Programmation Orientée Objets L3 Miage, Info – M1 IMDS

TP 1 : jeu de tir au canon

Boulet de canon

Vous avez déjà à disposition une classe `point` (fichiers `point.h` et `point.cpp`) représentant un point dans le plan et une classe `vector` (fichiers `vector.h` et `vector.cpp`) représentant un vecteur dans le plan, classes écrites dans l'espace de noms `geom`.

Vous écrirez toutes les classes et fonctions concernant le jeu de tir au canon dans l'espace de noms `cannongame`.

Un boulet de canon est déterminé par sa position et son vecteur vitesse. **Écrivez** la classe `cannonball` (dans les fichiers `cannonball.h` et `cannonball.cpp`) représentant un boulet de canon. On veut pouvoir : construire un boulet de canon par défaut (position à l'origine et vitesse nulle), à partir d'une position (vitesse nulle) et à partir d'une position et d'une vitesse ; connaître la position et la vitesse d'un boulet (méthodes `position` et `velocity`) ; afficher un boulet sur un flot et lire un boulet depuis un flot (méthodes `print` et `read` puis surcharges des opérateurs de lecture et d'écriture) sous le format `[(x,y),(`v_x`,`v_y`)]`.

Testez : écrivez la procédure de test `testBouletConstructeur` (fichier `testcannon.cpp`) avec un test qui construit un boulet avec une position et une vitesse et affiche cette position et cette vitesse et une procédure de test `testBouletES` qui lit un boulet puis l'affiche. **Compilez** et **exécutez**.

Un boulet de canon se déplace pendant un intervalle de temps *dt* soumis à la pesanteur et à sa vitesse. D'une manière générale, un corps à la position *p(t)* avec une vitesse *v(t)* et soumis à une accélération *a(t)* voit sa position et sa vitesse changer suivant les règles : $v(t+dt) = v(t) + a(t)\times dt$ et $p(t+dt) = p(t) + v(t+dt)\times dt$.

Ici l'accélération est constante, c'est l'accélération de la pesanteur $(0,-g)$ où g=9,81 m/s^2. **Rajoutez** dans la classe `cannonball` la constante `g` et la constante `accel` représentant le vecteur accélération de la pesanteur. **Rajoutez** la méthode `move(dt)` qui déplace le boulet de canon pendant un temps `dt`, c'est-à-dire met à jour sa vitesse et sa position.

Testez : écrivez la procédure de test `testFall` qui demande une hauteur de chute *h* et un pas de simulation *dt*, met un boulet à cette hauteur avec une vitesse nulle, laisse tomber ce boulet au sol et affiche le temps mis pour tomber ainsi que le temps exact de chute $\sqrt{2h/g}$.

Canon

On veut maintenant ajouter un canon pour tirer le boulet de canon. Un canon est défini par sa position, son inclinaison (c.-à-d. son angle de visée, entier en degrés) et la vitesse de sortie du boulet tiré (vitesse fixée au début). **Écrivez** la classe `cannon` (dans les fichiers `cannon.h` et `cannon.cpp`) représentant un canon. On veut pouvoir : construire un canon à partir de sa position, de la vitesse de sortie des boulets et des limites minimales et maximales d'inclinaison (par défaut 10 et 60, l'inclinaison du canon vaudra initialement son inclinaison minimale) ; construire un canon à partir de la vitesse de sortie des boulets et des limites minimales et maximales d'inclinaison (par défaut 10 et 60, l'inclinaison du canon vaudra initialement son inclinaison minimale et sa position sera l'origine) ; connaître la position (`position`), l'inclinaison (`inclinaison`), la vitesse de sortie du boulet (`speed`) le vecteur vitesse correspondant (`speedVector`) ; modifier la position (`changePosition`) et l'inclinaison du canon (méthodes `raise(n)` et `lower(n)` qui respectivement lève et baisse le canon de $n\times5°$ sans dépasser les limites) ; enfin tirer et récupérer le boulet qui sort du canon (`shoot`). **Remarque** : il faudra convertir dans les calculs les degrés en radians en les multipliant par $\pi/180$ (valeur à mettre dans une constante statique, π=`M_PI`).

Testez : ajoutez la procédure de test `testCannonShoot` qui demande les données d'un canon, demande le pas de simulation *dt*, demande combien de fois lever le canon, tire le boulet et le fait avancer jusqu'à ce qu'il retombe au sol et affiche la distance à laquelle le boulet est tombé ainsi que la distance théorique $d = (v^2/g)\times\sin2\theta$ (v vitesse et θ inclinaison en radians).

Affichage graphique

Pour mieux visualiser la simulation du boulet on veut faire un affichage graphique. Rajoutez les fichiers `graphics.h` et `winbgi.cpp` de l'archive à votre projet et rajoutez `-lgdi32` dans les options de l'édition de liens de votre projet.

Récupérez la classe `viewer` (dans les fichiers `viewer.h` et `viewer.cpp`) représentant un afficheur graphique utilisant `winbgi`. On le construit avec la taille de la fenêtre et de la bordure, on peut l'ouvrir (`open`), le fermer (`close`), effacer la fenêtre (`clear`), temporiser (`wait`) et attendre que l'on clique à la souris (`repeatUntilButton`, à appeler à la fin d'une série de dessins). Il a aussi des méthodes internes pour calculer les coordonnées pixels dans la fenêtre correspondant aux coordonnées réelles (`pixelx(x)` et `pixely(y)`) et faire des dessins de base en donnant des coordonnées réelles (tracer un segment `drawLine(p1,p2)`, tracer un cercle `drawCircle(c,r)`, tracer un rectangle `drawRectangle(pbg,phd)`). **Ajoutez** la méthode `drawGround` qui affiche le

terrain (ligne horizontale en y=0) et la méthode `drawCannonball` qui affiche un boulet `b` (en traçant un cercle de rayon 5.).

Testez : copiez la procédure de test `testFall` en la procédure de test `testFallGr` et modifiez la pour en plus visualiser graphiquement la chute du boulet.

Ajoutez dans la classe `viewer` la méthode `drawCannon` qui affiche un canon `c` en traçant un rectangle centré sur la position du canon et en traçant un segment à partir de la position du canon correspondant au vecteur vitesse de sortie.

Testez : copiez la procédure de test `testCannonShoot` en la procédure de test `testCannonShootGr` et modifiez la pour en plus visualiser graphiquement le tir du boulet.