L'objectif de ce projet est d'implémenter un mini moteur de simulation physique, qui permet de gérer le déplacement d'objets régis par un ensemble de forces, la collision entre objets, et la répercussion des forces suite à une collision. En outre, vous devrez implémenter une interface graphique qui permet d'interagir avec les objets à l'aide de la souris. Pour simplifier le problème, vous vous limitez à un espace à deux dimensions.

Pour le dire plus simplement, on vous demande de programmer une version simplifiée du jeu Angry Birds.

# 1 Moteur physique

Un moteur physique est une bibliothèque logicielle dédiée à la résolution de problèmes de la mécanique classique, plus particulièrement les problèmes liés aux mouvements et aux interactions d'objets soumis à des forces. L'application grand public la plus répandue des moteurs physique est dans les jeux vidéo. Des moteurs physiques plus précis sont utilisés également dans l'industrie, dans la recherche et le développement, par exemple dans des simulateurs de vol ou de pilotage, des simulateurs pédagogiques médicaux, en génie civil pour prédire le comportement mécanique des structures telles des ponts, des ailes d'avion, etc. Dans ce projet, nous allons utiliser des méthodes de résolution simples basées sur la simulation numérique, qui sont généralement tout à fait suffisantes pour obtenir des animations réalistes.

# 2 Livrables

L'implémentation du moteur physique se fera de manière incrémentale. Pour chacune des étapes, vous aurez des cours de modélisation mathématique pour aborder les fondements physiques et comprendre leur modélisation mathématique. Ensuite, vous aurez des séances de TP encadrées par un enseignant en informatique pour implémenter ces méthodes.

Voici les trois versions qui vous seront demandées à rendre à des dates précises. Les dates et les formats pour rendre le travail seront indiquées sur Moodle.

**Livrable 1 : Animation vol d'oiseau et collisions** Cette première version se concentre sur l'affichage, en ignorant pour l'instant la simulation physique et l'interaction avec l'utilisateur. Vous devez produire un programme qui, lorsqu'il est lancé, affiche une fenêtre et joue une animation graphique. Les objets qui composent la scène sont, voir aussi la Figure 1 :

- un "oiseau avec bec", représenté par un disque et un petit triangle pour le bec ;
- cinq à dix obstacles circulaires fixes, disposés dans la partie droite de la scène à différentes hauteurs.

Lorsque le programme est lancé, l'oiseau doit effectuer 10 vols définis par des courbes paramétrées (voir les exemples donnés en TD). La vitesse de vol doit être réaliste, ni trop rapide, ni trop lente. Le vol s'arrête ou bien parce que l'oiseau a atteint un bord de la scène, ou bien parce qu'il est rentré en collision avec un des obstacles, ou bien au bout de 15 secondes si aucun des deux autres critères n'est satisfait. La position initiale de l'oiseau est proche du coin inférieur gauche de la scène. Après chaque vol, l'oiseau est remis en sa position initiale, et le prochain vol commence 1 seconde plus tard. En cas de collision, l'oiseau et l'obstacle en collision changent de couleur et l'animation s'arrête pendant 2 secondes, avant de recommencer le prochain vol. Les dix vols ont des trajectoires différentes, paramétrées de manière aléatoire pour qu'on ait une chance de voir aussi bien des vols avec collision que des vols sans collision. La trajectoire de chaque vol doit être matérialisée par des pointillés, effacés après la fin du vol. Finalement, à tout moment de l'animation, le bec de l'oiseau doit se trouver dans la direction de la tangente à sa trajectoire en le point de sa position courante.

**Livrable 2 : Animation basée sur la physique, interaction.** Dans cette deuxième version, vous devez faire une animation similaire, mais avec quelques différences notables, voir aussi la Figure 2 :

- la trajectoire de l'oiseau est calculée par une simulation physique,
- le lancement de l'oiseau se fait suite à une interaction avec l'utilisateur,
- les obstacles peuvent être circulaires et rectangulaires,
- les obstacles sont mobiles et effectuent des aller-retours,
- la scène a un sol.

Plus précisément, au lancement du programme, l'oiseau se trouve immobile au sol à proximité du coin inférieur gauche de la scène. Pour le lancer, l'utilisateur doit effectuer un *drag and drop* avec la souris, qui détermine le vecteur de la force de lancement. La valeur maximale de la force est bornée. La trajectoire du vol est définie par la force de lancement et la force de la gravité. Comme précédemment, le vol s'arrête après une collision ou lorsqu'un bord est atteint, la trajectoire du vol est matérialisée par des pointillés, et la vitesse de vol est réaliste. Chacun des obstacles

