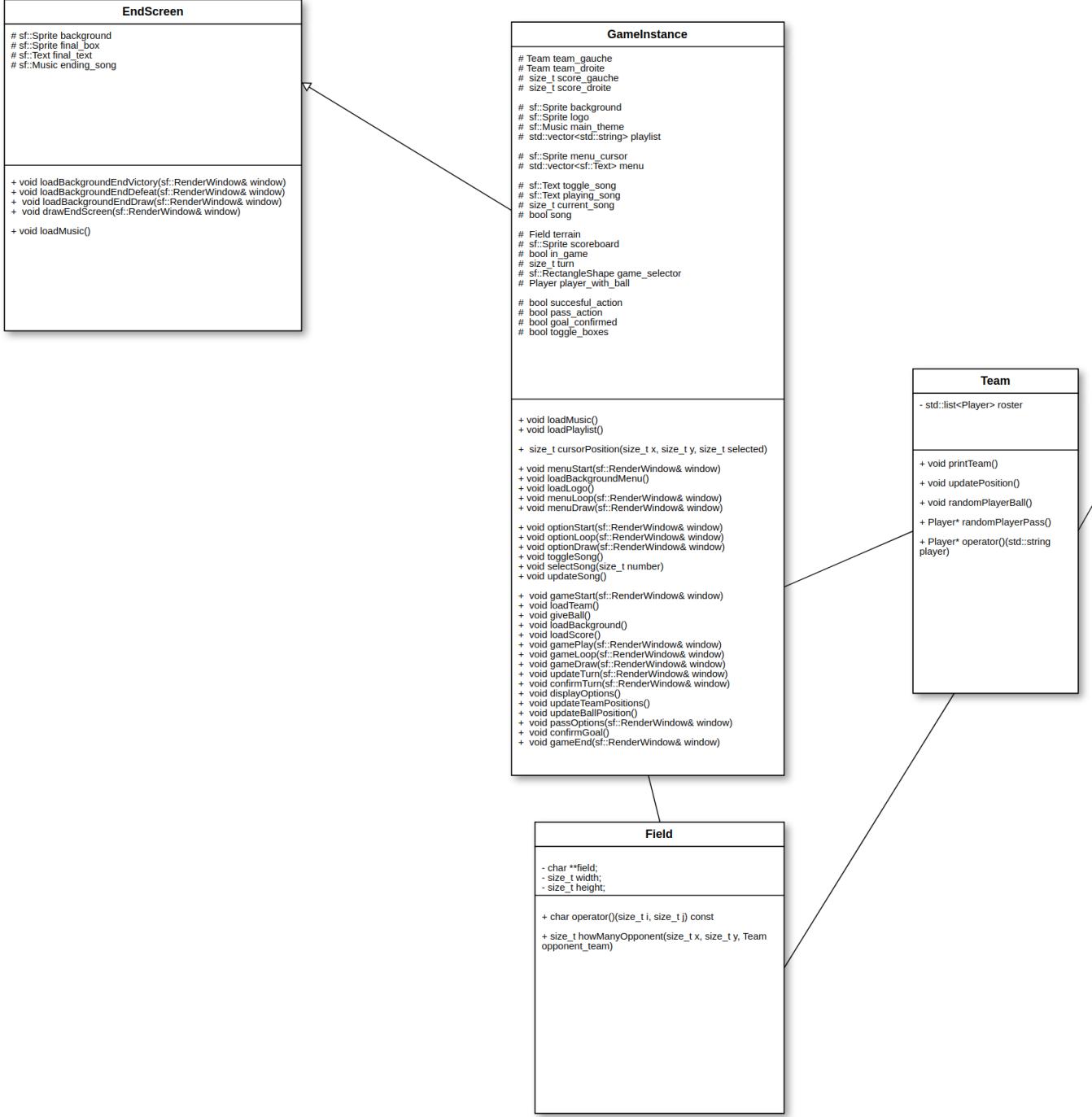


Figure 2: Diagramme du jeu et de l'interface graphique

Pour la partie graphique nous avons besoin d'une classe qui effectue les liens avec nos classes du jeu et l'affichage graphique, cette classe correspond à la classe **GameInstance**.

