Melaine Siako
Shagufta Khizar

# Projet C++ : Compte rendu
# Le concours de miss

13/01/2017

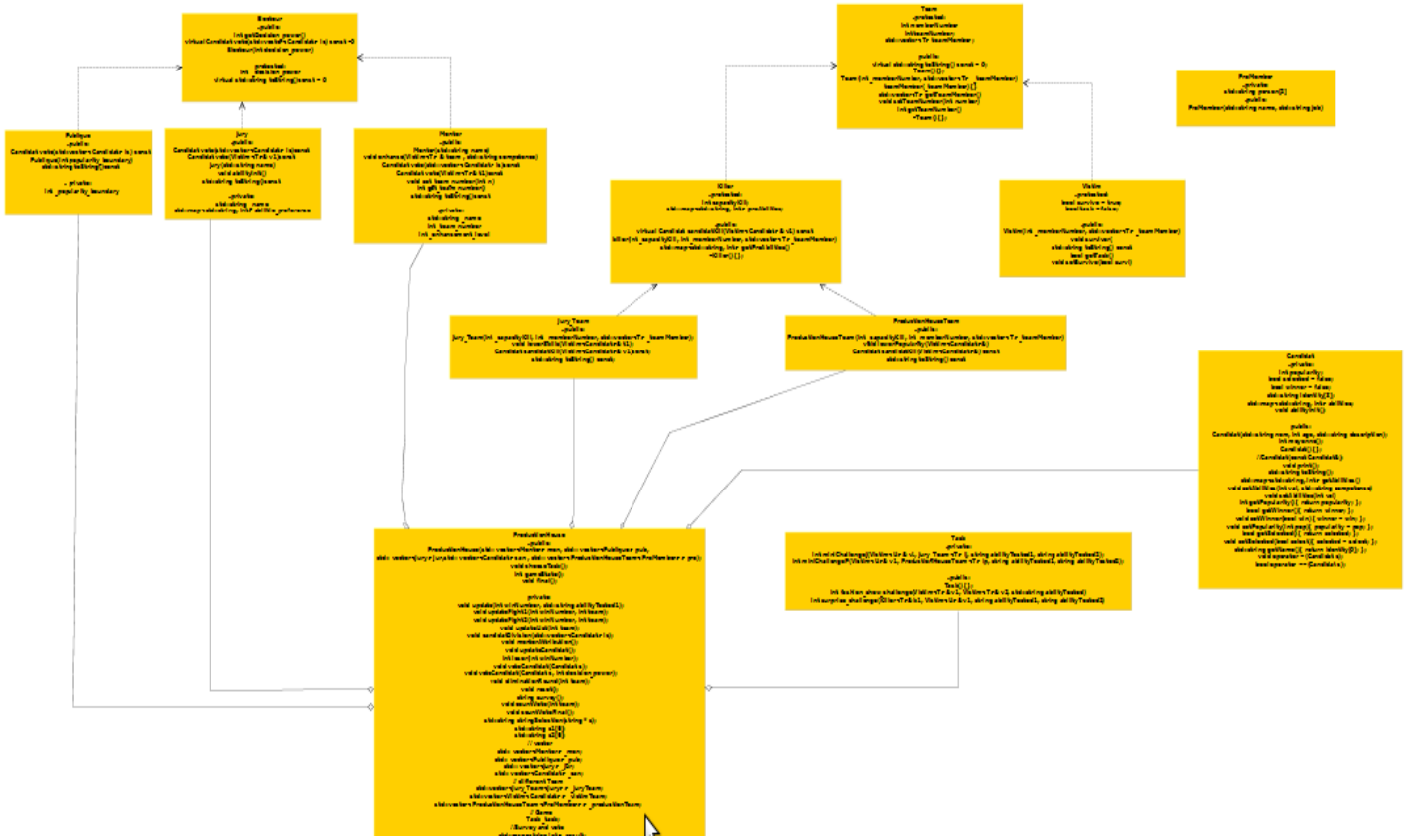
## Description du concours de miss :

