OPTIMISATION DU COUP À JOUER AU JEU DE BILLARD

21 avril 2015

Figure 1 – Knight, Charles : *"Old England : A Pictorial Museum"* (1845)

Micaël Lemelin (111 079 627)
Maxime Lefrançois (111 072 506)
Université Laval

1 Introduction

Apparu au 15^e siècle en Europe, le billard est un sport dont la pratique a évolué et qui continue encore aujourd'hui de déchaîner les passions. Bien plus qu'il n'y paraît, ce sport repose sur les mathématiques et la physique. Qu'il s'agisse de coups simples ou de manoeuvres plus complexes, tout repose sur la géométrie et la physique des corps en mouvement, ce que les joueurs professionnels savent plus que quiconque.

Et si l'on pouvait calculer, pour une table donnée, le meilleur coup à jouer ? Nous faisons l'hypothèse que c'est possible et qu'un modèle mathématique va nous permettre de trouver ce coup.

2 Description du problème

Le billard est un sport de table ayant plusieurs variantes. Nous utilisons les règles définies par le *Billiard Congress of America*, un organisme régissant le billard aux États-Unis et au Canada. Voici quelques fautes que nous allons tenir compte dans notre travail :

— Ne pas frapper de balles
— Frapper une balle adverse en premier
— Empocher la noire avant d'avoir entré toutes ses balles.

2.1 Fonction objectif

Nous voulons maximiser la fonction objectif $O(n_{i1}, n_{f1}, n_{i2}, n_{f2}, p)$ où n_{i1} est le nombre initial de balles du joueur et n_{f1} le nombre final, n_{i2} est le nombre initial de balles adverses et n_{f2} le nombre final. p est une pénalité relative au coup.

$$O_1(n_{i1}, n_{f1}, n_{i2}, n_{f2}, p) = n_{i1} - n_{f1} + n_{f2} - n_{i2} - p$$

FIGURE 2 – Fonction objectif donnant la valeur d'un coup au billard

On appliquera une pénalité infinie lorsqu'on fait une des 3 fautes décrites précédemment. De la même façon, une pénalité choisie par l'utilisateur sera appliquée lorsque la blanche est empochée.

2.2 Équipement

Plusieurs grandeurs de table de billard sont utilisées pour plusieurs jeux de billard. Le *Billiard Congress of America* définit une table réglementaire comme une table ayant un tapis avec une proportion de 2 : 1 et comportant six poches, réparties de la façon suivante suivante :

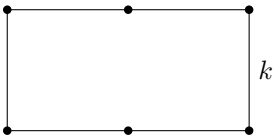

FIGURE 3 – Proportions d'une table de billard standard et emplacement des poches

Dans le cadre de notre travail, nous utiliserons une table de deux mètres de long. La table mesurera donc un mètre de large.

2.3 Unités de mesure et constantes

À moins d'avis contraire, les distances sont en mètres, les vitesses en mètres par seconde, les accélérations en mètres par seconde carrée et les masses en kilogrammes.

Pour calculer la friction et modéliser les collisions, nous avons besoin des constantes suivantes : le coefficient de frottement (μ_c), le rayon d'une balle (r), la masse d'une balle (m) et la constante gravitationnelle (g).

$$\mu_c = 0.01 \qquad m = 0.17$$
$$r = 0.286 \qquad g = 9.80665$$

FIGURE 4 – Constantes nécessaires aux calculs physiques[1]

2.4 Mouvement d'une balle

Au billard, le point de contact de la queue sur la blanche permet d'appliquer une rotation sur la balle. Pour simplifier les calculs physiques, le joueur ne pourra frapper qu'au centre de la balle. Autrement, il aurait fallu tenir compte de variables physiques telles que les vitesses et accélérations angulaires.

Une balle en mouvement est soumise à la friction entre elle et la table. Cette friction est la seule accélération que subit la balle, outre l'impulsion de départ avec la queue. La friction a la même *direction* que la vitesse, mais *est de sens opposé* à cette dernière.