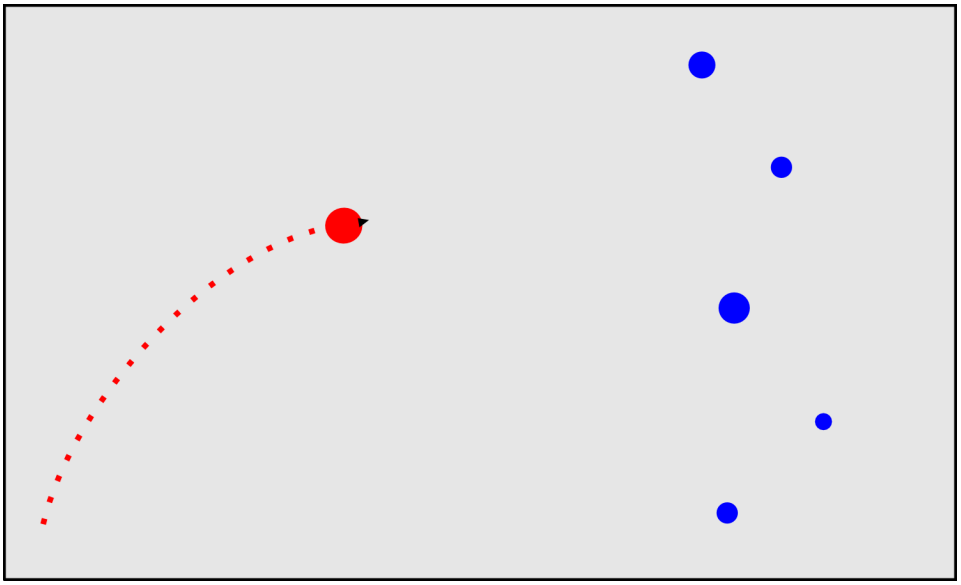

FIGURE 1 – Exemple de scène pour le Livrable 1. L'oiseau est en train d'effectuer une trajectoire parabolique de gauche à droite.

subit une force initiale de courte durée, puis à intervalles régulier une force de même intensité que la force initiale mais de direction opposée est appliquée à l'obstacle, ce qui aurait comme effet de lui faire faire des aller-retours. Les obstacles ne subissent pas la force de la gravité.

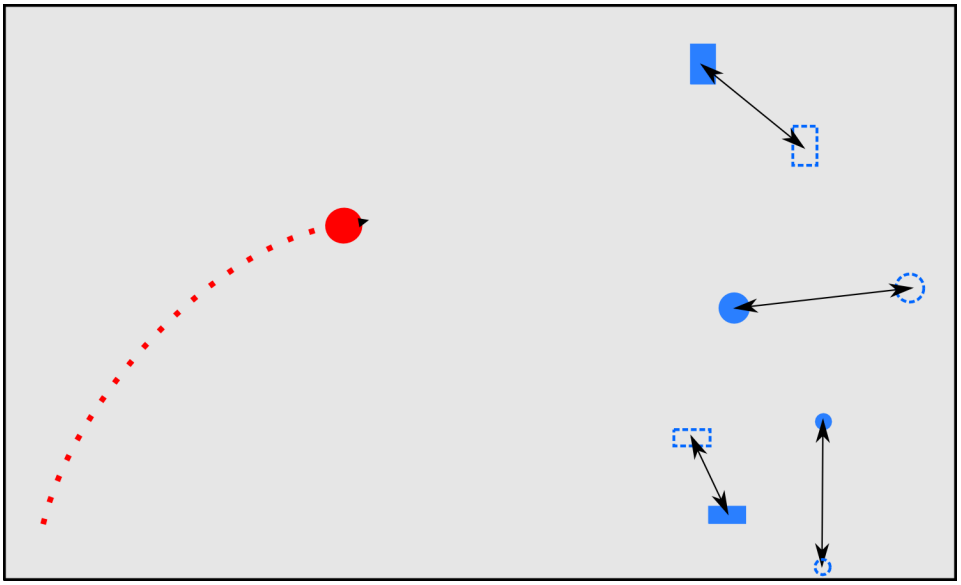

FIGURE 2 – Exemple de scène pour le Livrable 2. L'oiseau est en train d'effectuer une trajectoire parabolique de gauche à droite. Les obstacles effectuent des aller-retours le long des flèches.

**Livrable 3 : Moteur physique.** Dans cette dernière version, vous devez ajouter autant de fonctionnalités que vous voulez pour compléter l'ébauche de moteur physique faite pour le Livrable 2, et pour améliorer l'interaction. Les fonctionnalités minimales demandées sont les suivantes :

- la scène est semblable à la scène du Livrable 2, mais avec des obstacles immobiles comme pour le Livrable 1 ;
- lors d'une collision, l'objet en mouvement transmet son énergie à l'obstacle, qui se met en mouvement à son tour. La masse des deux objets est prise en compte ;
- on peut gérer une suite de collisions, si un obstacle entre en collision avec un autre obstacle ;
- l'oiseau et les obstacles sont dessinés par des images superposées à une forme géométrique.

