Compte-rendu LO43

Projet : Jeu de Basket (LO43 Shot)

Abdelhamid KACIMI – Aimeric MONTANGE – Timothée GUILLEMAILLE
LO43 – P19

Établissement :
Université de Technologie de Belfort-Montbéliard
Professeurs TP :
Jean-Charles CREPUT & Abdelkhalek MANSOURI
Responsable UV :
Jean-Charles CREPUT

Tables des matières

1. Réalisation du projet

Dans le cadre de l'UV LO43, un projet doit être réalisé en C++ ou en Java utilisant les principe d'analyse/conception orientée objet. En effet, notre projet doit présenter certaines notions vues en LO43 telles que les classes, la réutilisation avec l'héritage et l'agrégation, les listes, le polymorphisme, les templates, etc. Nous avons décidé de coder ce programme en C++ avec le framework QT. Il s'agit ici d'un jeu de basket-ball où il est question de réussir plus de lancer que son adversaire (5 tirs par user, 1 tir par joueur de chaque équipe avec 5 joueurs par équipe). Pour cela, des données numériques sont présentes afin de pouvoir calculer la réussite ou non du lancer, on retrouve ainsi la précision et la puissance de chaque joueur ainsi que l'impact que le coach à sur chaque joueur de son équipe.

Pour en arriver à cette idée, nous avons suivi la méthode du Génie Logiciel vue en cours de LO43 et de GL40 : le cycle en V. Nous avons donc commencé par établir un cahier des charges, par analyser les besoins puis nous avons débuter la conception et le codage

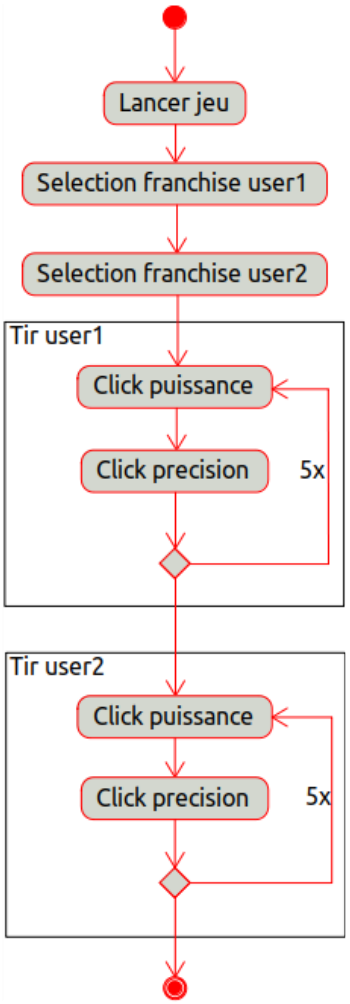
2. Cahier des charges

Notre principale contrainte est l'utilisation des notions vues en LO43. C'est ainsi que ce jeu nous permettra d'utiliser l'héritage (les joueurs et les coachs sont des personnes), l'agrégation (chaque franchise/équipe est composée de 5 joueurs et d'un coach). Les notions de liste et de template interviennent aussi puisque les joueurs et le coach sont présents sous la forme d'une liste de personnes.

Ici, notre programme sera un jeu opposant deux utilisateurs (users). Le but de ce jeu sera de réussir plus de lancer que son adversaire. Chaque users aura 5 tirs à réaliser avec 5 joueurs différents. On déterminera la réussite ou non du tir avec des notes de précision et de puissance que chaque joueur possède ainsi qu'avec une note d'impact que le coach de l'équipe apporte à chaque joueur de son équipe. Les utilisateurs pourront sélectionner une franchise (équipe) parmi les 30 franchises présentes en NBA. L'interface devra être agréable pour l'utilisateur, ergonomique et intuitive.

