Langages Orientés Objet : C++ moderne
TP N° 3 – Les collaborations

Objectifs :
Ce TP vise à vous faire mettre en pratique la théorie présentée dans le support de cours *Collaborations* qui est disponible sur Moodle → S5A → Langages Orientés Objet. Si malheureusement ce TP a lieu avant le cours, votre encadrant sera là pour vous aider. La page Moodle dédiée à LOO fournit aussi des liens vers d'autre documentation. N'hésitez pas non plus à chercher par vous même. Deux séances de TP seront dédiée à ce sujet.
En particulier ce TP traitera :

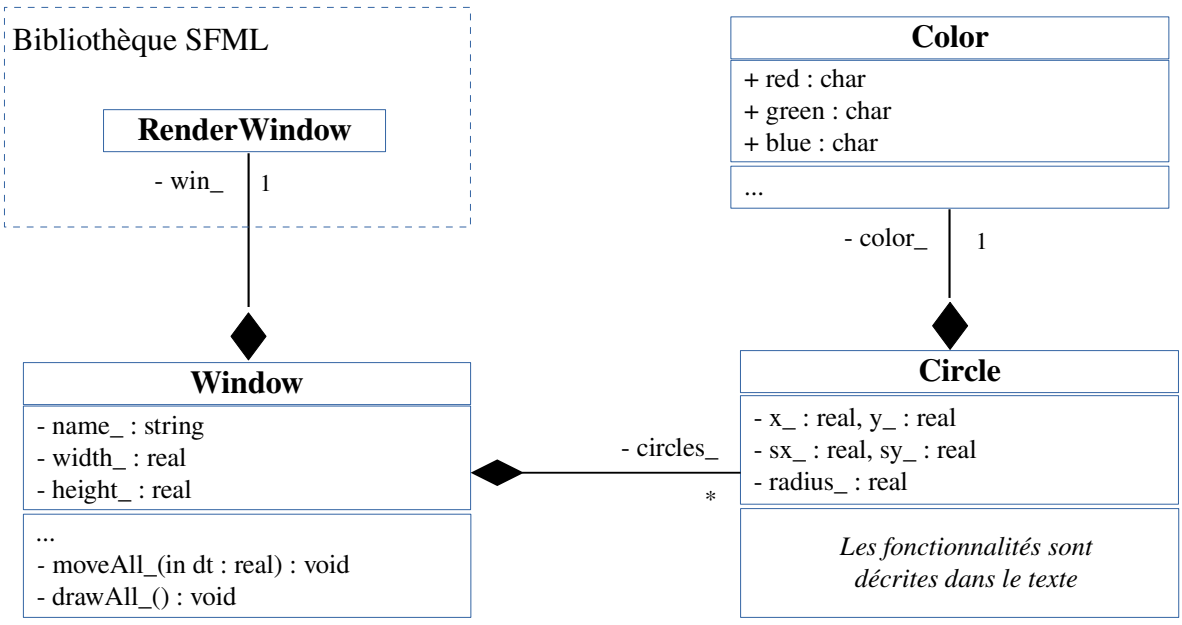
- Renvoie de valeurs multiples, utilisation des tuples.
- Collaborations :
  - composition.
- Utilisation d'une bibliothèque graphique.

Tous les concepts en C++ vus en S4 avant sont donnés pour acquis. Cela veut dire qu'il ne vous sera plus dit en détail ce qu'il faut faire. Par exemple, vous devez savoir quand un paramètre doit être passé par valeur ou par référence ou par référence constante, ou qu'une méthode doit être constante sans besoin qu'on le spécifie.

Contexte : Création d'une fenêtre contenant des objets graphiques
Dans le dossier de travail, nommé s5loo, que vous avez créé lors du premier TP, créez un sous-dossier dédié à ce TP, nommé *TP3_collaborations*. Placez dans ce dossier le code fourni sur Moodle dans la section dédiée au TP3.
Le code fourni contient :

- une classe *Window* qui se sert de la bibliothèque graphique **SFML** pour dessiner une fenêtre,
- une fonction globale *getTime()* dans le module window qui renvoie le nombre de secondes qui se sont écoulées depuis le 1 Janvier 1970,
- un fichier *prog.cpp* contenant le *main()*.

