ENSTA Bretagne

Projet informatique

Le billard français

Professeur :
Jalil Boukhobza

Groupe :
Bérénice du Baret de Limé
Hortense Leynier

4 juin 2021

Table des matières

1 Description du projet

1.1 Introduction du projet

L'objectif de ce projet est de simuler un billard français. Aussi nommé Carambole, ce jeu se joue se joue à deux joueurs avec 3 ou 4 boules, voir plus en cas de joueur débutant. Le but est de toucher d'un même mouvement avec sa balle au moins deux autres boules distinctes avant de s'arrêter.

Nous voulons donc faire une simulation du plateau de billard, dans laquelle on peut observer les boules évoluer, rebondir, et s'arrêter sur le tapis.

1.2 Lancer le jeu

Pour lancer le programme, faites tourner le module interface.

Par défaut, trois balles se trouvent sur le plateau. Si vous souhaitez jouez avec plus de trois balles, remplacer la ligne du module interface suivante window = BillardUI() (ligne 447) par la ligne window = BillardUI(4) pour quatre balles par exemple.

Il ne vous reste plus qu'à appuyer sur 'Commencer la partie' pour jouer ou 'Charger' pour reprendre une partie !

Pour faire un tir, appuyez dans la direction voulue par rapport à votre balle (grise pour le joueur 1, grise avec un point rouge pour le joueur 2, les billes oranges étant les billes neutres). Plus vous vous positionnez loin de la boule, plus le coup va être fort.

1.3 Les différentes pistes envisagées

Pour effectuer notre mouvement, nous avons considérés l'équation différentielle suivante qui décrit la vitesse d'une bille de masse m soumise à la seule force de frottement solide f :

$$\frac{\mathrm{d}v_o}{\mathrm{dt}} + \frac{f}{m}v_o = 0 \tag{1}$$

Deux manières de la résoudre se sont rapidement imposées à nous : la méthode d'Euler ou celle de Runge-Kutta.

La méthode du RK2 est plus précise, mais la méthode d'Euler permet une simplification d'application et une mise en place plus intuitive.

C'est donc vers cette option là que nous nous sommes tournées. La méthode d'Euler s'applique de cette manière :

$$x_{n+1} = x_n + hf(x_n, u_n) \tag{2}$$

où la suite (x) correspond à l'évolution de la vitesse de la balle.

Dans une version antérieure du programme, nous avions envisagé de réaliser le mouvement de chaque balle dans son intégralité, balle par balle. Mais cette idée engendrait de très nombreuses problématiques dans la gestion des collisions entre les balles et dans la gestion de l'IHM qui s'en suivrait. Nous sommes donc revenus sur l'idée proposée dans le sujet qui consiste à gérer les mouvements de toutes les balles durant un petit intervalle de temps noté h.

Nous avions également considéré le plateau comme une classe beaucoup plus importante qui devait également gérer les fonctions de collisions. Nous l'avons finalement simplifié pour le réduire au simple contenant des balles et ainsi alléger le code et tout en nous permettant de gérer plus simplement les collisions, les rebonds et le changement de vitesses des balles lors des contacts.

1.4 Les hypothèses simplificatrices

1.4.1 Les balles : un simple point

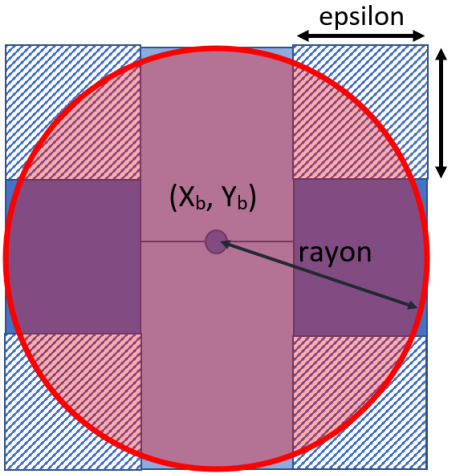
Comme indiqué sur le sujet et par soucis de simplification majeure, nous avons réduit les balles à des simple points pour restreindre les effets applicables à la balle. Cela simplifie grandement les équations et nous permet de faire un jeu plus facilement codable ou jouable simplement par des néophytes. En effet, appliquer uniquement un vecteur vitesse au centre de la balle simplifie énormément le gameplay du jeu et permet des approximations très acceptables pour la précision que nous pouvons obtenir avec la souris.