3. Analyse

a. Diagramme d'action



Au moment de la rédaction de ce rapport intermédiaire, le diagramme d'action mettant en avant le déroulement de l'application se rapproche grandement de l'image ci-contre. L'user commence ainsi par lancer le jeu, puis une phase de sélection des franchises a lieu. Ensuite, le jeu débute réellement avec les 5 tirs de l'user1 qui tentera d'obtenir les meilleurs résultats pour la puissance et la précision de chacun de ces tirs avant que l'user 2, à son tour, shoote 5 fois avec les jauges de puissance et de précision. Chaque tir seront traités afin de déterminer si le lancer est réussi ou non.

b. Déduction

Nous pouvons d'ores et déjà dégager quelques fonctions du logiciel à l'aide de ce diagramme. On devra retrouver une façon de stocker toutes les données relatives à chacune des franchises (nom, ville, joueurs, coach, notes, etc.). On sait que l'on retrouvera plusieurs classes différentes (Franchise, Coach et Player).

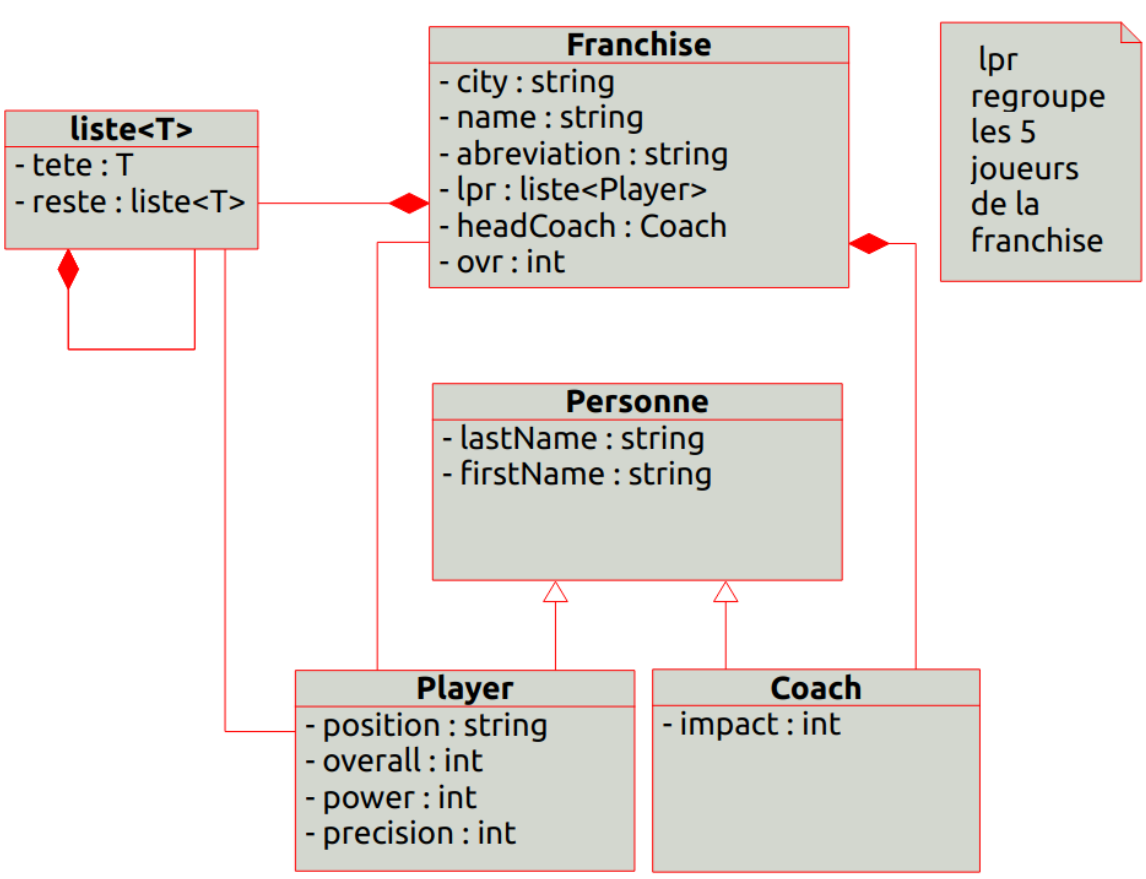
4. Conception

a. Nos solutions

Afin de satisfaire les précédents éléments, nous avons décidé de choisir certaines solutions. Dans un premier temps, il est plutôt évident que des classes Franchise, Player et Coach doivent être présente. Après cela, nous avons décidé de faire hériter Player et Coach d'une autre classe Personne. La classe Franchise contient ainsi 5 Player et 1 Coach mais nous avons ici aussi décider de faire intervenir une nouvelle classe : Liste. Cela nous permettre de regrouper ces 5 Player dans une liste<Personne>. Pour le stockage des différentes données se rapportant aux équipes, aux joueurs, aux notes et autres, nous avons fait le choix de regrouper ces derniers dans un fichier .txt. Ces données seront séparées par des espaces et des sauts de ligne. Cette solution-là permet une lecture simple et une compréhension rapide des données. L'interface graphique sera, en partie, réalisée à l'aide de l'UI de QT. Les signaux et les slots faciliteront l'affichage dynamique de toutes les données, du score, etc. De plus, l'UI et son système de gestion des ressources (fichier .qrc) permet de gérer bien plus simplement tout cela pour obtenir un résultat optimal et intuitif de ce que l'utilisateur aura devant ses yeux.

b. Diagramme de classe

À partir des choix que nous avons énuméré précédemment, nous avons construit ce diagramme de classes UML qui permet d'avoir une vision plus claire de ce que nous proposons pour la réalisation. On remarque donc bien la présence de l'héritage et de l'agrégation. Les listes sont, elles aussi, utilisées.


Franchise
- city : string
- name : string
- abreviation : string
- lpr : liste<Player>
- headCoach : Coach
- ovr : int
lpr regroupe les 5 joueurs de la franchise