Voici une formule permettant de calculer les composantes x et y de la friction à partir des composantes x et y de la vitesse :

$$a_x = -mg\mu_c * cos\left(atan\left(\frac{v_y}{v_x}\right)\right)$$
$$a_y = -mg\mu_c * sin\left(atan\left(\frac{v_y}{v_x}\right)\right)$$

FIGURE 5 – Formule de l'accélération d'une balle en fonction de sa vitesse

Dans notre cas, l'accélération reste constante tant que la vitesse ne change pas de direction ou de sens. L'accélération est donc calculée à chaque changement de vitesse.

La position d'une balle dans le temps ($p(t)$) est définie comme la somme de sa position initiale (p_i), sa vitesse initiale (v_i) multipliée par le temps et une demie de l'accélération (a) multipliée par le temps élevé au carré :

$$p(t) = p_i + v_i t + \frac{at^2}{2}$$

FIGURE 6 – Formule de la position d'une balle en fonction du temps

Cependant, il faut bien saisir que la constante accélération (*décélération*) de la balle finira par la faire reculer à l'infini. Nous devons donc être en mesure de calculer le temps t_{stop} auquel une balle s'immobilise. Heureusement pour nous, les trajectoires de notre modèle (sans vitesse et accélération angulaire) sont linéaires. Nous pouvons donc calculer le temps d'arrêt plutôt facilement avec cette formule :

$$t_{stop} = \frac{||v_i||}{||a_i||}$$

FIGURE 7 – Formule du temps t_{stop} nécessaire à une balle pour s'immobiliser

2.5 Collisions entre balles

Comme toutes les balles sont de même rayon et de même poids, cela simplifie grandement les calculs des vecteurs de vitesse résultants d'une collision.

Dans notre modèle, les collisions sont toujours entre 2 corps. On suppose donc qu'il n'y a pas de collisions simultanées entre 3 balles ou plus. Les collisions entre balles au billard sont presque parfaitement élastiques, c'est bien connu. Selon un professeur de génie mécanique de la *Colorado State University*, les coefficients de restitution vont de 0.90 à 0.96 pour les balles de billard[2]. Pour simplifier le modèle, les collisions entre balles seront traitées comme élastiques. Les collisions entre balles et coussins, étant elles aussi presque inélastiques, auront quant à elles un coefficient de restitution égal à 0.9.

2.5.1 Collisions entre une balle en mouvement et une balle immobile

Après une collision élastique entre une balle en mouvement et une balle immobile, les 2 vecteurs de vitesses résultants ont une séparation angulaire de quatre-vingt-dix degrés. Ces vecteurs forment donc une base β et on cherche d'abord à trouver les composantes du vecteur de vitesse de la balle en mouvement dans cette base. Les 2 vecteurs qui composent le vecteur initial seront les vecteurs de vitesse résultants.

2.5.2 Collisions entre 2 balles en mouvement

La collision entre 2 balles en mouvement est plus complexe. Tout comme le cas précédent, la vitesse de chaque balle doit être conservée, car notre modèle simule des collisions élastiques entre les balles. On trouve donc d'abord le vecteur c allant de la balle A à la balle B, puis on le normalise pour obtenir c_n. La vitesse de chaque balle (A et B) est ensuite donnée par la formule suivante :

$$v_A = v_{i_A} + c_n * (v_{i_A} \cdot c_n - v_{i_B} \cdot c_n)$$
$$v_B = v_{i_B} + c_n * (v_{i_B} \cdot c_n - v_{i_A} \cdot c_n)$$

FIGURE 8 – Formule pour calculer les vecteurs de vitesse résultants lors d'une collision entre deux balles (A et B) en mouvement

3 Recherche exhaustive de la solution

Il est trivial de remarquer que le déroulement d'un coup est déterminé par en fonction du vecteur de vitesse donné à la première balle frappée par la queue de billard. Les vecteurs résultant d'une collision entre 2 balles de billard étant calculés à l'aide de relations trigonométriques, il est moins approprié d'exprimer notre modèle avec un solveur linéaire comme *Choco*. Par conséquent, il a été décidé de construire un solveur spécialisé pour le problème.

