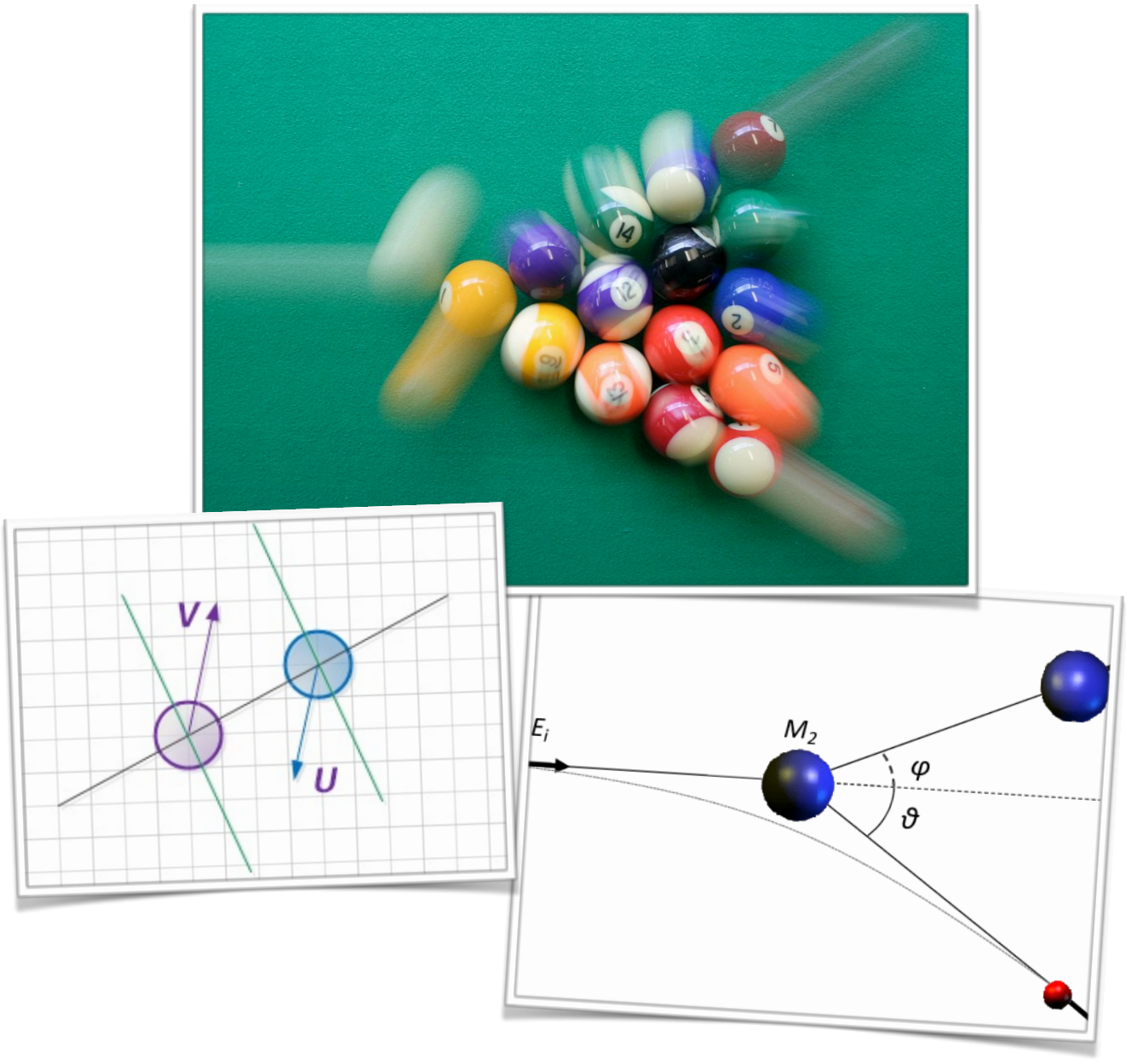
Rapport final

Mini-projet réalisé par :