liste<T>
- tete : T
- reste : liste<T>
Personne
- lastName : string
- firstName : string
Player
- position : string
- overall : int
- power : int
- precision : int
Coach
- impact : int

5. Les classes hors-UI

a. Franchise

Dans un jeu tel que le nôtre, il semble évident qu'une classe représentant les équipes avec lesquelles les 2 users s'affronteront. Ainsi, on retrouve cette classe franchise qui aura été la première implémentée. Dans celle-ci, on retrouve plusieurs attributs : *city*, *name*, *abreviation*, *lpr*, *headCoach*, *ovr*. Les trois premières sont des string et permettent de mettre un nom sur une franchise. Les attributs *city* et *name* permettent d'obtenir le nom « réelle » de la franchise. Le nom de chaque franchise est composé de la ville de celle-ci ainsi qu'un « surnom ».

Quant à elle, *abrevation* est, comme son nom l'indique, l'abréviation du nom en 3 lettres de la franchise. Cette abréviation s'affichera notamment dans le tableau des scores. Par exemple, les users pourront jouer avec les Toronto Raptors et TOR sera l'abréviation présente pour le score.

Le dernier attribut de Franchise est *ovr* (pour overall) qui représente simplement la note générale de la franchise en question.

Il manque toutefois ce qui est peut-être le plus important dans une équipe : les joueurs et le coach qui composeront l'équipe. Les joueurs sont regroupés dans une *liste* que l'on spécifiera plus loin dans le rapport. Cette liste est donc une *liste de Player* (spécifier plus tard aussi). Le coach de l'équipe est lui instancié à part et celui-ci est de type *Coach*.

```
class Franchise {
public :
    string city;
    string name;
    string abreviation;
    liste<Player> lpr;
    Coach headCoach;
    int ovr;
```

b. Personne

Dans cette configuration-là, nous sommes en présence de deux types de personnes, à savoir : un coach et cinq joueurs. Ces personnes sont composées de deux string qui représentent le prénom de la personne (*firstName*) et son nom (*lastName*)

```
class Personne {
protected:
    string firstName;
    string lastName;
```

c. Player

```
class Player : public Personne {
    int overall;
    float power;
    float precision;
```

La classe *Player* est créé par héritage de la classe *Personne*. On retrouve donc les deux attributs précédents auxquelles nous avons ajouté un int *overall* qui représente la note générale du joueur ainsi que deux float, *power* et *precision*, qui représentent respéctivement les notes de puissance et de précision de chaque joueur. La note *overall* ne sera là qu'à titre indicatif afin de donner au joueur une idée du niveau du joueur alors que les deux float seront utilisé dans les calculs pour valider ou non un shoot.