3.1 Approche orientée poches

Certains problèmes sont résolus efficacement en partant du but et en revenant jusqu'à l'état initial. Pour notre problème, il s'agirait de partir de chacune des 6 poches et de voir si on peut y entrer des balles. Or, pour chaque balle, au départ, il existe 6 poches pouvant l'accueillir.

Cependant, encore faut-il déterminer l'ordre d'interactions qui mène au résultat voulu. Cet ordre n'étant pas unique non plus, l'arbre de recherche grandit de manière exponentielle. Pour cette raison, nous avons abandonné cette avenue.

3.2 Approche orientée blanche

Cette méthode consiste à partir de la blanche pour trouver la solution. Étant donné une blanche ayant un vecteur de vitesse initiale, on simule et on calcule la valeur objective du coup. Pour cette méthode on divise le vecteur de vitesse en deux composantes : l'angle et la norme.

Pour chaque angle, on commence généralement par la plus grande vitesse possible. Ainsi, lorsque l'on constate qu'avec un angle a, la vitesse de la blanche ne lui permet pas d'entrer en contact avec au moins une balle du joueur, on peut rejeter dès lors cet angle pour toutes les vitesses inférieures restantes.

3.2.1 Filtrage 1^{ere} balle frappée

Lorsque la blanche entre en collision avec une première balle, à moins qu'elle donne toute son énergie à la balle, on passe d'une balle en mouvement à deux. Par conséquent, on doit profiter au maximum de la simplicité du modèle avant la première collision pour filtrer des angles.

Pour avoir une valeur objective qui est intéressante, il est évident qu'on doit frapper au moins une balle avec la blanche, que cela soit fait avec ou sans rebonds sur les coussins. On cherche donc à obtenir les intervalles d'angles qui pourraient permettre à la blanche d'atteindre les balles du joueur et ainsi filtrer tous les coups qui n'engendrent pas de collisions.

Pour obtenir une liste d'intervalles, nous cherchons d'abord l'intervalle unique permettant de frapper chaque balle et fusionnons les intervalles qui se chevauchent. Pour trouver l'intervalle d'angle pour frapper une balle, considérons le cas général suivant :

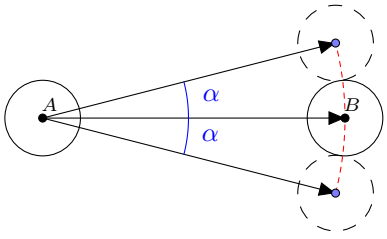

FIGURE 9 – Angles d'attaques possibles pour une balle frappée directement

L'angle α est l'angle qui nous permettra de créer notre intervalle. Pour le calculer il suffit d'un peu de trigonométrie. En effet, on peut représenter le vecteur entre la balle et sa position à une borne de l'intervalle comme l'hypoténuse d'un triangle rectangle.

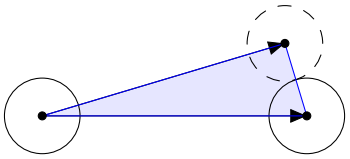

FIGURE 10 – Représentation du triangle rectangle utilisé pour résoudre l'angle α

Une fois l'angle α trouvé, on calcule l'intervalle d'angle à partir de l'angle a entre la blanche et la balle visée.

$$[a - \alpha, a + \alpha]$$

FIGURE 11 – Intervalle d'angles pour frapper une balle

Une fois tous les intervalles recueillis pour les balles du joueur, on serait tenté de croire que le filtrage des angles est complet. Mais encore faut-il enlever les intervalles des balles adverses qui sont plus proches de la blanche. Cette opération qui semble plutôt simple est en fait un casse-tête assez intéressant. On cherche un algorithme qui, à partir de deux ensembles d'intervalles d'angles, produit un ensemble qui est le résultat de la soustraction d'un ensemble à l'autre. De plus, cette soustraction doit avoir lieu uniquement lorsque l'intervalle du second ensemble a une priorité supérieure ou égale à l'intervalle du premier ensemble.