CHEN Qiaoqian, HUA Zihao,
NASSREDDINE Zakaria & WANG Yuxuan



Billard JAVA • La programmation orientée objet et la physique au service du divertissement

Idée générale et objectif

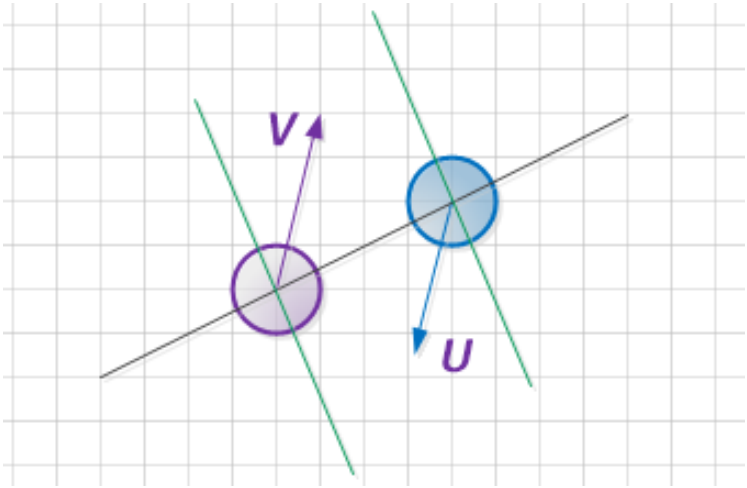
Notre projet a porté sur la conception et le développement d'un jeu de billard de table réalisé entièrement sous java en tirant profit de la programmation orientée objet. Le programme est doté d'une interface graphique permettant à l'utilisateur de se lancer dans une partie de billard en interagissant directement avec l'interface graphique qui lui est mise à disposition.

Les mouvements des billes, et on entend par cela les rebondissements résultants de collisions entre billes, entre billes et parois de la table, et tout ce qui en découle au niveau des vitesses et des accélérations sont régis par les fois fondamentales de mécanique Newtonienne.

Un billard, c'est cool. Mais la science derrière ?

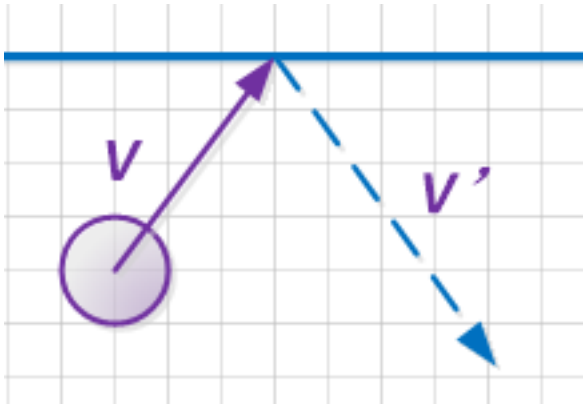
Nous avons commencé par la mise en équation du principe de conservation de la quantité du mouvement au niveau des collisions entre les différentes billes entre elles ainsi que les rebondissements au niveau des parois de la table. Ceci nous a permis de bien définir vitesses, accélérations et trajectoires des billes post-collision.

Principe général de fonctionnement :



Avant le choc, une bille a une vitesse et une accélération qui caractérisent son mouvement, en arrivant sur une paroi de la table ou on se cognant sur une autre bille, l'impact de ce contact se traduit par un changement de vitesse à la fois en norme, en sens et en direction. Pour un choc entre deux billes, la conservation de la quantité de mouvement impose, pour deux billes de masses respectives m1, m2 et de vitesses respectives **v**, **u**, l'égalité suivante : m1*__v__+m2*__u__=m1*__v'__+m2*__u'__ où __v'__ et __u'__ sont les vitesses respectives des deux billes après choc (les grandeurs écrites en gras représentent des quantités vectorielles). L'angle que fait la vitesse par rapport à chacun des axes du repère dont on a muni le plan est noté par

l'attribut arg. Après collision, les billes repartent avec des vitesses de sens opposés.