Cette classe est nécessaire pour gérer l'affichage graphique, les options du jeu, les représentations graphiques des joueurs mais aussi les dialogues.

Cette classe correspond à une association de la classe **Team** car on utilise les différentes équipes afin de les manipuler et les représenter graphiquement. Elle utilise également la classe **Field** pour pouvoir représenter le terrain

correctement graphiquement.

Les liens entre ces classes correspondent à des associations.

C'est une classe qui hérite directement de la classe `EndScreen`, qui elle, permet d'afficher l'écran final du jeu (c'est cette classe qui affiche si l'utilisateur gagne ou non lorsque le match s'arrête).

Dans cette classe, l'utilisation de la bibliothèque SFML est nécessaire.

Nous avons également une huitième classe qui permet de définir les touches qui vont permettre de jouer dans le jeu, voici le digramme de la classe en question:

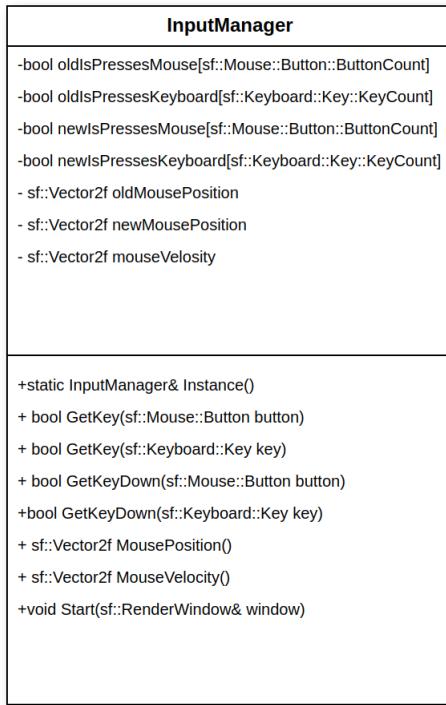


Figure 3: Diagramme UML de la classe `InputManager`

C'est une classe qui se base principalement sur les **détections d'input de la bibliothèque SFML**, cependant elle la modifie légèrement pour éviter certains bugs (par exemple la fenêtre qui s'éteint lorsque l'on bouge la souris).

## 4 Procédures d'installation et exécution

### 4.1 Procédures d'installation

L'application peut être retrouvée sur le GitHub suivant : [https://github.com/Yamacia/Project\\_World\\_Cup](https://github.com/Yamacia/Project_World_Cup) en faisant un simple `git clone`.

L'application s'exécute à partir d'un **compiler G++** (nous utilisons les versions **6.3 et 11.3.0**) ainsi que de la **bibliothèque SFML 2.5.1** (pour le compilateur **MinGW 7.3.0**). Ce dernier est téléchargeable sur le site du distributeur <https://www.sfml-dev.org/download/sfml/2.5.1/> mais elle est également comprise dans le repository GitHub téléchargé précédemment.

### 4.2 Exécution de l'application

Une fois que toutes les installations sont prêtes, dans votre terminal placez-vous dans le répertoire `Project_World_Cup/code`. Il suffit ensuite de compiler tout le code à l'aide de la commande `make compile`, puis de lancer l'exécutable `./main` dans le terminal. Si tout est bien réalisé, vous devriez arriver sur l'interface du jeu suivant :



Figure 4: Interface au démarrage de l'application

Note : Nous travaillions sur deux ordinateurs différents, et donc avions également **deux compilateurs différents** (un restait sur Windows, l'autre compilait sur Ubuntu). Nous avons donc **deux commandes différentes pour clean les fichiers .o** (respectivement make cleanw et make clean). Il faudra donc utiliser la commande appropriée selon le système d'exploitation utilisé.

Note : Les dépendances de fichiers ont été implantées à partir de **Visual Studio Code**.