Le *GNUmakefile* fourni sur Moodle dans la section « IMPORTANT – fonctionnement des Labos » vous permet de compiler aussi le code pour ce TP (et pour les prochains).
Ouvrez un terminal dans ce dossier et lancez la compilation avec la commande *make*. Puis lancez l'application résultante pour constater que, dans l'état actuel, le code fourni ne fait rien d'autre qu'afficher une fenêtre vide capable de capturer certains événements du clavier et de la souris.
Il faudra partir de ce code et implémenter le diagramme de classes suivant. Attention, faute de place le diagramme n'est pas exhaustif.



1. Classe Circle.
Déclarez une classe *Circle*, comme indiqué dans le diagramme de classes (sans oublier la composition avec la structure Color dont le code est donné dans le fichier *color.hpp* et celle avec la classe Window). L'invariant de cette classe est la position. Elle doit être restreinte à la dimension de la fenêtre dans laquelle nous visualiserons les cercles : les cercles devront être toujours visibles. En outre la position variera selon la valeur de la vitesse. Le rayon et la couleur du cercle sont donnés une fois pour toutes à sa création et ils ne pourront plus être modifiés. Implémentez les méthodes suivantes :

- un constructeur qui prend les dimensions (largeur et hauteur) de la fenêtre d'appartenance, la position x, y du cercle, les composantes sx et sy de sa vitesse, son rayon et sa couleur, et qui initialise proprement les attributs de la classe (n'oubliez pas d'utiliser la liste d'initialisation des membres) et de faire en sorte que l'invariant de la classe soit respecté : la position d'un cercle doit être restreinte à la dimension de la fenêtre dans laquelle il sera dessiné.
- une méthode *position()* pour lire la position *x_*, *y_* du cercle. Utiliser un *tuple*.
- une méthode *radius()* pour lire l'attribut *radius_* du cercle,
- une méthode *color()* pour lire l'attribut *color_* ,
- les 5 fonctions spéciales (constructeur par recopie, constructeur par déplacement, affectation par recopie, affectation par déplacement, destructeur). Réfléchissez à la nécessité (ou pas) de recopier des cercles et choisissez si la permettre (ou l'interdire). Faut-il permettre le déplacement ? Faut-il un destructeur ?

Fournissez aussi la fonctionnalité suivante sous forme de fonction non membre (pour augmenter le niveau d'encapsulation de la classe) :

- *draw()* qui ne renvoie rien et dessine l'objet Circle reçu en paramètre dans la fenêtre de type *sf::RenderWindow* passée elle aussi en paramètre. Les instructions pour dessiner un cercle en utilisant la bibliothèque SFML sont données en annexe. Évidement le code fourni n'est qu'un exemple, vous devez l'adapter à votre problème, il est là juste pour vous éviter la tâche de chercher dans la documentation de la bibliothèque SFML comment dessiner les formes géométriques (ce qui sortirait des objectifs de ce TP). Attention pour utiliser le type *sf::RenderWindow* vous devez inclure le fichier *SFML/Graphics.hpp*.

Compilez régulièrement votre code pour vous assurer de ne pas faire trop d'erreurs et de les corriger à fur et à mesure.