Notre jeu est destiné à des débutants ou des néophytes dans le domaine qui découvrent le billard et pas à des professionnels du sport. Ainsi, cette approximation n'est pas si loin de la réalité dans le billard bas-niveau en général. La physique n'est pas respectée, mais les débutants tentent à limiter le plus possible ces effets pour garder plus facilement le contrôle de leur balle.

1.4.2 Des collisions codifiées

Nous avons également dû considérer une deuxième simplification concernant les collisions et la manière dont les balles rebondissent entre elles.

En premier abord, les balles étaient carrées, puis nous avons affiné le modèle et désormais les balles sont modélisées comme suit :



Tout d'abord, si une autre balle (repérée par ses coordonnées : Xe, Ye) rentre dans le cercle rouge, alors il y a collision. Pour le code, l'argument est que la distance entre les deux centres est de moins de 2*rayon.

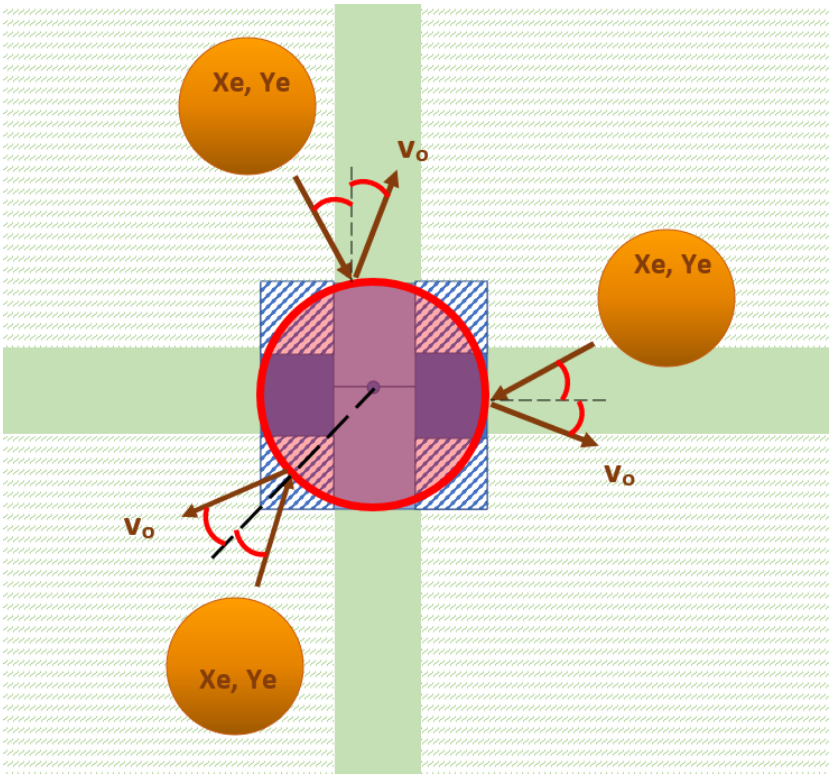
Ensuite, une fois la collision avérée, on repère dans quelle zone se situe la balle principale par rapport à la balle-cible.

Si elle entre en contact au niveau des zones bleu clair (bande verticale), on est en présence d'un contact : Top ou Bottom et la vitesse de la balle principale est modifiée en conséquence pour effectuer un rebond comme elle le ferait selon une bande horizontale du bord du plateau.

