# Poly-Head Soccer


Skhiri Youssef
Minthe Ibrahim

# I) Introduction

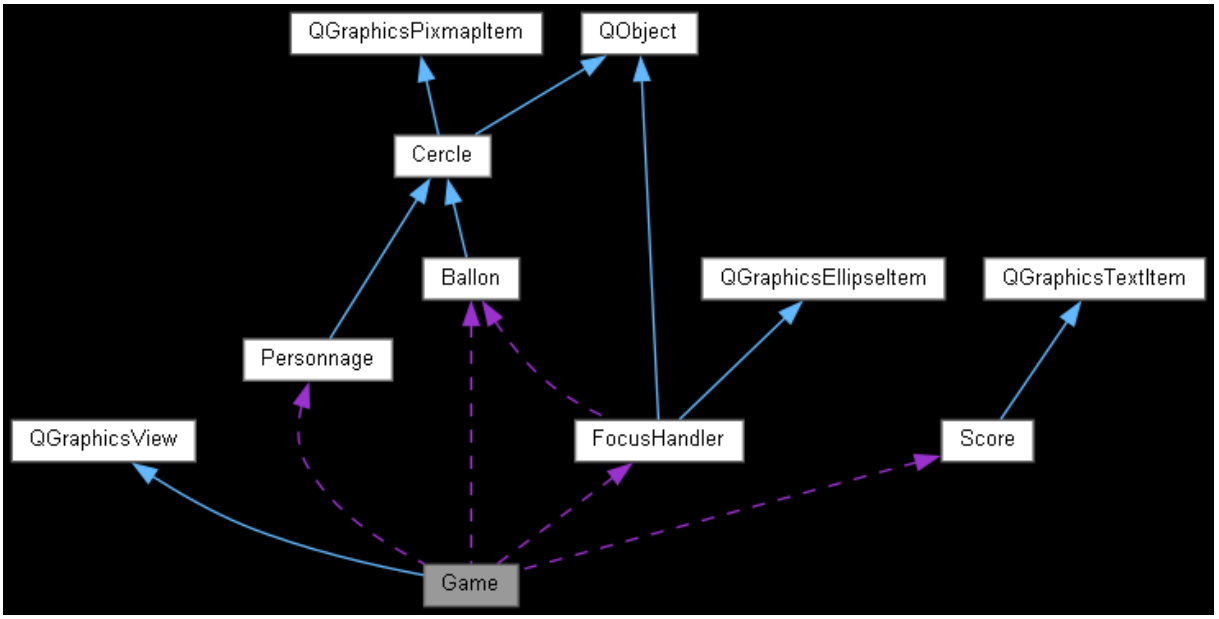
Dans le cadre de notre projet de fin de semestre en C++, nous avons décidé de recréer le jeu HeadSoccer. Notre nouveau jeu s'appelle donc Poly-Head Soccer. L'idée de base consiste à marquer des buts dans un contre-un et ce joue à 2. Chaque joueur est contrôlable grâce aux touches du clavier.

Nous avons décidé de reprendre ce jeu car nous y avons pas mal joué durant notre enfance mais de plus il rentre complètement dans le thème. En effet, il s'agit d'un jeu de foot et les personnages proviennent de différents pays ce qui renvoie directement à la coupe du monde. De plus, nous avons décidé de reprendre le thème du projet à la lettre en appliquant une coupe ou une couleur de cheveux différente à chacun de nos personnages. Enfin quand on parle de coupe du monde, on pense directement au foot et donc à la rivalité Messi Ronaldo. Ainsi, vous pourrez incarner l'une de ces grandes stars dans la quête de tous les personnages disponibles.

Notre jeu est facile à jouer et nos personnages aux coupes originales vous donneront envie de refaire une partie !!!

# II) Description des classes

Voici la hiérarchie des classes :




Une documentation plus détaillée avec un diagramme de classe est disponible dans le fichier index.html dans le dossier doc.

**Classe Cercle :**

Cette classe définit un objet "Cercle" qui hérite des propriétés et méthodes de la classe QGraphicsEllipseItem et QGraphicsPixmapItem . Elle contient des méthodes pour obtenir et définir la vitesse, la masse, et gérer les entrées clavier et déplacer l'objet. Elle utilise également les fonctionnalités de la bibliothèque QObject de Qt et est définie comme une classe abstraite avec des méthodes virtuelles pures pour être implémentée dans les classes dérivées.