Nous avons réalisé un concours de miss. Notre programme se compose d'une partie centrée sur le jeu, le « concours », et une autre partie sur les votes.
Voici le principe du jeu :
Tout commence avec des candidats. Ces derniers sont répartie en deux équipes constituées par la maison de production. Chaque équipe est chaperonnée par un mentor, une ancienne miss. Leur rôle étant de permettent aux candidats d'augmenter leurs compétences dans différents domaines.
La compétition s'effectue entre les deux équipes de candidats qui doivent s'affronter par rapport à leur compétences. Cependant nous avons aussi des « Killers », des équipes spéciales qui sont composées de jury ou de professionnels, possédant des compétences spéciales très élevées. Voilà pourquoi une équipe de candidat affrontant des Killers possède une probabilité assez faible de survie pour tout ses membres.
Ce jeu est fortement liée aux votes réalisés durant la partie :
Des jury ainsi que les mentors doivent voter pour éliminer un candidat d'une équipe perdante, ce vote est effectué après chaque combat. Lorsque nous atteignons la phase finale du jeu, nous réalisons un sondage auprès du publique. En fonction de celui-ci nous décidons du gagnant. La nouvelle miss élu n'est jamais celle que le publique souhaite.

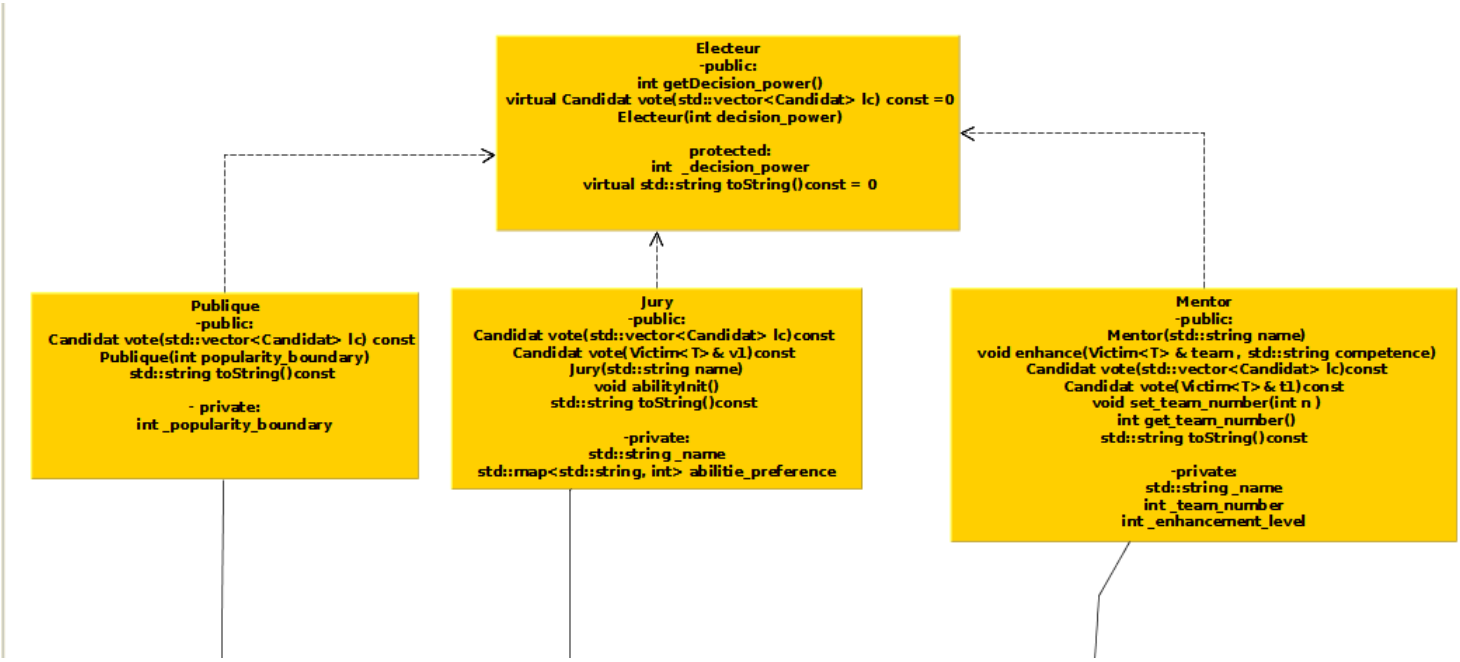
## Description du programme :

Comme je l'ai annoncé dans la section précédente, nous pouvons observé une dichotomie dans notre programme, même si les deux parties sont interdépendantes. Le lien entre les personnes qui votent et les équipes se fait via une classe qui est la maison de production.