Si elle entre en contact au niveau des zones bleu foncé (carrés à droite et à gauche), on est en présence d'un contact : Left ou Right et la vitesse de la balle principale est modifiée en conséquence pour effectuer un rebond comme elle le ferait selon une bande verticale du bord du plateau.

Si elle entre au contact au niveau d'une zone rayée (les coins), alors on est en présence d'un contact Coin et la vitesse est inversée totalement.

Voici trois différents cas de figure :



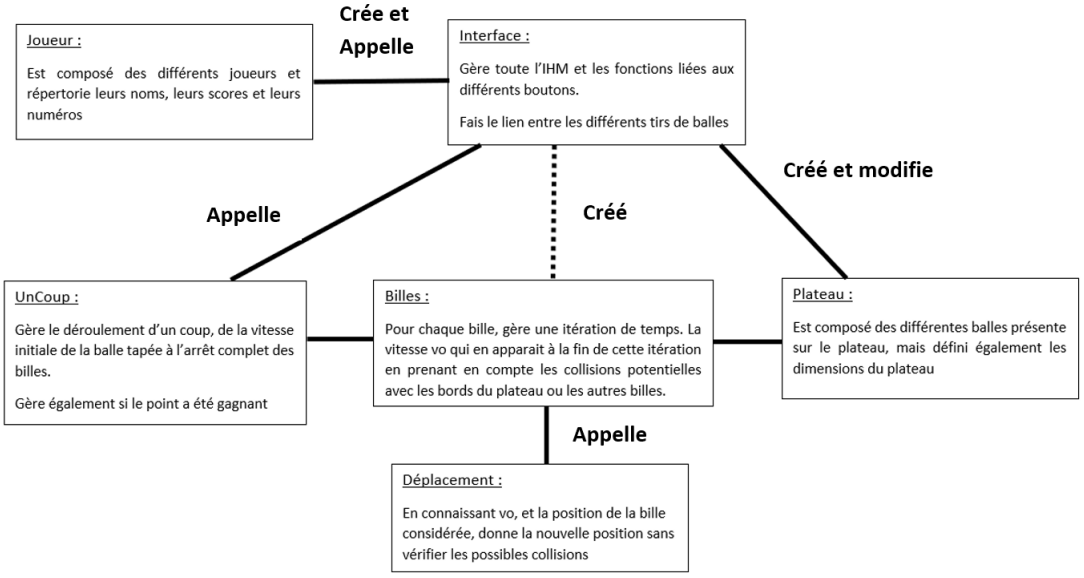
On remarque donc que, si les balles peuvent se percuter en tout point, leurs type de rebond est limité en 8 cas différent. Les angles sont calculés par rapport à 4 droites différentes, Ox, Oy, y =x et y=-x.

Dans tous les cas, la balle cible reçoit la vitesse de la balle principale tandis que celle-ci voit sa vitesse changer de sens à cause du rebond.

C'est une hypothèse aussi très réductrice pour la physique du jeu, mais bien plus simple à coder de manière naïve. De plus, dans le temps imparti pour ce projet, cette simplification nous a semblé une manière efficace de symboliser les rebonds pour avoir un rendu visuellement acceptable et jouable.

2 Description du code

2.1 Une vision générale



Nous avons une conception des classes proche de ce que demandait la consigne en gérant chaque boule sur un petit intervalle de temps : *"Le programme devra donc parcourir l'ensemble des boules, intégrer leur trajectoire sur un petit intervalle de temps, et recommencer tant qu'il reste des boules en mouvement."*

Un diagramme de classe plus précis et détaillé est disponible en Annexe.

2.2 La classe BillardUI

La classe BillardUI, présente dans le module interface, gère la partie en général ainsi que l'interface homme-machine(IHM). Après avoir créé l'IHM et généré les boules et les joueurs grâce à la méthode generer(), la partie se lance au clic sur le plateau en appelant la méthode MousePressEvent() qui lance la méthode tour_joueur().