La classe définit un constructeur qui est utilisé pour initialiser l'objet Cercle. Dans ce constructeur, les coordonnées de vitesse et d'accélération sont initialisées, le rayon et la masse du cercle sont définis, un timer est créé et connecté à la fonction move() de l'objet Cercle. Le timer est ensuite démarré pour exécuter la fonction move() toutes les 10ms. Cela permet de faire bouger le cercle à l'écran. Enfin il y a une variable globale "count" qui est initialisée à 0. Cette dernière est partagée par toutes les classes héritant de Cercle afin de simuler le temps.

**Classe Ballon :**

Cette classe définit un objet "Ballon" qui hérite des propriétés et méthodes de la classe "Cercle". Elle contient également des méthodes pour gérer les entrées clavier, déplacer l'objet, et jouer des sons

spécifiques lorsque des événements spécifiques se produisent. Elle utilise également les fonctionnalités de la bibliothèque QObject et QMediaPlayer de Qt.

Cette classe est responsable de simuler la physique d'une balle dans un jeu, comme son mouvement et la détection de collision. Cette classe comporte un vecteur de paires appelé "vit" qui suit la vitesse de la balle, et un vecteur de paires appelé "acc" qui suit l'accélération de la balle. La classe utilise également un objet QMediaPlayer pour jouer des effets sonores. La fonction move() met à jour la position de la balle en fonction de sa vitesse et de son accélération et vérifie également les collisions avec les bords de l'écran ou avec d'autres objets. La fonction KeyPressEvent est déclenchée lorsque l'utilisateur appuie sur la barre d'espace et réinitialise la position de la balle.

**Classe Personnage :**

Cette classe définit un objet "Personnage" qui hérite des propriétés et méthodes de la classe "Cercle". Elle contient des méthodes pour gérer les entrées clavier, déplacer l'objet, et stocker des informations spécifiques au personnage telles que les touches de déplacement (haut, bas, gauche, droite), le nom, l'état de jeu, et les informations de prix. Elle utilise également une std::map pour stocker les informations de prix des personnages, avec des méthodes statiques pour accéder et mettre à jour cette map. Il y a aussi une surcharge d'opérateurs pour sortir le contenu de la classe dans un flux de sortie.

Le constructeur de la classe Personnage prend 4 paramètres : left, up, right et down. Il initialise les variables membres _left, _up, _right et _down avec les valeurs de ces paramètres. Il initialise également les variables membres radius et mass avec des valeurs prédéfinies. Il hérite du constructeur de la classe de base Cercle. La fonction KeyPressEvent gère les entrées clavier pour déplacer le personnage en utilisant les constantes MAX_VX et MAX_VY pour limiter la vitesse du personnage. La fonction move gère la mise à jour de la position du personnage en fonction de la physique du mouvement, comme la friction et les collisions avec d'autres objets. Le code utilise également des fonctions de la bibliothèque C math et des bibliothèques C++ standard pour lire et écrire des fichiers, ainsi que les bibliothèques Qt pour l'affichage graphique.

**FocusHandler :**

La classe "FocusHandler" hérite de la classe QObject et QGraphicsEllipseItem. Elle contient une liste de pointeurs vers des objets Personnage, ainsi qu'un pointeur vers un objet Ballon. Elle possède une méthode "addPersonnage" qui permet d'ajouter un Personnage à la liste, une méthode "setBallon" qui permet de définir un Ballon pour l'objet, et une méthode keyPressEvent qui est utilisée pour gérer les événements de touche. Elle permet de gérer le focus des events clavier sur le ballon ou sur un joueur.

**Game :**

La classe Game est une classe qui hérite de QGraphicsView. Elle contient des éléments graphiques tels que des Personnages, un Ballon, un FocusHandler, un Score et un QGraphicsScene. Elle possède des méthodes pour afficher un menu principal, un menu pour sélectionner des personnages, un menu pour acheter des personnages, jouer, réinitialiser et quitter le jeu. Il y a également une méthode pour créer un fichier CSV à partir d'une liste de joueurs. Elle utilise également des signaux et des slots pour gérer les interactions avec l'utilisateur.

Le constructeur de la classe Game crée une nouvelle scène de jeu, définit les dimensions de cette scène et l'affiche à l'écran via QGraphicsView. Chaque nouveau menu du jeu est représenté par un slot .

