Campus Quest

Runtrack C++ : Jour 3

Contexte

Vous êtes un développeur passionné par les jeux de combat et vous décidez de vous lancer dans cette aventure en utilisant le langage merveilleux qu'est C++. Comme vous êtes aujourd'hui presque un pro du code, il est inimaginable de développer ce projet sans utiliser la programmation orientée objet (POO).

L'objectif de ce projet est de créer **Campus Quest**, un jeu interactif qui mettra à l'épreuve vos compétences de développeur et de joueur. Vous allez créer diverses classes pour représenter des vecteurs, des objets de jeu, des personnages, des armes et des éléments de décor. Le jeu doit inclure des fonctionnalités telles que des mises à jour et des dessins d'objets de jeu, des interactions entre les personnages, la gestion des collisions, et une interface utilisateur textuelle pour rendre le jeu interactif et vivant.

Job 01 - Vector2d

Dans le monde merveilleux de Campus Quest, chaque personnage et objet de jeu possède un emplacement. Afin de gérer ce positionnement en deux dimensions, créer une classe **Vector2d** qui possède des coordonnées (**double x**, **y**) des **accesseurs** et **mutateurs**, et de la **surcharge** pour les opérateurs + et - permettant l'addition et soustraction de vecteurs.

Il doit aussi posséder une méthode **distance()**, qui prend en paramètre une référence à un autre Vector2d, et qui renvoie la distance euclidienne entre les deux.

Job 02 - GameObject

Tous les objets interactifs, qu'il s'agisse de personnages, d'objets de décor,..., doivent posséder des fonctionnalités de base pour être correctement affichés et mis à jour dans le jeu.

Créer une classe abstraite **GameObject** qui hérite de vector2d, qui aura des méthodes virtuelles "**draw()**", et "**update()**".

Job 03 - Decor, Character

Dans votre jeu, on peut trouver des éléments de jeu variés, comme des décors et des personnages. Chaque élément a des caractéristiques et capacités différentes.

Créer des classes **Decor** et **Character**.

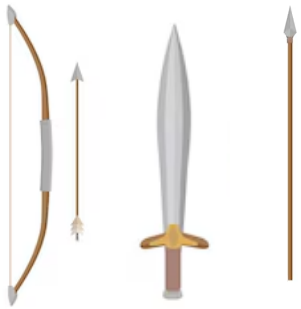
Decor doit implémenter "**update()**", qui ne lui permet pas de bouger.

Character doit avoir un nom, des points de vie, et une méthode "**isAlive()**".

Les deux doivent aussi implémenter la méthode "**draw()**".

Job 04 - Weapon

Dans Campus Quest, les personnages utilisent diverses armes. Créer une interface **Weapon** pour définir les fonctionnalités de base des armes et implémenter des classes spécifiques représentant des types d'armes. Ajouter la méthode "**attack()**" (qui prend en paramètre une référence à un Character).

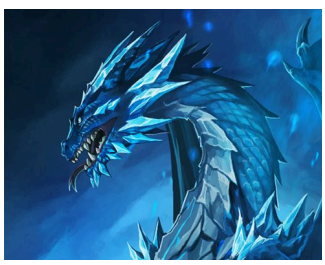


Créer les classes **Bow**, **Spear**, et **Sword**.

Bow doit avoir une portée de 4 et une puissance de 1, Spear 2/2, Sword 1/4.

Job 05 - Enemy, Player

Créer les classes **Enemy** et **Player** qui héritent de Character, et implémentent **update()**.



Les ennemis doivent s'approcher progressivement du joueur, ou lui infliger des dégâts s'il est à portée de main (portée de 1).

Le joueur doit posséder un **Weapon**, et une méthode **switchWeapon()**.

À chaque tour, le joueur doit attaquer avant de changer d'arme en suivant le cycle **Bow->Spear->Sword->Bow**...

Job 06 - Gestion

Créer un container pour **GameObject**. Il devrait utiliser des **pointeurs intelligents** (unique_ptr), ainsi que des fonctions pour ajouter des objets au jeu qui seront correctement placés dans le container.

Job 07 - Loop

Créer un programme principal qui instancie les éléments de jeu et gère le déroulement du jeu à l'aide d'une boucle principale. Cette boucle appliquera les méthodes **update()** et **draw()** à tous les objets de jeu, et nettoiera les objets qui ne sont plus nécessaires, comme les ennemis morts. Le jeu devra également afficher les actions dans la sortie standard et se terminer lorsque les conditions de victoire ou de défaite sont remplies.

Job 08 - Interactive

Rendez le jeu interactif en y ajoutant la possibilité d'utiliser des entrées utilisateur. À chaque tour, le joueur pourra se déplacer, attaquer ou changer d'arme..

Job 09 - Collisions

Ajouter la logique permettant de gérer les collisions. Les **GameObject** ne doivent plus pouvoir se traverser mutuellement.

Job 10 - Projectiles

Créer une classe **Projectile** héritant de **GameObject**, qui représente les flèches de l'arc.

Le projectile doit infliger les dégâts de l'attaque à la collision.

Job 11 - User Interface

Créer une interface textuelle pour donner un peu plus de vie au jeu.

Rendu

La runtrack doit être disponible sur votre github, au nom “**runtrack_cpp**”.

Les fichiers doivent être organisés précisément, comme indiqué dans les énoncés, dans un dossier correspondant à leur jour respectif.

Exemple : jour01/job01/hello_world.cpp

Base de connaissances

- ➔ interface vs classe abstraite
- ➔ fonctions statiques
- ➔ surcharge d'opérateurs