Voici une vue d'ensemble du diagramme de classe qui sera décortiqué au fur et à mesure :



- Les personnes devant voter : Les électeurs




Les classes dérivant de la classe abstraite Electeur sont le suivantes : Jury, Mentor, Publique. Ces classes ont le pouvoir de voter pour un candidat d'une équipe. Selon l'avancement du jeu le vote n'est pas effectué de la même façon.

> *Avant la finale :*

Seuls les jury et mentors peuvent voter. Une équipe ayant perdu un combat est donné aux jurys et au mentors, afin que ces derniers éliminent un candidat.
Les jurys possèdent toujours une compétence à laquelle ils accordent le plus d'importance. En effet dans la vrai vie un jury est en choisi en fonction de sa réussite dans un domaine particulier. Lorsqu'il juge un candidat il le fait par rapport à son domaine d'expertise.

Un mentor est là pour aider un candidat à progresser au niveau de ses compétences. Ce qui l'intéresse avant tout est l'évolution de celui-ci. Logiquement le mentor élimine le candidat possédant en moyenne de compétences la plus faibles.

A ce stade le publique ne peut pas voter. Nous souhaitons faire passer en final les candidats les plus méritant.

> *La finale :*

Lorsque le nombre de candidat restant est de deux nous atteignons la finale. Ici tous les électeurs peuvent voter.

Une fois encore le jury choisira en fonction de ses compétences préférés, alors que le mentor choisit le candidat au hasard puisqu'ils sont tous compétents.

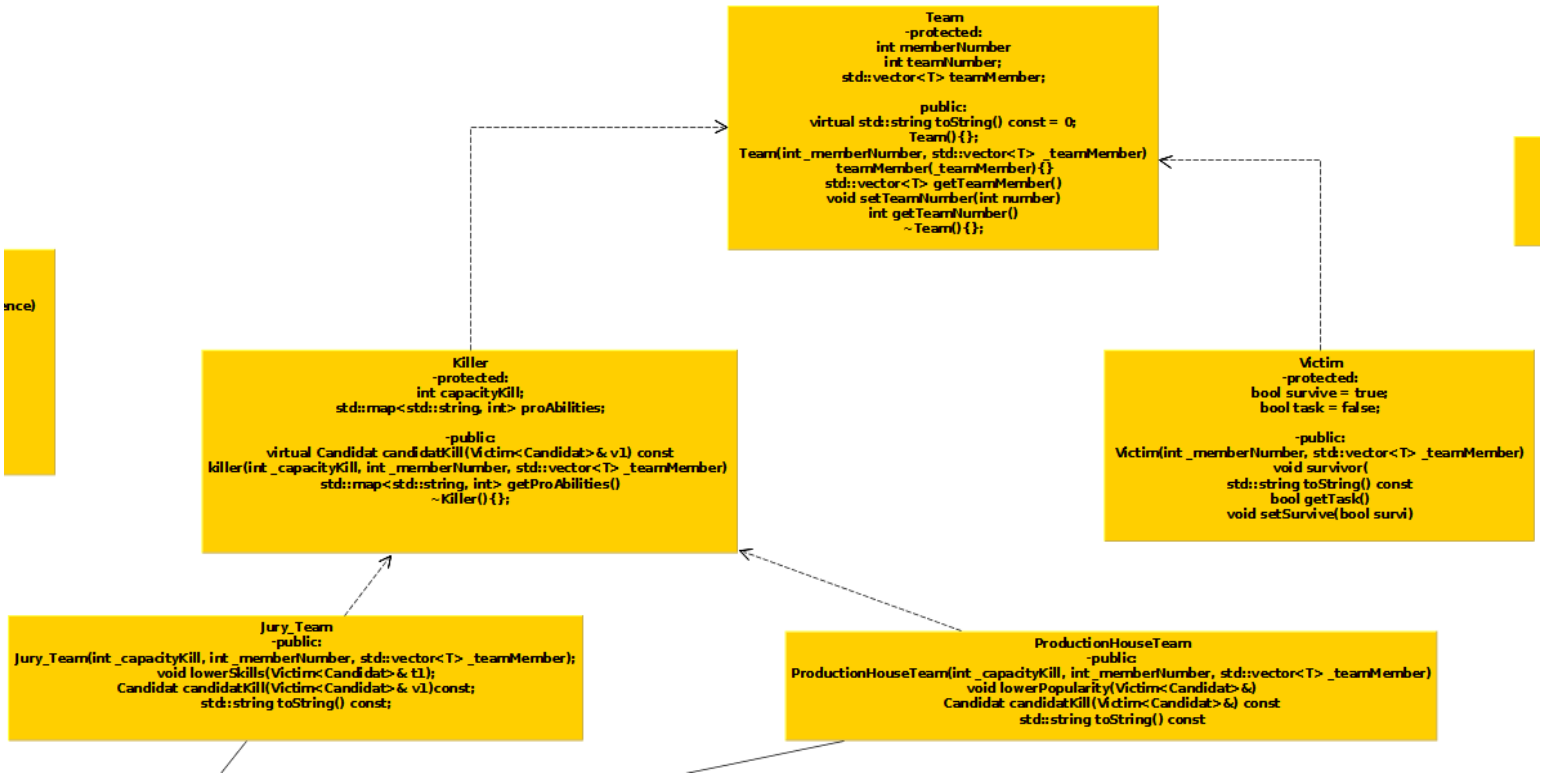
La maison de production réalise un sondage auprès du publique, il s'agit en réalité du vote de ce dernier. En effet il était  préférable de ne pas faire voter le publique dans le sondage et encore une fois en finale. Le résultat pouvait changer dans certains cas déterminés au hasard. Le publique vote pour des candidats en fonction de leur popularités. Si chaque candidat possède une popularité