2.2.1 La méthode tour_joueur()

Cette méthode prend en compte la position du clic de la souris par le joueur et en déduit le vecteur vitesse appliqué à la balle. Elle lance alors tous les calculs et récupère à la fin les différentes positions de toutes les balles au cours du temps. Elle appelle également la méthode calcul_des_points() qui calcule les points de chaque

joueur et décide de quel joueur doit jouer. Elle lance de plus le timer pour la simulation.

2.2.2 Les méthodes sauvegarder() et charger_partie()

Ces méthodes permettent la gestion des parties dans le temps via l'écriture et la lecture de fichier. En effet, le jeu étant long (il faut parvenir à 300 points pour gagner), pouvoir reprendre une partie est quasiment essentiel.

Pour cela, les données nécessaires au jeu sont enregistrées via l'écriture d'un fichier dans la méthode sauvegarder(). Ce fichier peut être un nouveau fichier ou peut écraser celui décrivant un état précédent de la partie. Dans ce fichier, on écrit la dernière position de chaque balle du plateau, le joueur qui doit jouer lors de la reprise, et enfin le nom et nombre de points de chacun des joueurs. La méthode charger_partie() permet, lors de l'ouverture de l'application, de reprendre une partie déjà commencée et enregistrée, quelque soit le nombre de balles en jeu.

2.3 La classe Joueur

La classe Joueur permet simplement de récupérer quelques informations sur les deux adversaires. Elle permet notamment de connaître leur nom et leur nombre de points. Elle ne dsipose d'aucune méthode remarquable.

2.4 La classe Plateau

La classe Plateau, disponible dans le module du même nom, hérite du type list. Elle consiste en la liste des balles générée par le biais la classe Billes, mais dispose aussi d'autres variables d'instance comme la taille du plateau, le nombre de billes (initialisé à 3 mais pouvant aller au-delà), le rayon des billes. Elle dispose de trois méthodes : une renvoi les dimensions du plateau, l'autre le nombre de balles et la dernière generation_bille_neutre().

C'est au travers de la classe Plateau que la position des billes est modifiée au cours du jeu.

2.4.1 La méthode generation_bille_neutre()

Cette permet de créer des billes neutres, c'est à dire non assignée à un joueur. Elle initialise notamment la position de la balle à un emplacement libre du plateau grâce à une fonction récursive, passant en revue les positions des autres balles déjà existantes.

2.5 La classe Billes

La classe Billes, présente dans le module éponyme, crée une bille et gère ses coordonnées et son environnement.

2.5.1 La méthode mouvement()

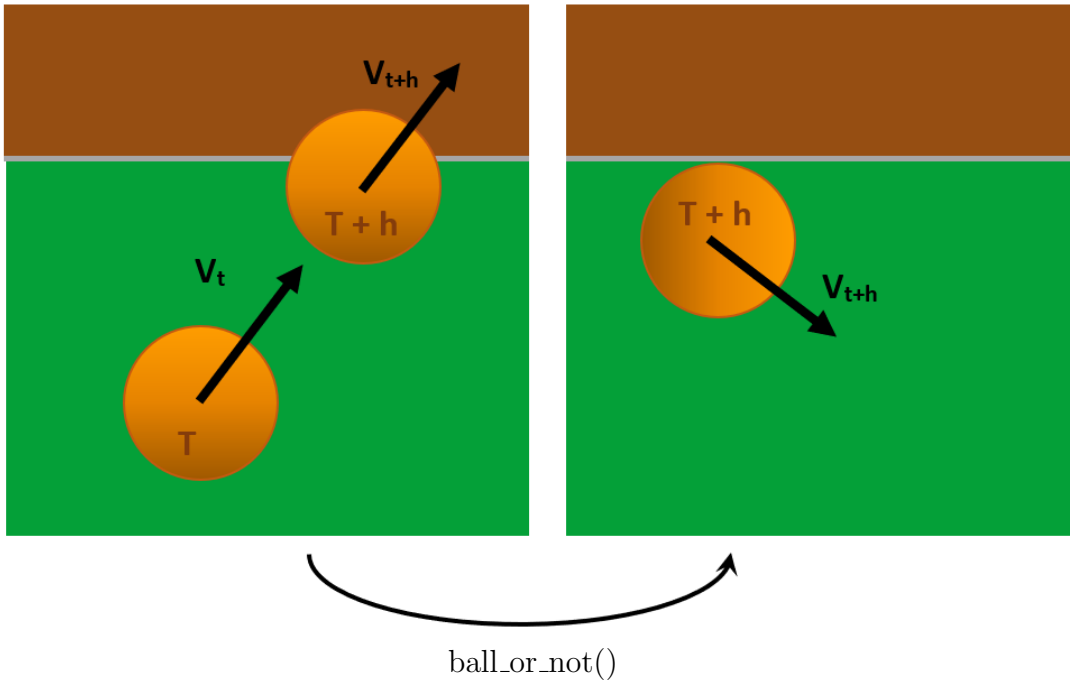
La méthode mouvement() est la méthode principale de la classe. Elle gère le mouvement de la bille durant un petit espace de temps et retourne sa position et sa vitesse à la fin du court intervalle de temps.