Dans la fonction *main()* dans le fichier *prog.cpp* instanciez trois objets de type Circle en fournissant des valeurs aléatoires pour leur position (x,y), leur vitesse (sx,sy), leur rayon et leur couleur (rouge, vert, bleu).
Pour avoir des valeurs aléatoires, le code est déjà fourni dans le *main()*. Les distributions données respecterons les contraintes suivantes :

- *xposDistr* permet d'obtenir x ∈ [0.0, *win.width()*]
- *yposDistr* permet d'obtenir y ∈ [0.0, *win.height()*]
- *speedDistr* permet d'obtenir sx et sy ∈ [-50.0, 50.0]
- *dimDistr* permet d'obtenir le rayon ∈ [10.0, 60.0]
- *colorDistr* permet d'obtenir les composantes de la couleur ∈ [0, 255]

Pas la peine d'appeler la méthode *draw()* sur les cercles créés, la fonction qui s'occupe du dessin de la fenêtre se trouve dans la classe Window et nous aborderons le dessin à l'étape suivante. Compilez et lancez le code. Dans l'état actuel rien ne changera dans la fenêtre, donc si vous voulez vous assurer que les objets ont été bien instanciés, vous pouvez surcharger l'opérateur d'injection pour le type Circle et afficher la valeur des attributs de chaque objet instancié dans le *main()* à l'écran. La surcharge de l'opérateur d'injection est facultatif.

2. Classe Window.
Ouvrez les fichiers relatifs à la classe Window et modifiez son code pour implémenter la composition avec la classe Circle. Remarquez que la composition voulue est de type *à plusieurs composants*. Appelez ce vecteur *circles_*, comme indiqué dans le diagramme de classes.

Ensuite, fournissez les fonctions membres suivantes :

- *add()* qui permet d'ajouter un cercle au vecteur *circles_*, n'oubliez pas que la donnée passée en paramètre doit être stockée, donc utilisez la fonction *std::move()* pour éviter au maximum des recopies inutiles,
- *drawAll_()* qui dessine tous les cercles de la Window, elle se contente d'appeler la fonction *draw()* définie dans le module dédié au type Circle sur tous les cercles.

Les fonctions *add()* et *drawAll_()* doivent accéder directement au vecteur de cercles, pour cette raison nous les implémentons sous forme de fonctions membres et pas de fonctions non membres.

Modifiez la méthode *display()* de la classe Window. Cette fonction contient une boucle qui est exécutée tant que la fenêtre reste ouverte. Elle permet de rafraîchir continuellement la fenêtre et de capturer les événements. Le code pour capturer le clic sur la souris et sur le clavier est fourni à titre d'exemple même si nous ne l'utiliserons pas dans le cadre de ce TP, il nous sera utile pendant les prochains TPs.
Au début de cette boucle, après l'appel à la méthode *clear()* sur l'attribut *win_*, il faut dessiner tous les cercles de la Window, n'oubliez pas que vous venez d'implémenter une méthode dédiée à ce propos.
Interrogez-vous maintenant sur la visibilité que la fonction *drawAll_()* devrait avoir.

Compilez régulièrement votre code pour vous assurer de ne pas faire trop d'erreurs et de les corriger à fur et à mesure.

Dans le fichier *prog.cpp* modifiez le *main()* pour que les trois objets de type Circle que vous avez instanciés précédemment soient ajoutés à la fenêtre représentée par l'objet *win*.

3. Ça bouge !

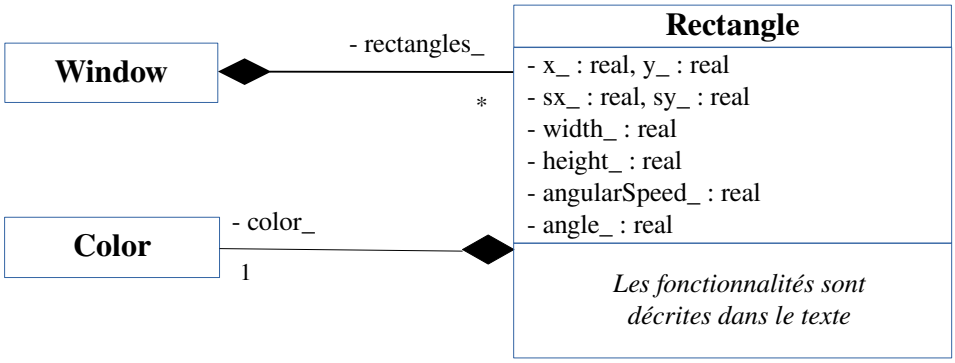
Les cercles colorés sont bien jolis, mais vite ennuyeux…
Pour rendre l'application un peu plus intéressante, commencez par doter les cercles de la capacité de se déplacer. Faites cela en ajoutant à la classe Circle une fonction membre *move().* Cette méthode ne renvoie rien et reçoit en paramètre les dimensions (largeur, hauteur) de la fenêtre d'appartenance et un double *dt* pour appliquer au cercle courant le déplacement causé par sa vitesse pendant l'intervalle de temps *dt*. Attention ! Le cercle ne doit pas sortir de la fenêtre et lorsqu'un bord est touché, la composante de la vitesse dans la même direction doit être annulée. Par exemple, si le déplacement du cercle fait que la coordonnée x de sa position devient négative, alors cette coordonnée doit être mise à zéro, ainsi que la composante sx de la vitesse.