suffisante alors l'élu est tiré au hasard. Dans le cas contraire le publique votera pour le plus populaire.
La différence entre cette phase et la précédente réside aussi dans le fait que les poids décisionnel entre les différents électeurs sont pondéré différemment.
Dans l'ordre croissant nous avons :

publique < mentor < jury.

Les deux méthodes virtuels de cette classe sont le vote et toString. Tous les électeurs doivent voter.

- Les personnes devant jouer : Les équipes




Une autre partie de notre programme nécessite de l'héritage. Toute équipe possède des membres ainsi qu'un numéro d'équipe. A cela s'ajoute la méthode virtuel toString qui permet d'afficher les caractéristiques de tous les membres de l'équipe. Ces dernières sont soient victimes, soient killer.

> *Les victimes :*

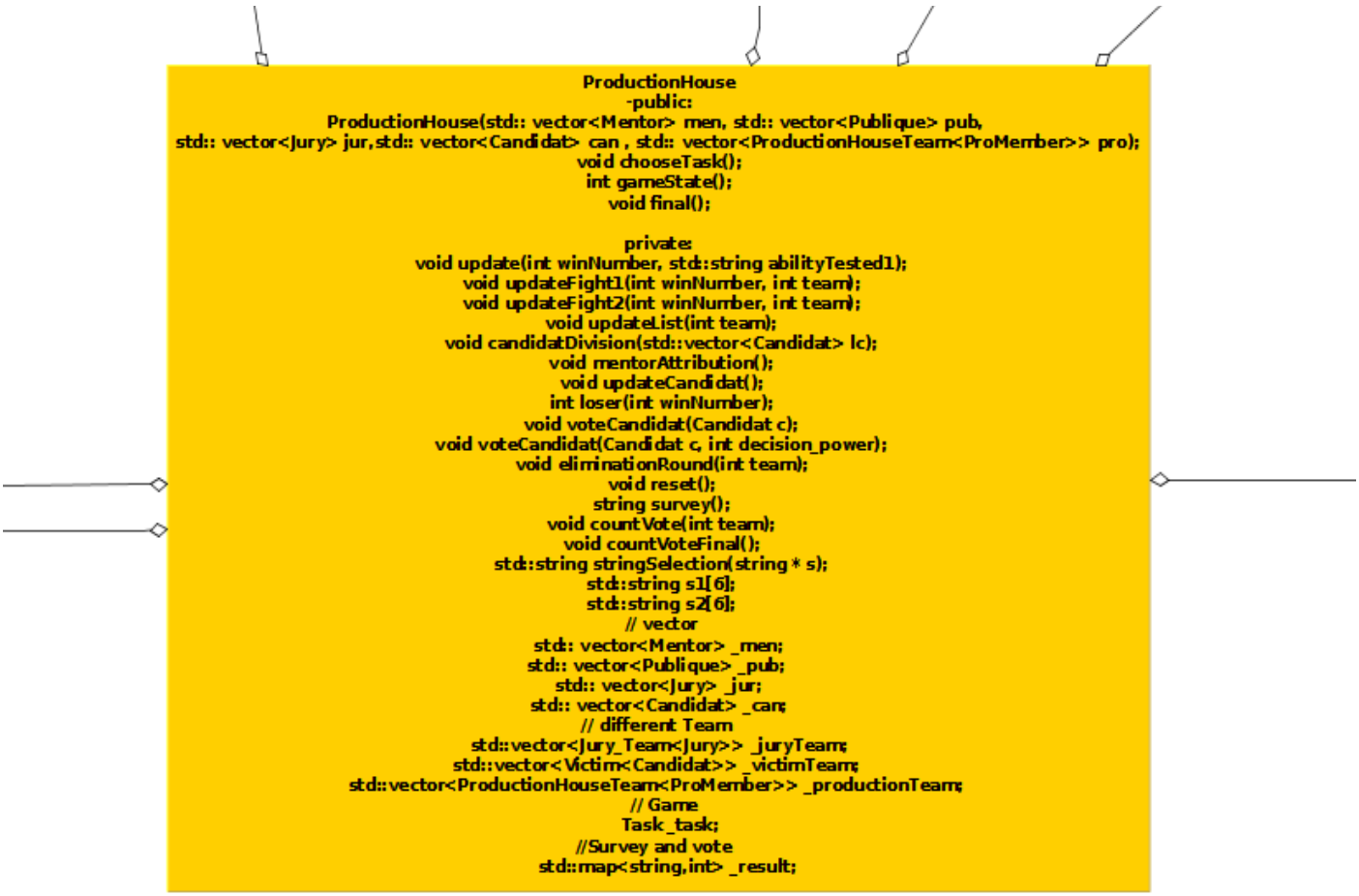

Les équipes de victimes sont formées de candidat classique. Chaque candidat possède une identité et des compétences qui lui sont propre, initialisées à la construction de celui-ci.
A chaque tour de la compétition un combat a lieu entre des équipes victimes. Une équipe perdra ce combat et une élimination s'en suivra. Seulement l'équipe peut, si le candidat en question est assez compétent, demander à rejouer une épreuve pour sauver ce membre. Cette action ne peut être faite qu'une fois. Parfois l'équipe n'a pas le choix de sauver un membre, il s'agit de combat contre killers.

> *Les Killers :*

Cette classe abstraite dérive également de la classe équipe de base, celle dont dérive les victimes. Elle se compose de deux sous-classes : les professionnels et l'équipe de jury. Chacune de ces équipes possède une capacité à « tuer » un candidat, mais aussi des compétences spéciales. Ces dernières permettent d'utiliser une autre méthode virtuelle : candidatKill. En effet un combat perdu face à une équipe de killers entraîne systématiquement l'élimination d'un candidat d'une équipe. Contrairement aux combats entre victimes, celui-ci ne donne pas accès à une augmentation de compétences via le mentor.
Au contraire l'équipe de jury peut également diminuer les compétences de chaque candidat. L'équipe de professionnels va diminuer la popularité seulement.

- Rassemblement de ces deux entités : La maison de production

Le rôle de la maison de production est de créer les équipes de victimes, organiser les combats, mettre à jour les vecteurs des équipes, mais surtout de permettre la réunion entre ceux qui votent et ceux qui jouent. C'est notamment la maison de production qui compte les votes, effectue les sondages et réalise un trucage de vote pour faire gagner une personne impopulaire.