Elle commence par appeler la classe Déplacement pour pouvoir calculer avec la méthode d'Euler les coordonnées théoriques de la balles. Puis elle appelle les méthodes wall_or_not() et ball_or_not().

Elle gère également la transmission de vitesse à la balle touchée et les facteurs de diminutions des vitesses en cas de contact.

2.5.2 La méthode wall_or_not()

Cette méthode permet de détecter les collisions de la balle avec les bords du plateau de billard et de changer le mouvement de la balle dans le cas d'une collision. Elle repositionne la balle contre la bande en cas de débordement de celle-ci.



2.5.3 La méthode ball_or_not()

La méthode ball_or_not() reprend la même idée que la méthode wall_or_not(), c'est à dire qu'elle détecte s'il y a eu ou non collision mais détecte, le cas échéant, avec plus de subtilités le lieu de collision des deux balles. Les différentes formes de collisions possibles sont expliquées précédemment dans la partie 1.4.2.

2.6 La classe UnCoup

La classe UnCoup, disponible dans le module du même nom, gère le déroulement d'un tour, c'est à dire du tir d'un joueur. Elle est composée de deux méthodes, l'une gérant spécifiquement les balles et l'autre assignée au décompte des points des joueurs pour un tir donné.

2.7 La classe Deplacement

La classe Deplacement gère toute la physique du jeu. En particulier, elle décrit le mouvement de chaque bille pendant un cours instant dans la méthode mouvement_unitaire(). Cette classe présente aussi d'autres méthodes, vestiges de la première version du code sans IHM, qui permettaient notamment d'afficher dans un graphique les positions initiales et finales des balles.

2.7.1 La méthode mouvement_unitaire()

Cette méthode est dédiée à gérer le mouvement théorique d'une balle, en renvoyant les nouvelles positions de la balle selon la méthode d'Euler.

2.8 Les tests

On va effectuer divers tests au cours du jeu, notamment sur les billes afin de s'assurer qu'elles sont sur le plateau à tout moment. On commence par vérifier que les billes créées par le plateau sont bien de type Billes. Ensuite, on vérifie qu'une bille est bien associé à son numéro. Puis viennent les tests sur les positions. Dans un premier temps, on teste si les positions des billes se trouvent dans les limites du plateau à l'initialisation de ce dernier. Dans un second temps, on teste si, après le coup d'un joueur, l'ensemble des billes sont encore sur le plateau.

D'autres tests sont effectués : on fait un test sur le type de la classe Plateau, qui hérite de list. On vérifie aussi le type du dictionnaire créé par la méthode on_tire() recensant les positions et vitesses et chacune des billes.