Pour aider à la compréhension, l'algorithme développé pour cette problématique considère chaque ensemble comme une couleur. Les intervalles bleus sont ceux auxquels on soustrait les intervalles rouges. On rassemble d'abord toutes les bornes des intervalles et on les trient par ordre de valeur. Une fois triés, la logique est d'itérer au travers de ces bornes tout en identifiant à chaque itération la priorité maximale bleue en cours et la priorité maximale rouge en cours. Pour ce faire, on établit une certaine hiérarchie entre les bornes inférieures des intervalles. Dans cette structure, une borne enfant a toujours une priorité inférieure ou égale à celle de son parent. Aussi, pour être un enfant éligible, l'intervalle de la borne enfant doit commencer avant ou pendant l'intervalle de la borne parent et se terminer après. On crée ainsi une sorte de succession de bornes où la borne à la racine est la borne de priorité maximale. Au travers des itérations, à chaque borne supérieure rencontrée, on désigne la nouvelle borne à priorité maximale comme étant la borne qui succède à la borne à priorité maximale courante. On peut alors

déterminer si les bleus gagnent ou si les rouges gagnent à chaque borne. Dès que les bleus commencent à gagner, on note la borne. Du moment qu'ils ne gagnent plus, on ajoute l'intervalle entre la borne notée et la borne courante à la liste d'intervalles. Pour les besoins du rapport, les calculs pour déterminer la structure de succession sont épargnés au lecteur. La complexité algorithmique temporelle de l'algorithme développé est en pire cas de n^2, et dans tous les cas bornée inférieurement par la complexité du tri fusion de *Java* ($\Theta(n \log n)$).

Avec un tel algorithme, il suffit de donner comme priorité le négatif de sa taille à un intervalle pour pouvoir trouver un ensemble d'intervalles qui sont la différence entre deux ensembles où les plus petits intervalles sont prioritaires. En plus, cet algorithme fusionne les intervalles qui se chevauchent créant ainsi une partition d'angle filtrée pour notre coup initial.

Maintenant, il nous reste à régler le cas des coups par la bande. Jusque là, il était supposé que nous frappions toujours les balles directement. Cependant, notre solveur ne serait pas digne s'il ne pouvait faire de coups avec rebonds. C'est là que nous avons identifié, avec l'aide de notre professeur, que nous pouvions considérer les rebonds comme un déplacement dans une table imaginaire. Nous concevons donc une table virtuelle ayant les réflexions des balles selon les 4 coussins principaux. Afin de permettre plusieurs rebonds, on crée autour de la table initiale suffisamment de tables réfléchies pour qu'il soit impossible que la blanche *sorte* au-delà des tables.

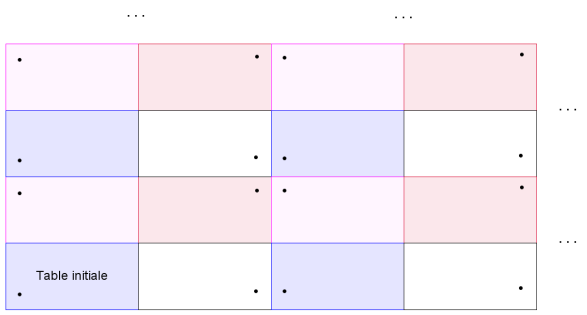

FIGURE 12 – Réflexions d'une table de billard n'ayant qu'une seule balle

Il est intéressant de noter que les tables de même couleur sont identiques et donc qu'il suffit de calculer 3 tables et de les répliquer dans l'espace pour construire nos tables à l'infini. Dans le code, on part de la table initiale et on crée les réflexions au-dessus d'elle et à sa droite. À la toute fin, on met la blanche à sa position dans la table du centre.