## 5 Gameplay et Utilisation

### 5.1 Manipuler le menu

Le menu correspond à l'interface suivante :



Figure 5: Menu principal

Ici, vous pouvez sélectionner les différentes cases en utilisant les **flèches directionnelles de votre clavier (Haut et Bas)**. Vous pouvez soit lancer une partie, soit accéder aux options du jeu, soit quitter le jeu. Pour confirmer votre choix, il suffit d'appuyer sur **Entrée**.

### 5.2 Manipuler les options

Les options correspondent à l'interface suivante :



Figure 6: Paramètres du jeu

Dans cet écran, vous pouvez **activer ou désactiver** la musique en cours, ainsi que **choisir une des trois chansons disponibles** :

- iShowSpeed - World Cup (musique par défaut)
- K'NAAN - Waving Flag
- Amine - La Roja
- NFL Theme Song

Pour cela, vous devez sélectionner la case désirée avec les **flèches Haut et Bas**, puis changer le paramètre en

place avec les **flèches Gauche et Droite**.

Note : Il est également possible d'accéder à ces paramètres depuis le jeu.

### 5.3 Manipuler le jeu

Une fois le jeu lancé, vous arriverez sur l'interface suivante :



Figure 7: Interface de jeu, avec les boîtes de dialogue pour décrire le jeu

Il s'agit de l'interface où vous prendrez toutes vos décisions pour jouer. Vous pouvez sélectionner l'action que vous souhaitez faire parmi celles proposées (elles seront différentes selon si votre équipe possède la balle ou bien l'équipe adverse).

Vous pouvez également cacher les boîtes de dialogue avec **la touche Tab de votre clavier** afin de mieux voir le terrain et la disposition des joueurs.

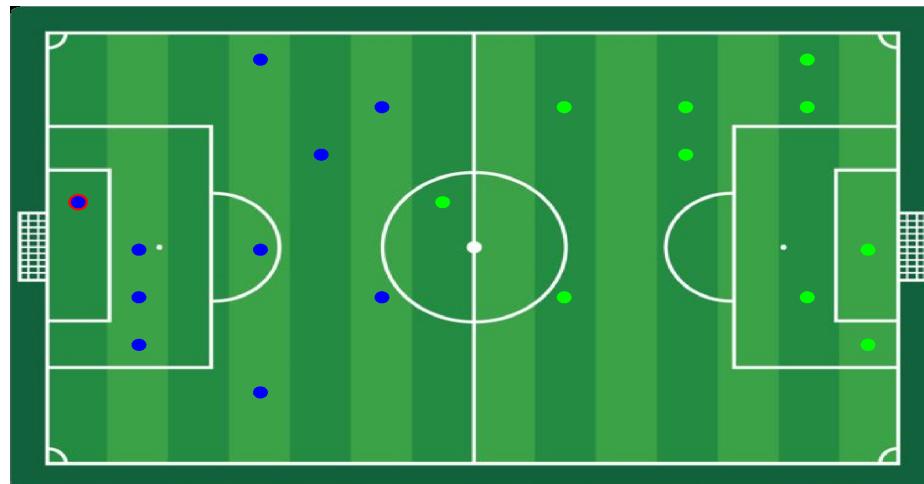


Figure 8: Interface de jeu, sans les boîtes de dialogue pour observer le terrain

Le but est maintenant de gagner la partie, selon les règles qui sont instaurées dans l'application.

### 5.4 Règles du jeu

Le jeu oppose deux équipes de **onze joueurs chacun** et dure **45 tours**. Au début du match, la balle est toujours donné à un **joueur français**. Dès qu'une équipe perd la balle, cette dernière sera donnée à un **joueur de l'équipe adverse aléatoire**. Chaque joueur possède un **potentiel stat qui lui est propre**.

Lorsque vous avez la balle, vous disposez de trois actions possibles : passer, dribbler et tirer, et dont les pourcentages

de chance de réussite **variant selon l'action**.