3 Conclusion

3.1 Les figures imposées et leurs résolutions

Pour la réalisation de ce projet, 7 figures étaient imposées parmi une liste de 11.

Nous en avons réalisées 8 que voici :

Factorisation du code : Nous avons trois modules-objects très distincts : les boules, les joueurs et le plateau.
De plus, les classe Interface, Déplacement et UnCoup gèrent respectivement l'IHM, le déroulé d'un coup et le mouvement des balles.

Documentation et commentaires du code : Le code est commenté et documenté.

Tests unitaires : Cf le point 2.8 et le fichier joint.

Création d'un type d'objet avec au moins deux variables d'instances : Le plateau est un objet qui prend pour instance les dimensions de la table et les caractéristiques des balles.
Les billes sont des objets qui prennent pour instance les paramètres propres de chaque bille à un instant t : ses coordonnées, sa vitesse, mais également son rayon et son identifiant.

Fonction récursive dans le cœur du projet : Une fonction récursive sert à gérer la création des billes neutres pour leur éviter de se superposer aux balles déjà existantes sur le plateau.
Cette fonction est très utile en cas de tests pour simuler plus d'une trentaine de balles sur le plateau sans chevauchement impétueux.

Héritage au moins entre deux types créés : Pour nous permettre d'afficher des images bien précises en fonction des différentes billes, nous avons utilisée des classes filles pour chaque type de bille.

Dans une version ultérieure du code, ces classes filles géraient également les points gagnant et la répartition de scores des joueurs. Mais pour factoriser et simplifier le code le plus possible, nous avons finalement décidé de le décaler dans la classe Interface.

Héritage depuis un type intégré : Pour contenir plus facilement les différentes billes et pouvoir les parcourir simplement, la classe Plateau hérite du type List.

Lecture/ écriture de fichiers : Cette contrainte s'est immédiatement cochée quand nous avons décidé de pouvoir enregistrer les parties et charger des parties déjà existantes.

3.2 Les différents tests effectués

Le test le plus classique consiste à effectuer une partie classique de Carambole : après avoir lancé une nouvelle partie, on écrit le nom des deux joueurs (ou plus souvent une suite de caractère trouvée de manière très artistique en écrasant le clavier au hasard), puis on joue à proprement parler.

Tout d'abord des coups classiques qui pourraient être joués par des vrais joueurs. En s'aidant des bandes pour percuter les balles ennemies et tenter de remporter un point.

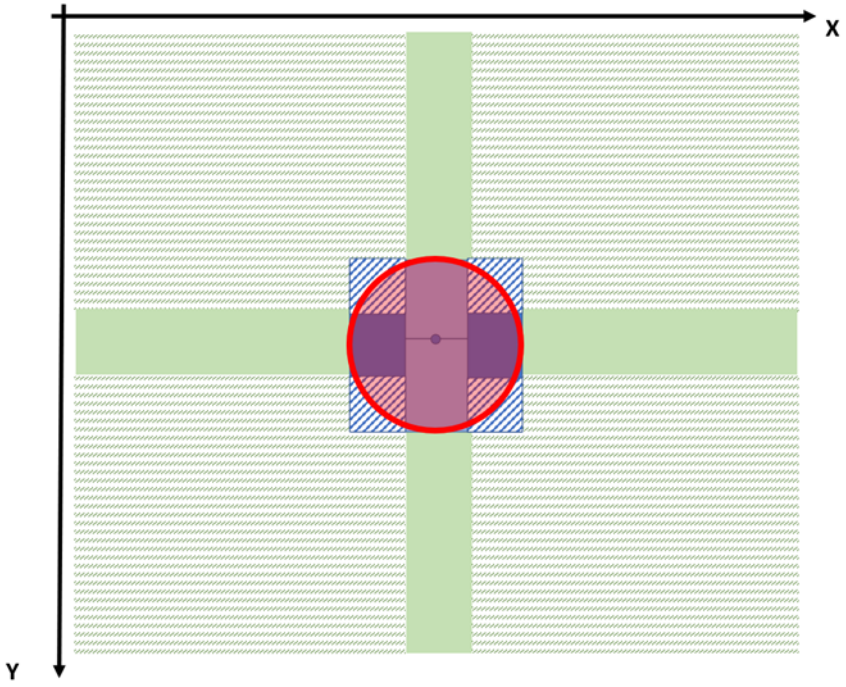
On enregistre ensuite la partie, puis on recharge la même partie dans une nouvelle fenêtre pour vérifier que les données ont bien été mémorisées aux bons endroits