d. Coach

```
class Coach : public Personne{
    int impact;
```

Les coachs de chaque équipe auront aussi leurs noms et leurs prénoms, hérités de la classe *Personne.* Seul un int s'ajoute à cela, nommé *impact,* qui représente simplement l'impact d'un coach sur chaque tir de son équipe. On retrouvera donc cette valeur dans les calculs plus tard.

e. Liste

```
template<typename T>
class liste {
    T tete;
    liste* reste;
```

Évoquée précédemment, la classe *liste* sera utilisé pour regrouper sous une même entité les joueurs d'une même équipe mais sera également utilisées plus tard afin de stocker les valeurs enregistrer lors des phases de tirs. Comme la classe *liste*

étudiée en cours, celle de ce programme est composé d'un attribut *tete* qui verra son type changer puisque nous avons ici la présence de *template.* L'autre attribut de la classe *liste* est *reste* qui est un pointeur sur une *liste.*

6. Déroulement par fenêtre

a. MainWindow

La première fenêtre du programme est *MainWindow.* Cette derniere présente trois boutons. Le premier est *newGameButton* qui lance simplement le jeu en passant à la fenêtre suivante. Le deuxième est *rulesButton* qui, lorsque l'on clique dessus, cache les trois boutons pour afficher un groupBox composé d'un texte présentant les règles ainsi qu'un bouton afin de cacher ce même groupBox et afficher à nouveau les trois premiers boutons. Le dernier de ces boutons est *exitButton* qui va simplement quitter le jeu après un clique.

b. TeamChoice

La fenêtre suivante est *TeamChoice.* On y retrouve deux *QListWidget* qui permettre de choisir parmi 30 franchises classées par ordre alphabétique. Au-dessus de ces deux éléments, deux *QLabel* affichant les logos des équipes sélectionnées. Entre ces deux parties, il y a un bouton *playButton et* un affichage d'informations concernant les franchises sélectionnées: Noms et notes générale des équipes, noms, prénoms et note des joueurs ainsi que les noms, les prénoms et les impacts des coachs de chaque équipe. Ces informations se mettent à jour si nécessaire, c'est à dire si l'item sélectionné dans un *QListWidget* change.

Pour faire ainsi, nous avons écrit un fichier .txt (*docBDD.txt*) regroupant toutes les informations nécessaires concernant chacune des 30 franchises.

on_listJ1_itemClicked & on_listJ2_itemClicked

Lorsqu'il y a un changement d'item sélectionné, plusieurs choses se passent dans les methodes *on_listJ1_itemClicked* et *on_listJ2_itemClicked*. Dans un premier temps, l'ouverture du document en lecture seulement a lieu ainsi que la déclaration de plusieurs string et int pour afficher les données. On retrouve aussi un string *ligne* qui va permettre de parcourir le fichier.

À l'aide d'un switch et de l'indice de l'item sélectionné, on va pouvoir afficher le logo de la franchise correspondante. Ce même indice va nous permettre de nous placer au début de la première ligne correspondant à la franchise sélectionnée (chaque franchise occupe huit lignes dans le document .txt). À partir de là, les éléments nécessaires au bon affichage des informations de la franchise sont stockés dans les string et les int que nous avons précédemment déclaré. *setText* va ensuite changé chaque texte des labels affichant les données. Rappelons qu'une conversion en QString est nécessaire pour cela.

Pour terminer, on referme simplement le fichier et on stocke la valeur de l'indice de la franchise dans un attribut de la classe *TeamChoice.*

on_playButton_Clicked

Lorsque l'on clique sur le bouton PLAY (*playButton*), la fenetre se ferme afin d'ouvrir la suivante mais plusieurs évenements ont lieu avant cela. En effet, on doit

stocker les données « définitives » concernant les équipes finalement choisies pour la phase réelle de jeu. Pour cela, nous ouvrons à nous le document .txt. On récupère toutes les informations concernant la franchise de l'user1 dans un objet de type *Franchise*. On instancie donc les cinq joueurs que l'on placera dans une liste. Le coach est lui aussi instancié, de même pour *city, name* et *abreviation.*

On réitère la même série d'opération pour la franchise de l'user2 qui réutilisera les mêmes variables pour la récupération des informations.