**Joueur :**

La classe Joueur est une classe qui représente un joueur dans le jeu. Elle a plusieurs caractéristiques comme un solde, un nom et une équipe de personnages. Elle a des fonctions pour obtenir ces caractéristiques, pour les définir et pour acheter des personnages. Elle a également une fonction qui permet de mettre à jour un fichier CSV servant de sauvegarde entre parties. La classe contient également un pointeur vers un personnage actif et une liste de joueurs. Il a également des fonctions pour effacer la liste de joueurs et pour réinitialiser les données.

Le constructeur Joueur prend en entrée le solde, le nom et l'équipe d'un joueur, et initialise les variables d'instance en conséquence. Il ajoute également le joueur créé à la liste des joueurs existants. Le destructeur ~Joueur supprime le joueur de la liste des joueurs existants. La méthode statique clearListeJoueur supprime tous les joueurs de la liste des joueurs existants. La méthode statique updateFile met à jour le fichier CSV des joueurs avec les informations les plus récentes sur les joueurs existants. La méthode buyPersonnage permet à un joueur d'acheter un personnage. La méthode statique reset réinitialise les joueurs à partir d'un fichier CSV par défaut.

**Button :**

La classe Button est un bouton graphique qui hérite de QObject et de QGraphicsRectItem. Elle contient des constructeurs pour créer un bouton avec un nom donné. Elle définit également des méthodes pour gérer les événements de la souris (clic, survol, sortie de survol), un pointeur vers un élément de texte pour afficher le nom du bouton, un entier pour stocker l'index du bouton et une fonction pour récupérer le texte du bouton. La classe définit également un signal "clicked" qui est émis lorsque le bouton est cliqué.

Le constructeur permet de dessiner un rectangle et un texte au centre de celui-ci. La méthode "mousePressEvent" permet d'émettre un signal "clicked" lorsque l'utilisateur clique sur le bouton. Les méthodes "hoverEnterEvent" et "hoverLeaveEvent" permettent de changer la couleur du bouton lorsque la souris entre ou quitte le bouton.

**Score :**

La classe Score est une classe qui hérite de QGraphicsTextItem, qui est utilisée pour afficher du texte graphiquement. Elle contient des méthodes pour augmenter les scores des joueurs 1 et 2, ainsi que des méthodes pour obtenir les scores des joueurs 1 et 2. Elle contient également des variables privées pour stocker les scores des joueurs 1 et 2.

La classe Score a un constructeur qui initialise les scores à 0 et affiche les scores à l'écran en utilisant la méthode setPlainText. Il y a deux méthodes pour augmenter les scores des joueurs, une pour le joueur 1 et une pour le joueur 2. Il y a également deux méthodes pour récupérer les scores des joueurs. La classe utilise les propriétés de QGraphicsTextItem pour formater le texte affiché.

# III) Codage du mouvement

Afin de simuler au mieux les mouvements d'un ballon, nous avons utilisé les principes de la physique pour simuler les forces qui agissent sur le ballon, comme la gravité, la friction, et les collisions.

Dans la méthode move() de la classe Ballon, il y a plusieurs étapes qui sont utilisées pour simuler les mouvements du ballon :

1. **Incrémenter un compteur**. Cela permet de suivre le temps écoulé depuis que le ballon a été lancé.
2. **Calculer le temps**. Le temps est utilisé pour calculer les accélérations et les vitesses du ballon. Il est calculé en divisant la valeur du compteur par 100.
3. **Appliquer la friction**. La friction est utilisée pour simuler la résistance que le ballon rencontre lorsqu'il roule sur le terrain. Elle est appliquée en multipliant la vitesse actuelle du ballon par un facteur de 0.0075.
4. **Appliquer la gravité**. La gravité est utilisée pour simuler la force qui attire le ballon vers le bas. Elle est appliquée en ajoutant une accélération de 0.10 au mouvement vertical du ballon.
5. **Mise à jour de la physique du ballon**. La physique du ballon est mise à jour en utilisant l'accélération et la vitesse calculées précédemment. La vitesse est incrémentée par l'accélération multipliée par le temps.
6. **Vérifier les limites de vitesse**. Il est vérifié que la vitesse du ballon ne dépasse pas les limites maximales définies pour la vitesse horizontale (MAX_VX_BALL) et verticale (MAX_VY_BALL).
7. **Mettre à jour la position du ballon**. La position du ballon est mise à jour en utilisant la vitesse calculée précédemment.
8. **Gérer les collisions**. Si la condition if (y()<= 0 || y()>= scene()->height()-200) est vraie, cela signifie que le ballon est sorti des limites de hauteur de la scène. Pour simuler une collision avec le plafond ou le sol, la vitesse verticale du ballon est inversée en multipliant par -0.98. Cela signifie que si le ballon se déplaçait vers le haut, il se déplacera maintenant vers le bas, et inversement. De plus, pour maintenir le ballon dans les limites de la scène, la position du ballon est également mise à jour pour qu'il soit collé au plafond ou au sol. Ensuite, les mêmes vérifications sont effectuées pour les limites de largeur de la scène, en utilisant les conditions if(x()>= scene()->width()-250 && y()<=scene()->height()-400) et if(x()<=175 && y()<=scene()->height()-400). Ces conditions vérifient si le ballon est sorti des limites de largeur de la scène et, si c'est le cas, simulent une collision avec les murs en inversant la vitesse horizontale du ballon.