Puis on a testé les limites du code en mettant des coups le plus fort possible en mettant le plus de distance entre la balle tirée et le clic du tir. On a également testé de frapper des balles déjà en contact pour les forcer à rentrer l'une dans l'autre.

On a ensuite répété tous ces tests avec 4 boules sur le plateau, puis 5, puis 8 puis 15 voir même jusqu'à 20 ou 30 pour pister vraiment les bugs finaux et les erreurs de billes qui s'accrochent l'une à l'autre dans certaines configurations.

3.3 Les limitations

La limitation principale est liée à la physique engagée lors des collisions. En effet, la manière dont est renvoyée la bille dépend dans quelle zone se trouve son centre :



Ainsi, la physique du contact est respectée au centre de chaque zone (quand la balle tape exactement sur l'axe Ox, Oy ou sur l'axe d'équation y = x et y=-x). Mais plus la zone d'impact s'éloigne de ces points, mois la physique est respectée, le point critique étant aux limites des zones.

3.4 Les perspectives et améliorations

Il y a tellement de choses que nous aurions aimé faire pour faire évoluer notre code si nous avions eu plus de temps.

La **première chose** que nous aurions aimé approfondir est le système de collisions. En effet, pour l'instant, la collision se gère grâce à des zones de contact et, si cela est fonctionnel, cela reste assez peu réaliste dans certains cas.

Nous aurions aimé mettre à la place un système de droite normale et de rebond avec une inversion de l'angle d'incidence. Ce serait une évolution du système que nous avons aujourd'hui dans notre code où cette méthode n'est appliquée que sur 6 axes au lieu de pouvoir s'adapter a une infinité de droites possibles

La **deuxième amélioration** possible aurait été de coller plus aux différentes variantes du billard français en incluant plus de règle spécifiques. Comme par exemple la règle des trois bandes. Avec un système de case à cocher comme nous avions inclus dans une version antérieure du code pour choisir le joueur actif, nous aurions choisi le mode de jeu : partie libre, la bande, la trois bandes...

Cette fonctionnalité était simple à instaurer, mais n'était pas très utile pour notre jeu. En effet, malgré nos talents pour le billard dans la vraie vie, la démonstration durant la présentation va montrer combien jouer avec une souris sur un écran est difficile.

De ce fait, il est déjà assez difficile et lassant de marquer plus de 5 points dans une partie. Rendre le jeu plus difficile encore en restreignant les coups n'aurait pas apporté beaucoup à l'expérience de jeu.

Une **autre idée d'amélioration** nous est venue alors que nous présentions notre code à des amis : Nombre d'entre eux ont eu la même réaction *"où sont les trous ?"*. Nous avons alors remarqué que notre billard français était facilement transposable en son homologue américain, plus habituel dans l'esprit des gens.

Il aurait fallu uniquement instaurer des cercles noirs dans le fond et définir ces zones comme des espaces à part, où la balle ne peut pas ressortir, la faisant ainsi disparaître du plateau pour les joueurs.

Toutefois, nous nous sommes également rendu compte que le système de règles et de comptage de points étaient très différent et nécessitait de créer un tout autre jeu. Nous avons seulement posé les mécaniques des jeux de billes sur un plateau. Ces

bases sont désormais adaptables sans aucun soucis pour créer d'autre variantes.

4 Annexes : diagramme des classes