On déclare ensuite une fenêtre de type *GameWindow* (présenté à la suite) avec plusieurs paramètres. Cela nous permet d'afficher certaines choses dès l'ouverture de cette nouvelle fenêtre. On effectivement besoin d'afficher dès le début les abréviations des deux franchises ainsi que les informations concernant le premier tireur.

c. GameWindow

La dernière fenêtre est GameWindow qui est ni plus ni moins la fenêtre les deux users vont effectuer leurs lancer et jouer. Plusieurs éléments sont présents ici. On a la présence d'un *groupBox* formant le tableau des scores. Ce dernier est composé de deux *QLabel* affichant les abréviations des deux franchises. Un joueur portant un maillot blanc et une ligne affichant les données du joueur tirant sont présentes aussi. Cette ligne est écrite sur fond blanc et ce fond blanc sera remplacé par un fond noir lorsque l'user2 jouera. Il en sera de même pour la couleur du maillot du joueur. Sur la partie basse, à droite, on retrouve ce qui est certainement le plus important ici, il s'agit des barres de tir et du bouton associé. En effet, les users devront cliquer à deux reprises sur ce bouton, en tentant de stopper le changement de valeur des barres sur la valeur la plus élevée possible. Un calcul aura lieu afin de déterminer si un le tir est réussi ou non.

on_continueButton_clicked

Le déroulement des éventements de cette fenêtre est divisé en plusieurs parties:

- 0: Cinq lancers de l'user1
- 1: Cinq lancers de l'user2
- 2: Affichage du résultat final

On commence donc par la partie 0 mais au tout début, on a l'affichage de quelques éléments de transition qui a lieu (fond, texte et bouton). Cela indique que c'est à l'équipe blanche de jouer. Un clic sur le bouton *continueButton* est requis pour continuer. Ces éléments sont ainsi cachés et on retrouve l'interface de jeu.

Lorsque l'user1 aura effectué ses cinq tirs, on affichera à nouveau ces éléments de transitions avant de passer aux tirs de l'user2. Toutefois, quelques changements auront lieu : changement des couleurs du maillot et de la ligne de fond comme évoqué plus haut, affichage des données du premier tireur et du coach de la *franchise2.*

Après ces cinq nouveaux lancers, un clic sur ce bouton quittera simplement le jeu puisqu'il sera accompagné d'un affichage du résultat final.

changeBarValue

Pour faire varier la valeur des barres de tir, nous avons écrit une fonction qui permet de d'augmenter ou diminuer ces valeurs pour la barre de puissance ou pour celle de précision. En fonction de la valeur de l'attribut int *pop (*pour power or précision*),* on appliquera ce changement de valeur à la bonne barre. Ensuite, un autre attribut int intervient (*iod* pour increase or decrease), sa valeur permettra de déterminer si la valeur de la barre en question devra augmenter ou diminuer.

Si augmentation il y a, un test if sera effectuer afin de vérifier si la valeur de la barre a atteint la valeur maximum (100) car, si tel est le cas, la valeur de *iod* doit changer afin de faire diminuer la valeur lors des prochains appels de la fonction.

A contrario, si nous étions dans un cas de diminution, une vérification doit avoir lieu concernant la valeur de la barre puisque si celle-ci vaut désormais 0, *iod* doit voir sa valeur être changée pour repasser en phase d'augmentation.

changeDisplayImage

La méthode *changeDisplayImage* est très simple à comprendre, elle permet simplement de mettre à jour les informations du joueur. On récupérera ses données depuis la bonne *Franchise* grâce à la valeur de *part* et on parcourra la *liste<Joueur>* jusqu'au bon joueur à l'aide de l'int *tenClick* qui aura pour valeur la nombre de clic sur le bouton de tir sur la partie en question. On retrouvera donc *tenClick* plus tard.

on_shotButton_clicked

Il s'agit-là de la fonction la plus fournie en ligne de ce programme. Elle contrôle effectivement beaucoup de choses dans cette fenêtre et est évidemment plus qu'essentiel au bon fonctionnement de celle-ci.