À partir de la table virtuelle et de la nouvelle position de la blanche, on génère les intervalles permettant d'atteindre les balles du joueur comme vu précédemment. Cela permet de créer des intervalles facilement tout en ignorant la complexité des rebonds. Au final, on obtient des intervalles d'angles pour lesquels la blanche touchera une balle du joueur en premier que ce soit sans ou avec rebonds. Seule ombre au tableau, il est possible qu'une poche se trouve entre la blanche et une balle dans une autre table virtuelle.

Nous avons constaté de façon empirique que cette méthode est très efficace : Elle filtre en moyenne plus des deux tiers de tous les angles possibles pour la blanche.

4 Détection de collision et simulation

Une fois les angles et normes à tester décidés, il faut simuler le coup. La simulation abandonne les tables virtuelles et se concentre sur la table initiale. À cette étape, on cherche à simuler les collisions jusqu'au point où plus aucune balle ne bouge.

Gérer les collisions avec les coussins et les poches comme n'importe quelle autre collision nous permet de simplifier notre modèle. Par exemple, pour chaque *paire de balles qui ont un vecteur vitesse différent*, on vérifie si elles entrent en collision. Si elles ont le même vecteur vitesse, elles ne peuvent pas entrer en collision. Aussi, pour chaque paire de balle et de poche ainsi que chaque paire de balle et coussin, on doit vérifier s'il y a une collision *lorsque la vitesse de la balle est non nulle.*

La détection de collision va de pair avec la recherche du temps auquel la collision survient. Dans le code, les deux se font au même moment.

4.1 Table au temps t

La première approche de simulation se base sur la prémisse suivante : À partir de la configuration d'une table au temps t_i il est possible de déterminer la configuration de cette table à un temps t_{i+y} où y est un court délai. y doit être choisi afin de ne pas « manquer » une collision. Par exemple, deux balles très proches qui se déplacent rapidement en direction opposée pourraient se croiser sans déclencher de collision si le y est mal choisi.

Pour qu'il y ait une collision avec cette méthode, une balle devrait entrer seulement légèrement en intersection avec une autre balle. y peut être soit constant, déterminé en fonction de la vitesse maximale possible ou calculé aussi fréquemment que l'on veut pour tenir compte de la vitesse de la balle la plus rapide. Un petit y permet d'avoir plus rarement plus d'une collision « en même temps » ainsi qu'obtenir des positions de collision qui se chevauchent le moins possible.

On avance à coup de y à partir du temps 0. On peut

trouver facilement si 2 balles se chevauchent. Il suffit de vérifier si leur centre sont séparés par une distance inférieure au diamètre d'une balle. Si c'est le cas, elles sont en collision.

Pour vérifier si une balle entre dans une poche, on recourt à une méthode semblable. On sait que le centre de masse d'une balle est le même que son centre géométrique. Vu qu'une poche est circulaire, on vérifie si le centre de la poche et le centre de la balle sont séparés par une distance inférieure au rayon d'une poche.

Pour les coussins, on calcule l'intersection de la trajectoire de la balle avec le coussin avancé du rayon de la balle vers le centre de la table. S'il y a intersection, il y a une collision à l'endroit de l'intersection. Avec cette méthode, une balle peut toutefois se *glisser* entre deux coussins avancés. C'est pourquoi on place une balle virtuelle de rayon 0 à l'intersection entre ces coussins, ce qui règle le problème.

