TP 7 — C++ Avancé

1. Modèle de classe

Modèle de classe, std::set.

1. Écrire un modèle de classe `ensemble<T>` représentant un ensemble de valeurs de type `T`. Ce modèle de classe est bien plus simple que ce que nous avons écrit dans le TP3 – Exercice 6 : ici il n'y aura pas de classe abstraite accompagnée de sous-classes utilisant des structures de données différentes ; Il y aura une seule implantation de la structure de données, par exemple à l'aide d'un `std::vector`. Ce modèle de classe sera muni des méthodes suivantes :
   constructeur par défaut (construisant un ensemble vide) ;
   constructeur par recopie (si nécessaire, en fonction de la structure de données choisie pour représenter les éléments de l'ensemble, mais si vous avez la possibilité d'appliquer la règle des 0, faites-le) ;
   destructeur (idem) ;
   `empty` retournant un booléen valant `true` si l'ensemble est vide et `false` sinon ;
   `find` prenant comme paramètre un T et retournant un booléen valant `true` si l'ensemble contient ce T et `false` sinon ;
   `insert` prenant comme paramètre un T et l'ajoutant à l'ensemble s'il n'y figure pas, et levant une exception (par exemple `std::invalid_argument`) s'il y figure déjà.
2. Écrire une méthode `unionwith` prenant comme paramètre un ensemble et rajoutant à l'ensemble courant tous les éléments de l'ensemble passé. À titre d'exercice, cette méthode devra appeler la méthode `insert` pour ajouter les éléments, on interdit un appel direct à `std::vector::push_back`. Attention, l'utilisateur de la méthode `unionwith` ne devra pas voir d'exceptions !
3. Écrire un `main` permettant de tester le modèle de classe `ensemble` pour représenter des ensembles de `int` (par exemple) et tester notamment la méthode unionwith.
4. *(question optionnelle)* Écrire un programme de test effectuant un grand nombre (plusieurs dizaines de milliers, à vous d'adapter en fonction de la puissance de la machine, mais il faut que votre programme prenne plusieurs secondes pour s'exécuter) d'ajouts et de recherches dans un `ensemble<int>` et écrire un programme faisant la même chose en utilisant `std::set<int>` http://en.cppreference.com/w/cpp/container/set. Comparer les différences de performance en utilisant la commande `time nomexecutable` dans un shell.

2. Pac-man

Ce sujet est une version modifiée de l'examen de seconde session de 2022-23.

On veut développer le moteur d'un jeu ressemblant à PacMan. *Dans ce jeu, sur le plateau, il y a différents éléments : le pacman (dirigé par le joueur), des fantômes (qui se déplacent aléatoirement), des murs (qui empêchent pacman et fantômes de passer) et des pacgommes qui sont mangées par le pacman quand il y passe dessus.*

1. Écrire un type énuméré `direction` pouvant prendre les 5 valeurs suivantes : `stop`, `droite`, `gauche`, `haut`, `bas`.
2. Les différents éléments qui composent le jeu ont une position, qui est formée d'un couple d'entiers non signés représentant des coordonnées dans le plan. Écrire une classe `position` pour représenter ces positions, et munir cette classe des méthodes suivantes :
   Constructeur à deux arguments entiers, pas de constructeur par défaut ;
   Deux accesseurs aux coordonnées nommés `x()` et `y()` ;

Une méthode `tostring` retournant une chaîne de caractères de la forme « *(12,18)* » ;
La possibilité de comparer deux `position` par l'opérateur d'égalité `==` et de différence `!=` ;
Un opérateur de sortie sur flux permettant d'écrire, soit `p` une `position`, `std::cout << p ;`

3. Écrire une classe `taille` mémorisant une largeur et une hauteur. Cette classe aura un constructeur à deux arguments entiers non signés et deux accesseurs appelés `w()` et `h()` permettant d'accéder respectivement à la largeur et à la hauteur. On ne définira pas de mutateurs, et il pourra être intéressant de définir un opérateur de sortie sur flux (notamment dans un objectif de mise au point du code).
4. Définir des classes pour représenter les éléments du jeu avec les conditions suivantes :
   Chaque élément a une position[1] et une taille.
   La taille du pacman est fixe, (13,13), comme celle des fantômes, (20,20), et des pacgommes, (3,3). Chaque mur peut avoir une taille différente, elle est fixée à la construction. La largeur et la hauteur d'un mur doivent être toutes les deux supérieures ou égales à 10.
   Définir des méthodes `pos` et `tai` permettant d'accéder à la position et à la taille, ainsi que `setpos` permettant de modifier la position.
   Écrire une méthode `typeobjet` retournant un caractère correspondant au type de l'élément (P pour pacman, G pour pacgomme, F pour fantôme, M pour mur).
   Certains éléments peuvent se déplacer : Le pacman et les fantômes. On mémorisera donc pour ces éléments une `direction`. Le déplacement initial est passé en argument au constructeur, et on définira les méthodes `deplacement` retournant la direction du déplacement et `setdir` pour modifier cette direction.
   Le pacman a un niveau d'invincibilité représenté sous la forme d'un entier. À la construction, ce niveau est 0. Vous définirez les méthodes `invincible` retournant un booléen valant vrai si et seulement si le niveau d'invincibilité est > 0 ; `decrementerinvincible` enlevant 1 au niveau d'invincibilité si celui-ci est > 0 ; et `devenirinvincible` rajoutant 200 au niveau d'invincibilité.
5. Écrire un opérateur de sortie sur flux permettant d'écrire, soit `e` un élément, `std::cout << e;`. L'affichage sera composé du type de l'élément, de la position et de la taille, par exemple : `M(310,10)-10,180`. Dans le cas du pacman, le niveau d'invincibilité sera aussi affiché.
6. Écrire une méthode `contient` permettant de tester si un élément A en contient un autre B[2].
   Écrire une méthode `intersection` permettant de tester si deux éléments sont en collision[3].
7. Définir une classe `jeu` représentant l'état du jeu. Dans cette classe vous déclarerez **un seul** attribut pour représenter l'ensemble des éléments du jeu. Un `jeu` contient aussi un attribut représentant l'état de la partie : une valeur d'un type énuméré `etat` pouvant prendre les valeurs `encours`, `defaite`, `victoire`.
8. Faire en sorte qu'une instance de jeu soit copiable. La copie devra être totalement indépendante de l'original.
9. Écrire une méthode `afficher` prenant comme argument un flux de sortie et envoyant sur ce flux l'ensemble des éléments du jeu ainsi que l'état.
10. Écrire une classe `exceptionjeu` sous-classe de `std::exception` mémorisant une chaîne de caractères décrivant la nature de l'exception. Cette chaîne sera initialisée à la construction et retournée par `what()`.
11. Écrire dans `jeu` une méthode `ajouter` permettant d'ajouter un élément au jeu. Si l'élément à ajouter entre en collision avec un élément existant, l'ajout ne sera pas fait et une `exceptionjeu` sera levée. Contrainte supplémentaire à titre d'exercice : pour faire la recherche de collision, plutôt que d'utiliser une boucle, vous utiliserez un algorithme de la bibliothèque standard.