Dans un premier temps, cette fonction stoppera le *QTimer* : ce dernier sera en action quand cette méthode sera appelée et a pour fonction première de stopper ce *timer* et de stocker la valeur de la barre. Ces valeurs seront stockées dans des *liste<int>.* Ces listes seront donc composées de 10 valeurs et seront traités deux par deux. Ainsi, un int nommé *valeur* est déclaré et prendre la valeur de la barre de puissance ou de la barre de précision selon la valeur de *pop.* Si la valeur de *pop* vaut 0, le bouton de tir s'est vu être cliqué afin de déterminer la puissance du joueur. Ainsi, les deux phases de tir ont eu lieu (clic pour puissance et clic pour précision).

Si tel est le cas, plusieurs choses sont à effectuer. On commence par déterminer quelle équipe a éffectué ce lancer avec l'attribut *part.* Ensuite, on détermine la réussite (ou non) du lancer à l'aide d'un calcul. Ce calcul est le suivant:

$$\text{impact}_{\text{coach}} \cdot \left(\text{power}_{\text{Player i}} \cdot \text{valeur}_{\text{powerBar}} + \text{precision}_{\text{Player i}} \cdot \text{valeur}_{\text{precisionBar}}\right)$$

Ce calcul prend donc en compte les deux notes de puissance et de précision du joueur en question, ainsi que les deux valeurs obtenues suite aux deux cliques et l'impact des coachs. Pour que le tir apporte un point à l'équipe, il faut que cette valeur soit supérieure à 1 et que chacun des valeurs des barres soit supérieurs à 40%. Si le bout de la barre cesse de changer de position sur un fond rouge, le tir sera raté.

Si le tir est réussi, on augmente le score de l'équipe d'un point et un switch a lieu afin de changer la couleur des voyants pour passer du noir au vert. Dans le cas contraire, si le shoot est raté, un switch nous permettra de passer du noir au rouge pour le voyant correspondant.

Après cela, un test if est effectué afin de vérifier s'il s'agissait du dernier tir de l'équipe en question. Si *tenClick* vaut 9, les dix clics (5x2) ont été effectués. Après chaque dernier tir de chaque équipe, le thread se met en repos pendant une seconde grâce à la fonction *sleep()* afin de laisser les users regarder son score. On réutilise *part* ici afin de savoir s'il s'agissait du dernier tir de la *franchise1* ou de la *franchise2*. S'il s'agissait du tir de l'équipe de l'user1, il suffit de réinitialiser la valeur de *tenClick* à 0 pour l'équipe de l'user2 et d'afficher les éléments de transition en indiquant, cette fois-ci, que c'est à l'équipe noir de jouer.

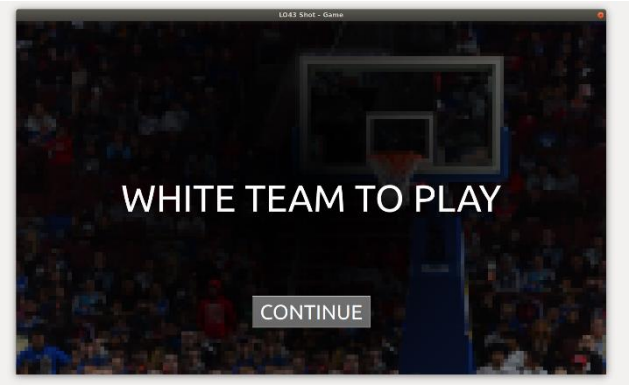
S'il s'agissait du tir de l'user2, on doit passer à l'affichage des résultats. Pour se faire, on change le texte de *continueButton* pour passer de « CONTINUE » à « EXIT ». On procède également aux changements de position du texte de transition et du tableau des scores. Le texte de transition indiquera le vainqueur s'il y en a un ou bien s'il y a eu égalité.