Voici donc un diagramme de flot qui montre les étapes générales d'une telle méthode :

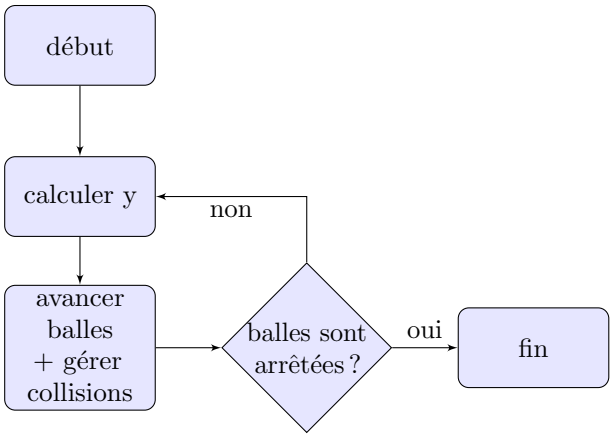

FIGURE 13 – Concept général de la méthode de la table au temps t

On trouve que la complexité de cette méthode est en fonction du nombre de balles (B), du nombre de poches (P), du nombre de coussins (C) et de la durée du coup (S). En supposant que la détection s'effectue en temps constant pour tous les types de collisions :

$$\Theta\left(\frac{S}{y}\left(\frac{B*(B-1)}{2}+BP+BC\right)\right)$$

FIGURE 14 – Complexité algorithmique de la méthode de la table au temps t considérant un coup d'une durée S

4.2 Prédiction de collision avec la position en fonction du temps

En prédisant les collisions, on peut les traiter dans l'ordre où elles surviennent et éviter de générer des étapes sans collisions. Pour vérifier si deux balles vont entrer en collision, il faut tenir compte de leur position dans le temps. Lorsque deux balles sont à une distance égale au double du rayon d'une balle, alors il y a collision. La distance entre deux balles en fonction du temps est donnée par cette fonction :

$$d = \sqrt{\left((x_{i1}-x_{i2})+(v_{xi1}-v_{xi2})t+\frac{(a_{x1}-a_{x2})t^2}{2}\right)^2 + \left((y_{i1}-y_{i2})+(v_{yi1}-v_{yi2})t+\frac{(a_{y1}-a_{y2})t^2}{2}\right)^2}$$

FIGURE 15 – Distance entre deux balles en fonction du temps

Si on transforme cette fonction en équation quartique pour $d =$ *le diamètre d'une balle* et que l'on recherche la plus petite racine réelle positive du polynôme, on trouvera le premier temps de collision entre les deux balles. Cependant, on rappelle que la fonction de position ne considère pas l'arrêt complet de la balle. Il faut donc vérifier que cette collision est valide en fonction des temps d'arrêt des deux balles. C'est pourquoi il est parfois nécessaire de déplacer une des balles à son point d'arrêt, de lui donner une vitesse nulle et de refaire le calcul. Si aucune racine réelle positive valide n'existe, alors les balles n'entrent pas en collision.

Pour évaluer le temps de collision entre une balle et une poche, il suffit de faire le même procédé. Cette fois par contre, la valeur d sera équivalente au rayon d'une balle additionnée du rayon d'une poche. Une poche étant immobile, les valeurs de vitesse et d'accélération seront des zéros.

Enfin, pour déterminer le temps de collision entre une balle et un coussin, il suffit d'imaginer des lignes de collisions qui se situent aux abords des réels coussins à une distance d'un rayon de balle. Ces lignes de collisions sont représentées en magenta dans la figure suivante.

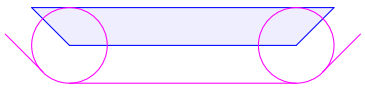

FIGURE 16 – Représentation graphique d'un coussin (en bleu) et de son modèle de collision (en magenta)

Pour trouver le temps de collision avec les coins du coussin (les cercles dans la figure), on procède de la même façon que les poches. En ce qui concerne les segments de droite, on procède en plusieurs étapes. Tout d'abord, il faut créer un segment entre la position actuelle de la balle et sa position à son temps d'arrêt. Ce segment est la trajectoire de la balle. Si la trajectoire ne croise aucun segment de collision du coussin, alors il n'y a pas de collision. Sinon, le temps de collision est le temps auquel la balle est à l'intersection du segment de collision et de la trajectoire de la balle. Le temps t auquel une balle est à une position est donné par la plus grande des racines des deux polynômes suivants :

$$0 = \frac{a_x}{2}x^2 + v_x x + p_{balle_x} - p_x$$
$$0 = \frac{a_y}{2}y^2 + v_y y + p_{balle_y} - p_y$$