1 Les coordonnées du point en haut à gauche du rectangle définissant l'élément. On supposera que le point d'origine du repère (0,0) est situé en haut à gauche de l'écran.

2 En d'autres termes, la méthode détermine si le rectangle de B est inclus dans le rectangle de A.

3 En d'autres termes, la méthode détermine si les rectangles des deux éléments ont une intersection non vide.

12. Écrire une méthode `ajouterfantomes` prenant comme argument un entier `e` et ajoutant `e` fantômes au jeu à des positions aléatoires dont la valeur x est comprise entre 0 et 320 et y entre 0 et 200[1] ainsi qu'une méthode `ajouterpacgommes` faisant la même chose avec un nombre de pacgommes passé en argument.
13. Écrire une méthode `accespacman` permettant d'accéder au pacman du jeu ; et une méthode `directionjoueur` prenant comme argument une direction d et modifiant la direction du pacman en appelant la méthode `setdir` sur le pacman[2]. Pour ces deux méthodes, pour rechercher le pacman, plutôt que d'écrire une boucle, vous utiliserez un algorithme de la bibliothèque standard.
14. Écrire une méthode privée `appliquerdeplacementcolllisionmur` effectuant le déplacement[3] de tous les éléments actifs du jeu. Le déplacement s'effectue de « une unité » dans la direction choisie, et s'il y a une collision avec un mur, le déplacement n'est pas fait[4].
15. Écrire une méthode privée `appliquerdeplacementcontact` vérifiant que le pacman n'entre pas en collision avec un fantôme. S'il est en collision avec un fantôme, l'état de la partie passe à `defaite`, sauf si le pacman est invincible, dans ce cas-là, le fantôme disparaît.
16. Écrire une méthode privée `appliquerdeplacementmanger` vérifiant si le rectangle correspondant au pacman ne contient pas une pacgomme. S'il en contient une, la pacgomme en question disparaît, et le pacman gagne de l'invincibilité (appel à `devenirinvincible`).
17. Écrire une méthode `changerdirectionfantomes` modifiant aléatoirement la direction des fantômes.
18. Écrire une méthode `tourdejeu` appliquant les déplacements ci-dessus, et faisant passer l'état de la partie à `victoire` s'il ne reste plus aucune pacgomme. Pour tester s'il ne reste plus de pacgomme, plutôt que d'écrire une boucle, vous utiliserez un algorithme de la bibliothèque standard.
19. Vous trouverez sur Moodle un fichier contenant un `main`[5] qui utilise les classes et méthodes demandées ci-dessus pour gérer une « partie » de pacman. Bien évidemment, ce `main` s'attend à ce que vous fournissiez des classes et méthodes ayant exactement les noms décrits ci-dessus. Vous avez donc le droit de modifier ce `main` pour l'adapter à ce que vous avez fait notamment si les signatures des méthodes ne sont pas tout à fait les mêmes. Dans tous les cas, il vous faudra sans doute légèrement modifier le `main` pour construire les obstacles (lignes 32 et suivantes) en fonction des choix que vous avez faits.

1 Attention, le tirage aléatoire peut produire des coordonnées dans lesquelles il y a déjà un élément. Il conviendra donc de faire attention lors de l'appel à ajouter pour faire en sorte que cette méthode ajouterfantomes fasse correctement l'ajout de e fantômes.

2 Attention, votre méthode devra prendre en compte « proprement » le cas où il n'y a pas de pacman parmi les éléments du jeu.

3 Dans la direction voulue, c'est-à-dire celle retournée par l'appel à deplacement() sur l'élément.

4 La détection de collisions se fait uniquement avec les murs. Il n'y a pas de détection de collisions avec le pacman, les fantômes ou les pacgommes.

5 Ainsi qu'un fichier `CMakeLists.txt` pour permettre la construction avec SFML (à utiliser sur vos portables, donc, comme l'exercice Space Invaders, car cette bibliothèque n'est pas installée sur les All-In-One). Il y a aussi un fichier `spites.png` à copier dans le répertoire contenant l'exécutable (*build*).