Passer : cette action vous donne l'option de passer la balle à **3 alliés différents**. La probabilité de réussite se calcule de la manière suivante :

$$passer = \max(100 - (distance_x)^2 - (distance_y)^2, 0) \quad (1)$$

En cas de réussite, la balle est passé au nouveau joueur et celui-ci devient le joueur que vous contrôlez

Dribbler : cette action vous permet de contourner les adversaires qui sont présents sur la même case que vous. La probabilité de réussite se calcule de la manière suivante :

$$dribbler = \begin{cases} 100 & \text{si NbAdversaire} = 0 \\ \max\left(\frac{50 * stat}{NbAdversaire}, 0\right) & \text{sinon} \end{cases}$$

Où NbAdversaire est le nombre d'adversaires sur la même case que le joueur avec la balle.

En cas de réussite, le joueur avance d'une case vers l'avant.

Tirer : cette action permet de tirer dans les cages adverses. La probabilité de réussite se calcule de la manière suivante :

$$tirer = \max\left(\frac{100 * stat}{distance_x + distance_y}, 0\right) \quad (2)$$

En cas de réussite, l'équipe marquante gagne un point et la balle est donné à un joueur aléatoire de l'équipe adverse.



Figure 9: Interface de jeu lorsqu'un joueur marque

Lorsque l'adversaire à la balle, celui-ci va essayer d'avancer jusqu'au cage, avec la **même mécanique de dribble** que celle expliquée auparavant. Lorsque celui-ci ne peut plus avancer, il va essayer de marquer.



Figure 10: Interface de jeu lorsqu'un joueur marque

Puisque vos joueurs se déplacent aléatoirement, le jeu tourne tout seul jusqu'à ce que vous récupérer la balle (soit en la récupérant, soit lorsque l'adversaire marque).

Au bout de 45 tours, le jeu s'arrête et un écran de fin apparaît en fonction de si l'utilisateur a gagné ou non. Les différents résultats sont **victoire, défaite ou match nul**.

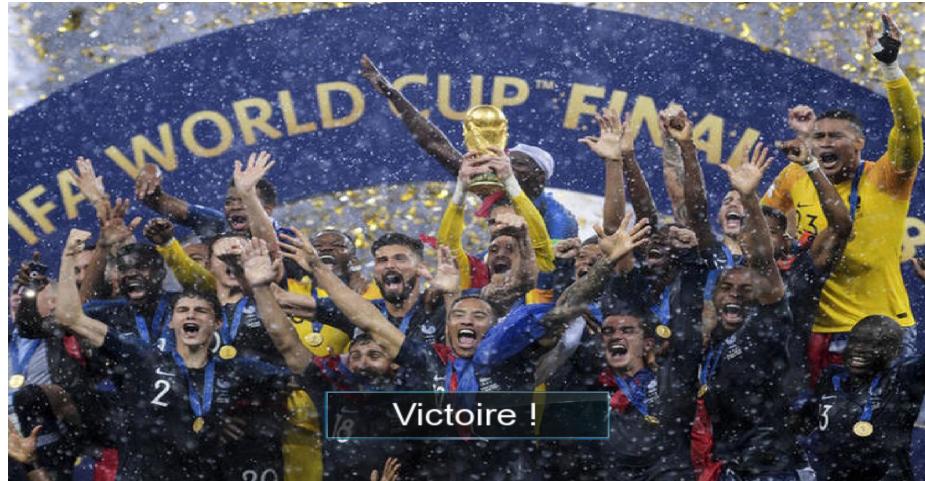


Figure 11: Interface de jeu lorsque le joueur gagne.

## 6 Nos plus grandes fiertés

### 6.1 Le rendu esthétique

Nous avons passé énormément de temps à créer des boîtes de dialogue, centrer les textes dans ces boîtes, rendre le rendu plus visuel, etc. Beaucoup de ces sprites ont même été **modifié et/ou crée sur un logiciel de montage** pour améliorer l'esthétique globale de l'application.



Figure 12: Capture d'écran du jeu. Toutes les boîtes de dialogues ont été créées à la main, notamment l'**écran des scores (notre petite fierté)**.

## 6.2 Implémentation d'une musique

Il ne s'agit là que de l'implémentation d'une musique à l'aide de la bibliothèque son SFML, mais nous sommes quand même contents d'avoir réussi à mettre en place une musique **en boucle et à volume adéquat**. C'était aussi un petit plus d'avoir une musique en rapport avec la Coupe du Monde.