7. Test - Jeu

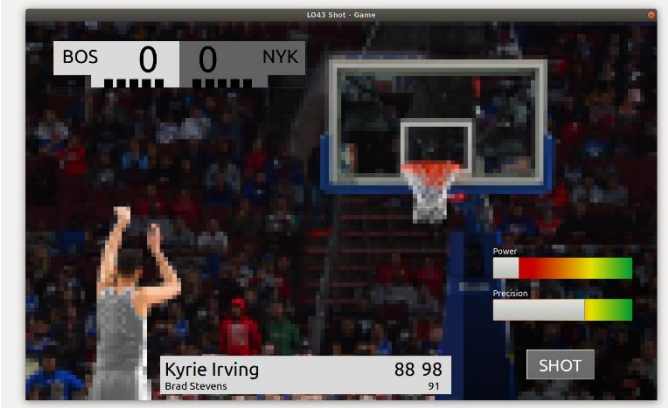
Lorsque la première fenêtre s'affiche, trois boutons s'affichent et laisse le choix à l'utilisateur de lancer le jeu, de découvrir les règles du jeu ou bien de le quitter. Si l'utilisateur souhaite lire les règles en cliquant sur le deuxième bouton, les trois premiers boutons se cachent afin d'afficher les règles accompagnées d'un bouton qui, lorsque l'on clique dessus, retournera à la première interface.

Après un clic sur le bouton EXIT, la fenêtre se ferme et après un clic sur le bouton NEW GAME, on passe à la fenêtre de choix des équipes. Cette dernière est initialisée sur deux franchises, les Golden States Warriors et les Toronto Raptors. En sélectionnant d'autres équipes, l'actualisation de l'affichage des données fonctionnent bien puisque les informations se mettent bien à jour.

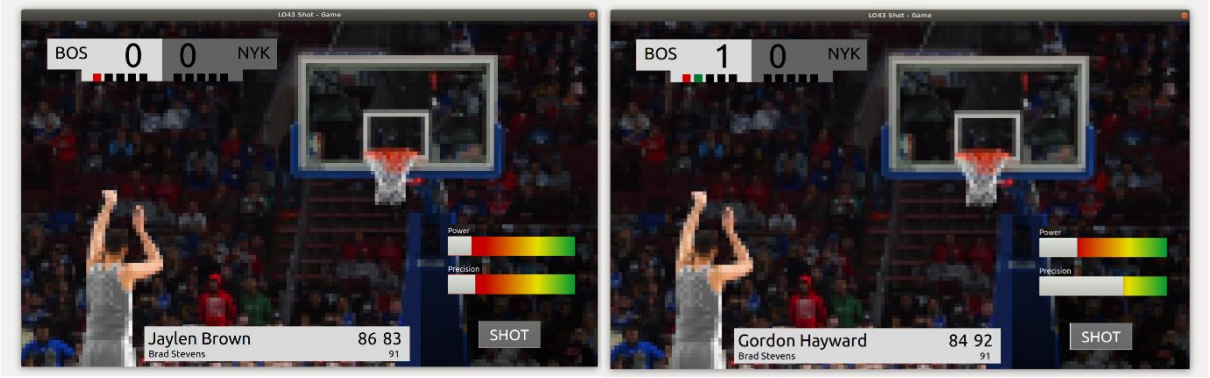
Une fois que les utilisateurs ont sélectionnés leurs équipes, ils peuvent passer à la fenêtre suivante en cliquant sur le bouton PLAY. Une interface de transition à lieu en indiquant que c'est à l'équipe blanche de jouer.



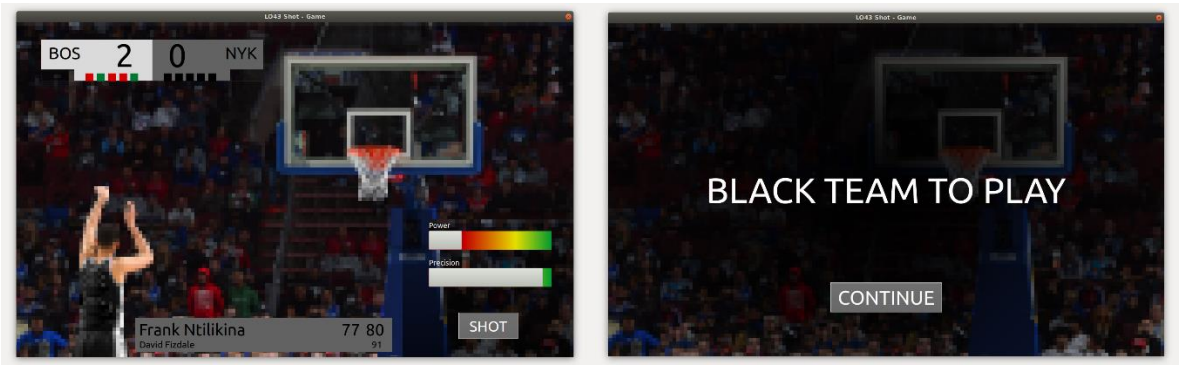
Il faut cliquer sur le bouton CONTINUE pour passer à l'interface de jeu. On remarque sur cette nouvelle interface que les différentes informations s'affichent correctement. Le tableau des scores est bien initialisé et les informations concernant le joueur et le coach de la première équipe sont les bonnes. Le mouvement de la première barre de tir est bon aussi.