La classe Window doit maintenant se servir de la fonctionnalité *move()* de la classe Circle pour animer les cercles qu'elle possède. Pour cela ajoutez une méthode *moveAll_()* qui appelle la fonction membre *move()* sur tous les cercles contenus dans la Window en lui passant les dimensions de la fenêtre et le temps écoulé *dt* qu'elle même reçoit en paramètre.
Similairement à la méthode *drawAll_(),* la fonction membre *moveAll_()* doit accéder directement au vecteur de cercles, pour cette raison nous l'implémentons sous forme de fonction membre et pas de fonction non membre.
Quelle est la visibilité de la fonction *moveAll_()* ?

Modifiez la méthode *display()* de la classe Window. Au début de la boucle l'intervalle de temps *dt* écoulé entre une exécution de la boucle et la précédente est déjà calculé. Servez vous de cette valeur pour déplacer les cercles sur la fenêtre selon leur vitesse et le temps écoulé, n'oubliez pas que vous venez d'implémenter une méthode dédiée à ce propos.

Si vous avez bien implémenté les compositions et les fonctions demandées vous verrez les objets de type Circle apparaître dans la fenêtre et ils bougeront selon leur vitesse.

4. Classe Rectangle.
Ajoutez une classe *Rectangle* selon le diagramme de classes suivant :



N'oubliez pas la composition avec la structure Color dont le code est donné dans le fichier *color.hpp* et celle avec la classe Window.
L'invariant de la classe est le même que celui de la classe Circle.
Implémentez les méthodes suivantes :

- un constructeur qui prend les dimensions (largeur et hauteur) de la fenêtre d'appartenance, la position x, y du rectangle, les composantes sx et sy de sa vitesse, sa largeur et sa hauteur, sa couleur et une vitesse angulaire (qui doit avoir comme valeur par défaut 0.0), et qui initialise proprement les attributs de la classe en vous assurant que l'invariant de la classe soit respecté : la position d'un rectangle doit être restreinte à la dimension de la fenêtre dans laquelle il sera dessiné. L'attribut *angle_* doit simplement être initialisé à 0.0, il ne faut pas passer une valeur au constructeur pour l'initialiser. Cet dernier attribut représente l'angle de rotation du rectangle.
- une méthode *position()* pour lire la position *x_*, *y_* du rectangle. Utilisez un *tuple*.
- une méthode *size()* pour lire les attributs *width_*, *height_* du rectangle. Utilisez un *tuple*.
- une méthode *color()* pour lire l'attribut *color_*,
- deux méthodes *angle()*, une pour lire et l'autre pour modifier l'attribut *angle_*, gardez cette valeur entre -180 et 180. Attention ! La méthode pour modifier l'attribut *angle_* n'a qu'une utilisation interne à la classe, donc elle doit être privée.
- *move()* qui ne renvoie rien et reçoit en paramètre les dimensions (largeur, hauteur) de la fenêtre d'appartenance et un double *dt* pour appliquer au rectangle courant le déplacement causé par sa vitesse pendant l'intervalle de temps *dt*. Elle doit aussi calculer l'angle de rotation (*angle_*) en lui ajoutant l'*angularSpeed_* multiplié par *dt* (rappelez-vous que vous venez d'implémenter une petite fonction privée qui permet de modifier correctement l'attribut *angle_*, utilisez la !). Attention, comme pour les cercles, les rectangles ne doivent pas sortir de la fenêtre !
- les 5 fonctions spéciales.