## Difficultés rencontrées :

Dans les objectifs que nous avions fixées au départ le jeu ainsi que les votes devaient d'abord être fait via le terminal. Une fois que cette partie fonctionnait nous devions utiliser QT pour faire une partie graphique. En effet le concours de miss se prête bien au visuel.
Cependant nous avons pris plusieurs de mauvaise décisions durant ce projet.

L'une d'entre elles à été notre façon de programmer. Nous avons écrit nos méthodes chacune de notre côtés après s'être réparti les différentes classes. Seulement le seul élément auquel nous avons prêté attention pendant les vacances est la compilation de ces dernières. A la fin des vacances nous avions une partie de notre programme qui compilait. Seulement au retour des vacances nous avons finit d'écrire les méthodes manquantes, ce qui représentait l'autre partie de notre programme. Une fois que tout le programme a été écrit nous l'avons exécuté. Puisque nous n'avions pas tester nos méthodes au préalable, l'exécution a provoquée énormément de « segmentation fault ». Après avoir débugger nos méthodes générant des segmentation fault, nous avons remarqué que nous avions due modifier toutes les méthodes écrites pendant les vacances. Ceci nous a fait perdre beaucoup de temps, nous empêchant de réaliser la programmation graphique que nous avions imaginé. Ne pas être capable de réaliser un des objectifs pour avoir perdu du temps dans notre organisation a provoqué un grand sentiment de frustration. C'est avant tout une erreur de débutant en programmation.

S'ajoute à cette erreur d'organisation un programme relativement complexe à implémenter. En effet nous avons énormément de champs, de méthodes, de classes, permettant d'effectuer trop d'action différentes. Les votes ne sont pas uniformes et dépendent de combats variables.

Nous avons également rencontré plusieurs difficultés techniques liées à la programmation. Nos équipes sont réalisées grâce à de l'héritage. Pour ne pas avoir un vecteur de membre de jurys, de candidats, ou de professionnels nous avons fait appel à un template. Le champ membre d'une équipe est donc unique, ce qui implique que nous n'avons pas de répétition de différents type de vecteurs. Cela nous évite d'avoir uniquement un champ initialisé sur trois à chaque instanciation d'une équipe. Malheureusement l'utilisation des templates a aussi posée des problèmes. Nous avons appris qu'une variable static d'une classe template ne peut être utilisée comme nous l'avons fait dans les TP précédents. Puisque nous avons des équipes, logiquement nous avons pensé à attribuer des numéros d'équipe. Ce numéro d'équipe est une variable static. Par contre une variable static issu d'une classe template est réinitialisée pour chaque instanciation avec un nouveau type. Pour palier à ce problème nous avons fait appel à une méthode externe qui incrémente une variable static n'appartenant pas aux équipes, mais chaque équipe peut quand même avoir un numéro.
Nous avons opté pour des vecteurs de membres d'équipe. En effet le vecteur est ce qu'il y a de plus pratique lorsqu'il faut accéder à un membre précis dont on connaît la position dans le conteneur. Cependant chaque combat donne lieu à l'élimination d'un candidat d'une équipe. Ce qui implique que nous devons être capable de supprimer un candidat précis dans le vecteur. Le conteneur le plus pratique pour ce type d'action se trouve être une liste. Puisque l'utilisation de la méthode pop_back, qui n'élimine que le dernier élément d'un vecteur, n'était pas pratique ici nous avons trouver un autre moyen. En effet nous avons copié le vecteur contenant les membres de l'équipe dans une liste, supprimé le candidat en question, et recopié cette nouvelle liste dans un vecteur.

## Parties de l'implémentation dont nous sommes les plus fières :

Globalement, même si nous n'avons pas atteint nos objectifs nous sommes plutôt fière de l'ensemble du programme, plus particulièrement de la maison de production. En effet il s'agit de la classe la plus longue, et la plus complexe que nous avons eu à gérer. Le défi majeur résidait dans le fait que toutes les classes en dehors de celle-ci possèdent des instances dans cette dernière. Aussi il fallait faire attention de bien modifier les caractéristiques de nos équipes de maison de production même après avoir appelé une méthode qui réalisent des modifications locale. C'est aussi dans cette classe que nous avons beaucoup manipulé des map, des vecteurs et même des listes. Cette classe nous a générée le plus d'erreurs et donc nous a permit d'apprendre plus.