Un choc contre une paroi est basé sur le même principe, avec une variation de la vitesse qui agit cette fois-ci sur une seule bille.

Se reporter aux classes boule.java et table.java pour les équations de rebondissement et de collision.

Et l'utilisateur dans tout ça ?

L'interface graphique a été conçue pour simuler le jeu de la meilleure façon possible, dans les limites de nos capacités d'étudiants débutants en informatique. On s'est servi de Swing pour se rapprocher de l'expérience visuelle d'un vrai billard. L'utilisateur pilotera la partie principalement à l'aide de la souris dont le mouvement déterminera l'orientation de la tige (représentée virtuellement par une ligne en traits interrompus). Il peut également choisir la vitesse initiale qu'il veut communiquer à la bille à travers la durée de son maintien du clic sur la souris.

L'architecture du programme en bref

Notre programme fait appel à quatre classes : vecteur, boule, table et billard (la classe principale contenant le Main).

L'object **vecteur** a été intégré pour des fins de modélisation, il contient comme attributs les coordonnées cartésiennes d'un vecteur dans le plan, son argument et sa norme. On s'en sert pour coder les formules mathématiques mettant en jeu des quantités vectorielles : accélérations, vitesses et positions. Tout au long du programme, il est instancié de différentes manières puisqu'il a possède différents constructeurs, chacun adapté à un cas de figure relatif à un besoin particulier.

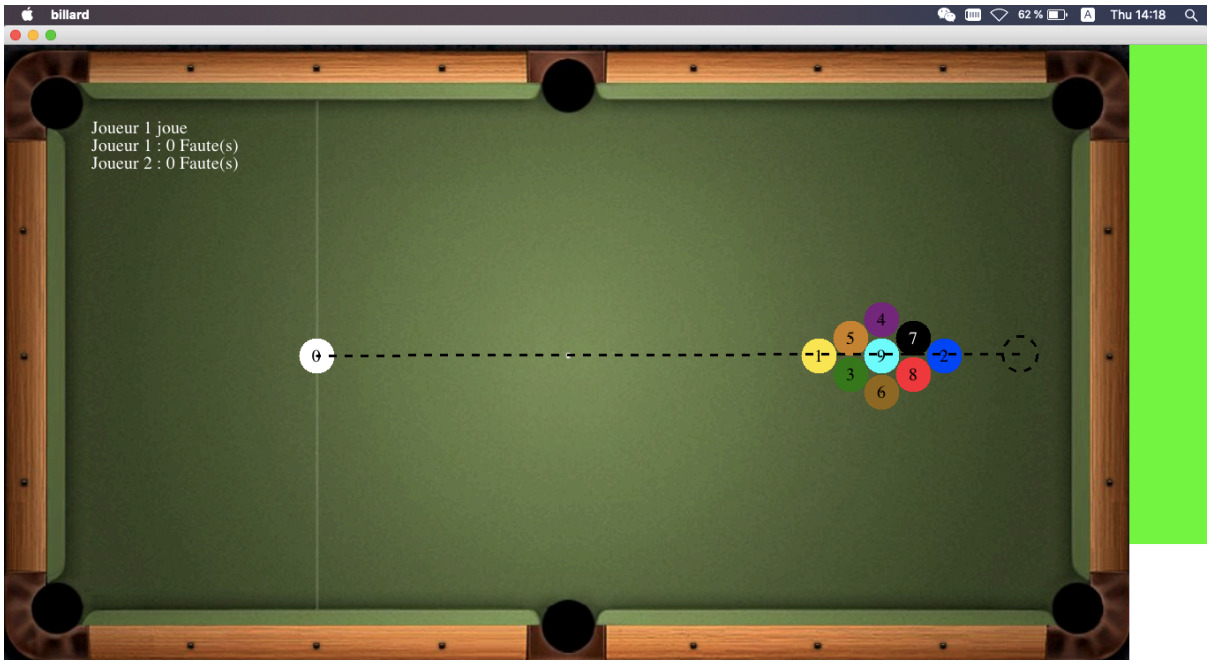
L'objet **boule** représente, comme son nom l'indique, les différentes boules présentes sur la table de billard. Une boule est caractérisée par sa position, sa vitesse, son rayon, sa couleur, son numéro et un boolean relatif au fait qu'elle soit rentrée ou non dans un trou.