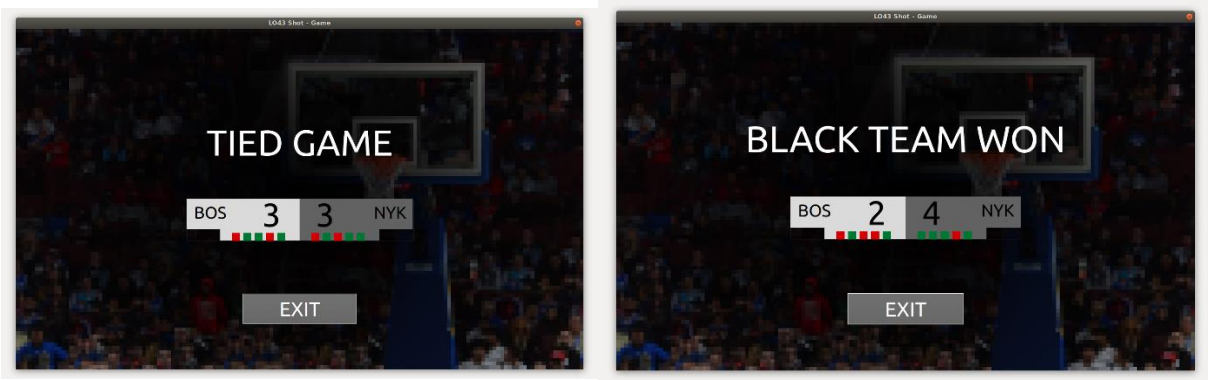
En cliquant sur le bouton shot, la barre de puissance cesse de bouger et c'est au tour de la barre de précision de voir sa valeur changée. En cliquant à nouveau sur ce même bouton, le joueur change, le score se met à jour si le tir a été réussi, le voyant change de couleur en fonction de cela aussi. Si le tir est raté, le voyant passe au rouge, sinon, il passe au vert.



Après que tous les tirs de la première équipe ont eu lieu, l'interface de transition refait apparition, indiquant cette fois-ci que c'est à l'équipe noire de jouer. Ensuite, on remarque que la couleur du maillot change, ainsi que la ligne se trouvant derrière les informations du joueur et du coach. L'affichage de ce dernier aussi a été mis à jour.



Après tous les tirs, l'interface de transition réapparaît. Le texte donnera le résultat final et changera de position. Cette fois-ci, le tableau des scores reste à l'écran en passant au centre. Le bouton n'est plus CONTINUE mais EXIT et un clic dessus fermera simplement.



8. Conclusion

Finalement, ce projet a été très enrichissant pour nous sur énormément de points différents. Dans un premier temps, il nous a permis de mettre en pratique ce que nous avons appris concernant la programmation orientée objet. Il nous aura également permis de partager nos connaissances mais aussi d'apprendre encore avant, pendant et après la partie codage. Nous avons pu mettre en œuvre et appliquer une véritable démarche de génie logiciel.

Toutefois, le jeu peut être encore améliorer sur plusieurs points. En effet, d'autres fonctions auraient pu être ajouter, d'autres modes de jeu, etc... Ce programme avait initialement pour but de mettre en pratique les connaissances acquises en LO43 et c'est chose faite puisqu'il regroupe plusieurs notions vues en cours autour de la POO.