## 6.3 Sélection de musique

Comme nous voulions implémenter plusieurs musiques, il nous fallait un **écran de sélection de musique** et c'est ce que nous avons réussi à implémenter dans un onglet parallèle au jeu.



Figure 13: Écran de sélection de musique

Cela nous permettait de jouer au jeu avec l'ambiance désirée, et ça nous évitait aussi de jouer en boucle la musique de iShowSpeed qui, au bout d'un moment, donne malgré tout mal à la tête.

## 6.4 Basculer d'un écran à l'autre

Ce n'était pas toujours évident de faire en sorte que les différents menus (Options et Jeu par exemple) communiquent entre eux de manière à ce qu'on puisse **basculer d'un écran à l'autre**. Mais cette communication permet d'avoir une transition fluide pour tout ce qu'on veut afficher (par exemple l'écran de fin.)

## 6.5 Le gameplay en tour par tour

Comme l'affichage (et donc le jeu) tourne en boucle et surtout **en continu**, il fallait fixer le temps à un moment t pour conserver les variables jusqu'à ce que l'utilisateur prenne une décision, pour pouvoir non seulement **mettre à jour les variables** pour le moment t+1, mais surtout pour que l'utilisateur puisse observer **quel est le résultat de la décision qu'il vient de prendre (Réussite ou Echec)**. Cette mécanique est réalisée par la méthode *confirmTurn* qui bloque le tour jusqu'à ce que l'utilisateur confirme le passage au moment suivant.

## 6.6 Cacher l'interface graphique

Évidemment, c'est difficile d'observer le terrain lorsqu'il y a dénormes boîtes de dialogue qui cachent le terrain. Instaurer un booléen pour les afficher ou non a permis de mieux visualiser ce qu'il se passe sur le jeu.

# 7 Aller plus loin

## 7.1 Ajouter plus d'équipes

Jusqu'à présent, le jeu n'oppose que deux équipes, **la France et le Portugal**. De plus, l'utilisateur joue forcément la France. Une amélioration serait de proposer une **sélection d'équipe pour le match**.

## 7.2 Choisir ses joueurs

De la même manière, des joueurs fixes sont déployés sur le terrain, directement initialisés dans le code. Mais les fichiers textes utilisés pour initialiser nos joueurs comportent plus que onze joueurs, et on pourrait implémenter un **écran de sélection de joueurs pour le match**.

## 7.3 Mécanique de passe

L'algorithme de passe comporte de nombreux défauts. Son principe est de proposer trois joueurs de la même équipe aléatoire, puis de faire la passe à l'un d'entre eux. Cependant, l'algorithme peut proposer **plusieurs fois la même personne**, voire même proposer la passe à **soi-même**. Un meilleur algorithme est nécessaire pour rendre la passe plus utile.

## 7.4 Savoir à qui on fait la passe

L'algorithme de passe nous donne les personnes à qui on peut faire la passe, cependant on ne sait pas à **quel sprite sur l'interface ils correspondent**. Il faudrait une astuce pour déterminer à quel sprite correspond chaque joueur (avec un contour coloré ou un vrai sprite par exemple).

## 7.5 Méthodes de calcul de probabilités

Les formules pour déterminer la probabilité de réussite sont aussi **assez simplistes**. De plus, les passes échouent rarement, tandis que le dribble échoue assez souvent. La différence de performances entre les joueurs est **minime également puisque les stats sont relativement proches**. Il faudrait rééquilibrer les équations pour que ne pas privilier une stratégie sur une autre.

## 7.6 Intelligence de l'adversaire

Le jeu de l'adversaire détaillé précédemment est trop simpliste et ne se sert pas de toutes les actions proposées à l'utilisateur. Il faudrait donc un jeu plus intelligent mais cela est compliqué à implémenter.

## 7.7 Mécanique de défense

Comme expliqué précédemment, lorsque l'adversaire a la balle, nos joueurs se contentent de **se déplacer aléatoirement** et potentiellement bloquer l'adversaire sur le chemin. Il serait préférable d'instaurer d'autres options de blocage, comme le tacle par exemple.