C'est au niveau de la classe **table** qu'est codée la plus grande partie des instructions et commandes nécessaires au bon fonctionnement du jeu et à la visualisation du programme vu qu'elle définit l'interface graphique ainsi que les règles de jeu, les collisions etc…

La classe **billard** contient le main. C'est en l'exécutant qu'on peut se servir du programme.

Interface graphique

L'interface graphique consiste en une table de billard classique : une image qu'on a téléchargée sur internet (évidemment à des fins non commerciales). Sur la table viennent se placer les billes de billard sur des positions initiales définies.



La partie est pilotée par les mouvements de la souris. On a donc implémenté à cet effet un MouseMotionListener et MouseListener pour "écouter" les évènements de la souris qui sont traités pour décider des mouvements des billes sur la table. Une ligne en pointillés (Graphics 2D dashed Line) permet à l'utilisateur d'anticiper la direction dans laquelle il tire.

Règles de jeu

-La bille blanche doit commencer par toucher la bille présente sur la table dont le numéro est minimal.

-Si la bille blanche rentre dans un trou, touche une bille autre que celle dont le numéro est minimal ou si elle ne touche rien, cela compte pour une faute pour le joueur en question.

-En cas de faute, l'autre joueur peut placer la bille 0 là où il veut à gauche de la ligne blanche par un double clic sur la souris (en cliquant dans cette zone i.e à gauche de la ligne blanche verticale).

-La partie est terminée lorsque la balle numéro 9 est rentrée ou quand l'adversaire commet sa troisième faute. Le jeu est donc relancé.

Problèmes rencontrés et perspectives d'amélioration

On a eu beaucoup de mal à programmer la partie qui concerne la disparition des billes une fois rentrées dans les trous ainsi que les rebondissements aux voisinages des trous. Les parois ayant initialement été modélisées par un cadre rectangulaire, il nous a été difficile d'isoler des trous de forme circulaire. On estime que cette partie présente une marge d'amélioration malgré le résultat acceptable auquel on a abouti.

On a également rencontré des problèmes pour l'ajout d'effets sonores, d'autant plus que c'est une partie qu'on ne voit pas au programme de 2ème année de l'INSA ce qui a fait que nous n'avions pas vraiment de bases pour appréhender le traitement de fichiers audio sous java. On a eu des erreurs de compilation et de compatibilité de versions java entres nos différentes machines. Le programme contenu dans l'archive ne propose pas d'effets sonores, mais pour les activer, merci de bien vouloir décommenter les lignes prévues à cet effet (et de bien enlever le 'static' dans la méthode collision) .

Bref échéancier approximatif décrivant l'évolution de notre travail

Semaine 1 : Brainstorming d'idées, propositions de projets, 2 idées de projets retenues : résolution automatique de puzzles et billard.

Semaine 2 : Élimination du projet puzzles, adoption définitive du projet billard, mise en équations des lois mécaniques et réflexions sur l'architecture des données, début de la programmation, création des objets boule et vecteur.

Semaine 3 : Codage des fonctions selon les classes, premières ébauches d'interface graphique.

Semaine 4 : Rebondissements sur les parois, trous, ajustement de la force du tir par une barre d'intensité alternative et proportionnelle à la durée du maintien du clic.

Semaine 5 : Amélioration des rebondissements, ajout de la fonctionnalité multi-joueurs, ajout d'effets sonores, rédaction du rapport final.