Fournissez aussi la fonctionnalité suivante sous forme de fonction non membre :

- *draw()* qui ne renvoie rien et dessine l'objet Rectangle reçu en paramètre dans la fenêtre de type *sf::RenderWindow* passée elle aussi en paramètre. Elle doit également appliquer la rotation de *angle_* degrés au rectangle. Les instructions pour dessiner un rectangle (et lui appliquer une rotation) en utilisant la bibliothèque SFML sont données en annexe. Évidement le code fourni n'est qu'un exemple, vous devez l'adapter à votre problème.

Implémentez la composition entre les classes Window et Rectangle montrée dans le diagramme de classes précédent. Suivez la même démarche suivie pour l'implémentation de la composition entre Window et Circle. Remarquez que pour ajouter un objet de type Rectangle à la Window, vous pouvez surcharger la fonction membre *add()*.

Compilez régulièrement votre code pour vous assurer de ne pas faire trop d'erreurs et de les corriger à fur et à mesure.

Dans la fonction *main()* dans le fichier *prog.cpp* ajoutez à la fenêtre, représentée par l'objet *win*, trois objets de type Rectangle en fournissant des valeurs aléatoires pour leur position (x,y), leur vitesse (sx,sy), leur dimensions (largeur, hauteur), leur couleur (rouge, vert, bleu) et la vitesse angulaire.
Comme vous avez déjà vu, le code pour avoir des valeurs aléatoires est déjà fourni dans le *main()*. Les distributions données respecterons les contraintes suivantes :

- *xposDistr* permet d'obtenir x ∈ [0.0, *win.width()*]
- *yposDistr* permet d'obtenir y ∈ [0.0, *win.height()*]
- *speedDistr* permet d'obtenir sx et sy ∈ [-50.0, 50.0]
- *dimDistr* permet d'obtenir la largeur et la hauteur ∈ [10.0, 60.0]
- *colorDistr* permet d'obtenir les composantes de la couleur ∈ [0, 255]
- *angularDistr* permet d'obtenir la vitesse angulaire ∈ [-30, 30]

Si vous avez bien implémenté les compositions et les fonctions demandées vous verrez des cercles et des rectangles apparaître dans la fenêtre et glisser selon leur vitesse. Les rectangles effectueront aussi une rotation selon leur vitesse de rotation.

Annexe

Exemple d'instructions pour dessiner un cercle vert de rayon 15.0 en position (20.0, 50.0) avec la bibliothèque SFML (code à adapter à votre problème). L'origine de l'objet est placée dans le centre du cercle. L'objet *window* doit être de type *sf::RenderWindow* :

```
double r = 15.0;
sf::CircleShape s{(float)r};
s.setFillColor(sf::Color{0, 255, 0});
s.setOrigin(float(r), float(r));
s.setPosition(20.0f,50.0f);
window.draw(s);
```

Exemple d'instructions pour dessiner un rectangle rouge de dimension 10.0x20.0 et en position (1.0, 1.0) et lui appliquer une rotation de 15.0 degrés avec la bibliothèque SFML (code à adapter à votre problème). L'origine de l'objet est placée dans le centre du rectangle. L'objet *window* doit être de type *sf::RenderWindow* :

```
double w = 10.0;
double h = 20.0;
sf::RectangleShape s{sf::Vector2f{(float)w, (float)h}};
s.setFillColor(sf::Color{255, 0, 0});
s.setOrigin(float(w*0.5), float(h*0.5));
s.setPosition(1.0f,1.0f);
s.setRotation(15.0f);
window.draw(s);
```

Attention : la bibliothèque SFML utilise principalement des *float*.