Le principe est assez similaire dans le cas d'un Personnage. Dans la méthode Personnage::move(), on vérifie si le Personnage entre en collision avec d'autres objets de la scène. Si c'est le cas, on utilise un cast pour vérifier si l'objet en collision est un autre Personnage ou un objet Ballon. Si c'est un autre Personnage, la fonction retourne immédiatement sans rien faire. Si c'est un objet Ballon, on met à jour les vitesses du ballon en utilisant la conservation de l'énergie cinétique pour définir les vecteurs.

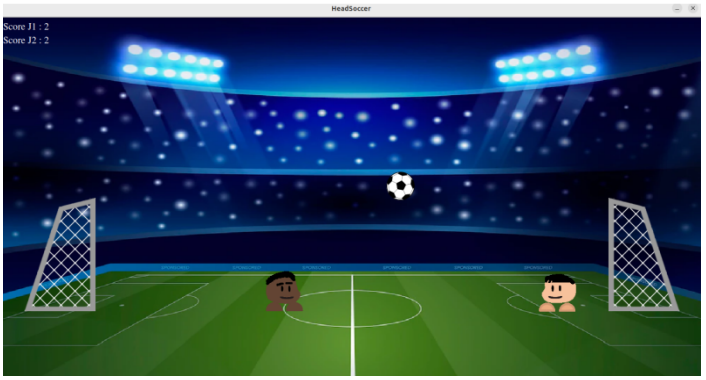
# IV) Gameplay

Lorsque vous lancez le jeu, vous avez le choix entre 3 possibilités :

- Le bouton Start qui permet de passer au menu de sélection des joueurs
- Le bouton Reset qui permet de réinitialiser l'avancement des 2 comptes (vous n'aurez à ce moment plus qu'un personnage dans votre équipe et un solde de 100 points)
- Le bouton Quit pour quitter le jeu

En appuyant sur Start, vous passez donc au menu de sélection des joueurs. Dans ce menu, vous pourrez sélectionner les joueurs disponible dans votre équipe où appuyer sur le bouton Market afin d'aller acheter de nouveaux personnages. Dans ce menu, vous retrouverez, pour les 2 comptes, la liste des joueurs encore disponible et leur prix.

Une fois que les deux joueurs auront fait leur choix, vous pourrez appuyer sur Play et commencer le match. Un match se termine lorsque l'un des 2 joueurs marque 3 buts. A chaque but, le ballon sera remis en l'air au centre du terrain. Le compte lié au personnage gagnant recevra alors 100 points.




Afin d'améliorer l'expérience de jeu, nous avons rajouté des petits sons. Il y a donc une musique d'intro, des coups de sifflet en début et fin de match et enfin un petit son à chaque but marqué.

Enfin, dans les rares cas où le ballon se coince, nous avons rajouté un « bouton reset » qui permet de remettre la balle au centre. Il vous suffit d'appuyer sur la touche espace.

# V) Conclusion

Nous avons pris énormément de plaisir à coder notre jeu. Le voir évoluer de jour en jour était un réel plaisir. De plus, nous sommes fières du résultat final car ce n'était pas évident au début pour nous notamment lorsque nous avons dû coder une sorte de « gravité » dans le jeu et réaliser un système de sauvegarde. Le thème nous a aussi directement parlé même si choisir qu'elle jeu faire s'est révélé plus difficile qu'attendu. Enfin, nous vous souhaitons de prendre un maximum de plaisir à y jouer !!!!