Bibliographie

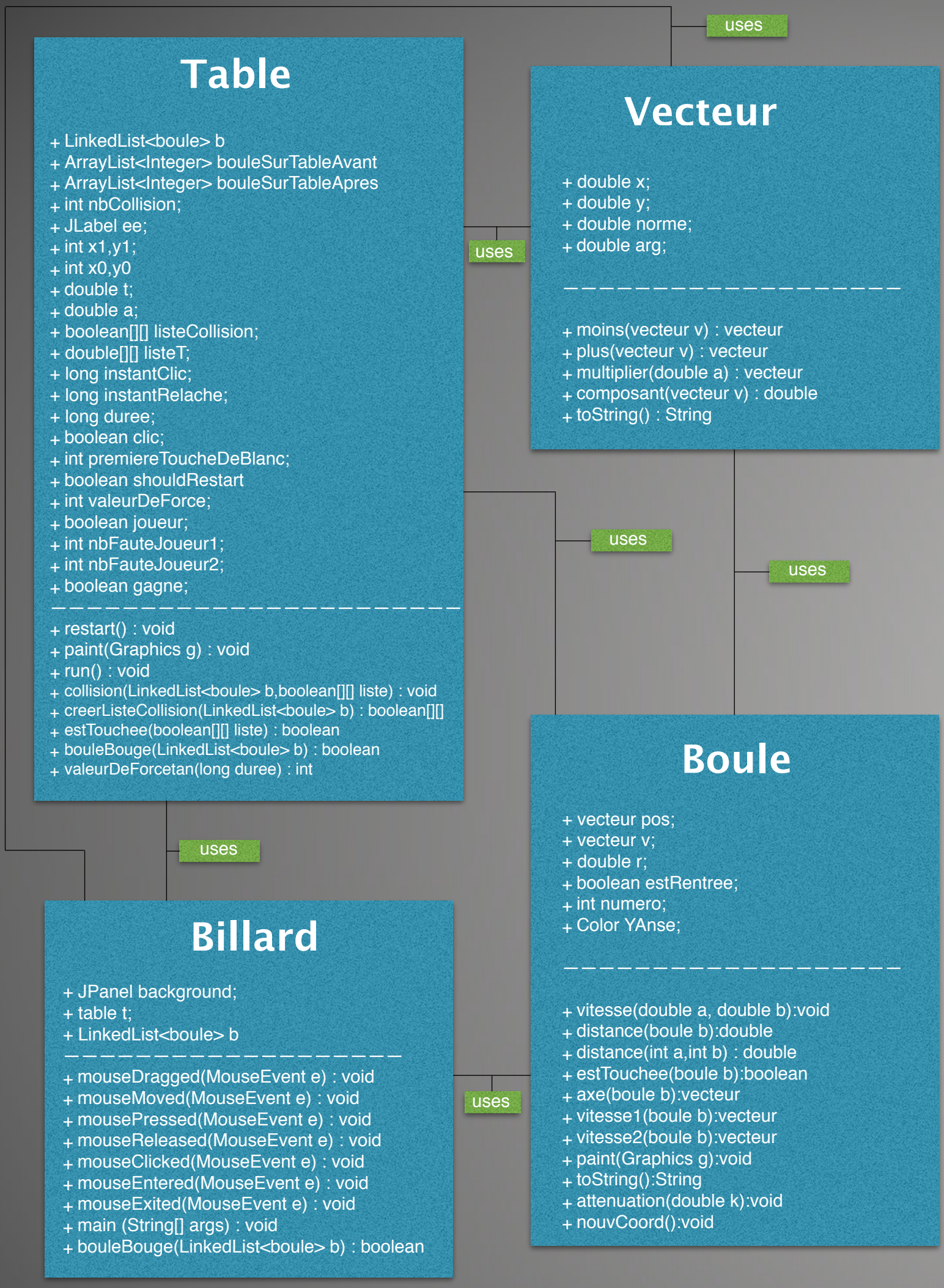
https://www.cnblogs.com/mumuxinfei

https://stackoverflow.com

https://docs.oracle.com/en/java/javase/12/

https://www3.ntu.edu.sg/home/ehchua/programming/java/j4a_gui.html

http://www.vnea.com/8-ball-rules.aspx



STRUCTURATION DES DONNÉES - DIAGRAMME UML