Voici également quelques idées de fonctionnalités supplémentaires ou de variantes possibles :

**rotations** suite à une collision, les objets (rectangulaires) se mettent en mouvement, mais aussi en rotation si le point du choc est excentré par rapport au centre de gravité de l'objet ;

**chocs élastiques** un choc est élastique si, suite à une collision, une partie de l'énergie de l'objet en mouvement est transmise, et une autre partie est dissipée. Ainsi, en fonction des masses des deux objets et du coefficients

d'élasticité, suite à la collision l'objet qui était initialement en mouvement peut s'arrêter (et tomber s'il est régi par la force de la gravité), continuer son mouvement dans la même direction, ou bien changer de direction. En particulier, le sol peut être considéré comme un objet avec une très grande masse, qui suite à une collision ferait rebondir les objets au lieu de les arrêter net ;

**forces de frottement** prendre en compte la force de frottement de l'air ;

**obstacles cassants** chaque obstacle a un seuil d'énergie qu'il peut encaisser suite à un choc. Au delà de ce seuil l'objet se casse, en deçà il se met en mouvement ;

**constructions d'obstacles** les obstacles forment initialement une construction. Pour simplifier, au départ ils ne subissent aucune force (donc la construction est stable), et suite à une collision ils commencent à subir la force de la gravité, en plus de la force résultant du choc ;

**habillage différent de Angry Birds** normalement votre moteur physique est suffisamment générique pour qu'on puisse en faire n'importe quelle simulation dans le plan. Il suffirait pour cela de modifier les icônes affichées pour chaque objet et d'adapter les interactions possibles avec les objets ;

**...** toute autre fonctionnalité qui vous semble intéressante à avoir dans un moteur physique

# 3 Étapes pour le Livrable 1

Pour produire le programme demandé pour le Livrable 1, vous pouvez suivre ces étapes intermédiaires :

1. Dans une fenêtre, tracer une courbe paramétrée (tracé par une ligne continue).
2. Tracer un échantillonage de la courbe paramétrée (ligne pointillée).
3. Faire une "animation" pour le dessin de la courbe : la courbe se dessine au fur et à mesure en quelques secondes. La classe `java.util.Timer` pourrait vous être utile.
4. Utiliser un "oiseau avec bec" pour "dessiner" la courbe ; l'oiseau laisse une trace pointillée. Veiller à ce que le bec soit dans la direction de la tangente à la courbe.
5. Dessiner une scène d'objets. Chaque objet est une entité à part et a comme caractéristiques sa taille (diamètr du disque).
6. Détecter les collisions entre objets.
7. Implémenter l'animation demandée.

Si nécessaire, des indications vous seront données également pour les Livrable 2 et 3 au cours du semestre.

# 4 Critères de notation

Le projet contribuera à votre note dans plusieurs matières :

**Modélisation** La note prendra en compte la qualité du moteur physique, la vraisemblance et la correction de la simulation du mouvement et des collisions.

**Conception orientée objet** La note prendra en compte les aspects suivants de votre travail :
- le respect du MVC ;
- la qualité de l'UML rendu ;
- la mise en oeuvre de deux design patterns de votre choix ;
- la documentation fournie (Javadoc) ;
- la mise en oeuvre des tests (boite blanche et/ou boite noire).

**Projet tuteuré** La note prendra en compte aussi bien la qualité du logiciel final que le respect d'une bonne méthode de travail. Il est **indispensable que votre projet se trouve dans le dépôt Gitlab de l'IUT**, accessible par tous les membres du groupe de projet et par l'enseignant-e en informatique, ceci **dès la première séance de TP encadrée**. Cela permettra à l'enseignant-e de s'assurer du bon déroulement du projet. Plus précisément, voici les critères pris en compte dans la note :
- **la qualité des logiciels livrés**, qui devront implémenter les fonctionnalités demandées, être exempts d'erreurs, facile à lancer et à utiliser grâce à une courte documentation utilisateur. C'est le critère principal ;
- **le respect des délais** pour rendre les livrables ;
- **l'avancement régulier** : l'enseignant-e qui encadre les TPs doit pouvoir constater une réelle progression d'une semaine à l'autre. Le volume horaire à consacrer au projet tuteuré correspond à 5 heures de travail personnel par étudiant et par semaine. Pour un groupe de 4 étudiants, cela fait 20 heures de travail par semaine, qui doivent permettre l'ajout effectif de nouvelles fonctionnalités d'une semaine à l'autre. En outre, lors de chaque séance de TP encadrée, vous devez faire une courte démonstration du logiciel. Si la dernière version ne tourne pas, vous pourrez utiliser une ancienne version pour la démonstration ;

— **la répartition équitable du travail** dans le groupe de projet : chacun des membres du groupe de projet doit contribuer au logiciel final. Cette contribution pourra être vérifiée grâce aux commits sur le dépôt git. Il n'est pas obligatoire que tous les membres du groupe produisent le même nombre de lignes de code. Certains peuvent consacrer du temps à des tâches telles la documentation et la recherche d'information sur le Web, la définition de scènes et autres cas de test, la conception UML, etc. Cependant, le résultat de ce travail doit être **visible**, par exemple en ajoutant au dépôt git un répertoire de documents annexes (liens Web, photos de schémas, comptes rendus de discussions). En outre, vous êtes des étudiants et étudiantes en informatique formées pour le métier de développeur d'applications. De ce fait, chacun doit contribuer avec une quantité raisonnable de lignes de code, quelles que soient ces autres contributions au sein du groupe de projet. En cas de pair programming, indiquer explicitement dans le message du commit sur git les noms des deux étudiants ainsi : "pair programming Nom1 Nom2".