FIGURE 17 – Quadratiques représentant la distance x et y entre la balle et une position

Comme ce sont des polynômes du second degré, il est aisé de calculer leurs racines. Si une des racines n'est pas réelle, alors la balle ne passera jamais par cette position. Dans le cas contraire le temps de passage de la balle est donné par la plus grande valeur. Encore une fois, il faut s'assurer que ce temps est inférieur au temps d'arrêt de la balle.

4.3 Solution appliquée

À la lumière des conclusions tirées de l'analyse des deux solutions que nous avons évaluées, nous avons décidé de choisir la seconde. Elle est beaucoup plus rapide que la première et plus précise de surcroît.

5 Définition du meilleur coup

Depuis le début, on veut trouver *le meilleur coup*. Au fait, qu'est-ce que ça veut dire, *le meilleur coup* ?

Par exemple, l'angle 27.924 pourrait faire entrer trois balles du joueur, alors qu'avec un millième de degré plus haut, le coup n'empoche qu'une seule balle et un millième plus bas, rien du tout.

5.1 Coup avec le plus faible risque d'échec

Plusieurs utilisateurs voudront utiliser notre programme pour obtenir le coup *le plus facile à réussir*. Plus formellement, c'est le coup (C) où on peut se permettre la plus grande marge (m) d'erreur, tant par rapport à l'angle (a) que la norme du vecteur vitesse (n). Entre 2 solutions ayant la même marge d'erreur, on départagera avec la somme pondérée des valeurs objectif des coups voisins :

$$\sum_{v_a=-m+a}^{m+a}\left(\sum_{v_n=-m+n}^{m+n} o(C_{v_a v_n} * k^{-|v_a|-|v_n|})\right)$$

FIGURE 18 – Calcul des moyennes pondérées des valeurs objectifs des coups voisins

Plus k est grand, moins l'on donne d'importance aux coups voisins et de moins en moins selon leur éloignement. Une valeur de 1 donne aux voisins autant d'importance. Nous avons mis la valeur k égale à 2 dans le programme.

Afin d'obtenir facilement et rapidement les *voisins* des coups, nous stockons les angles et la norme de vitesse de départ comme des entiers. Dans le cas des angles, par exemple, une valeur entière correspond à des millièmes de radians. On évite l'utilisation d'un delta, parce qu'on sait que l'arithmétique de nombres à virgules stockés selon la norme $IEEE$ 754 n'est pas exacte. Pendant la simulation, nous manipulons des nombres réels avec un type à double précision.

5.2 Coup chanceux

Les utilisateurs désirant calculer le meilleur coup, sans égard à sa faisabilité, pourront choisir la stratégie du coup chanceux. Très rapide à calculer, elle retourne le coup qui maximise la fonction objectif. Si la fonction objectif est à son maximum pour un coup, elle peut arrêter sa recherche.

6 Développement du solveur

La réalisation du programme s'est faite de façon incrémentale. Nous avons d'abord cherché à trouver :

1. L'angle pour empocher une seule balle avec la blanche, si cet angle existe.
2. Le coup qui empochera le plus de balles d'une table ayant au moins 2 balles.
3. Le coup qui optimisera la fonction objectif o définie à la fin de la section 1.

Tout au long du développement, nous avons écrit des tests unitaires pour s'assurer que le comportement désiré était atteint. Comme le code était immense, un petit changement accidentel dans une formule physique aurait pu retarder énormément le développement. Nos tests unitaires et de régression ont veillé à ce que ça n'arrive pas.

7 Résultats

Nous avons soumis notre programme à des tables de 2 à 15 balles. Voici ce que nous avons constaté comme temps d'exécution :

Temps d'exécution en fonction du nombre de balles

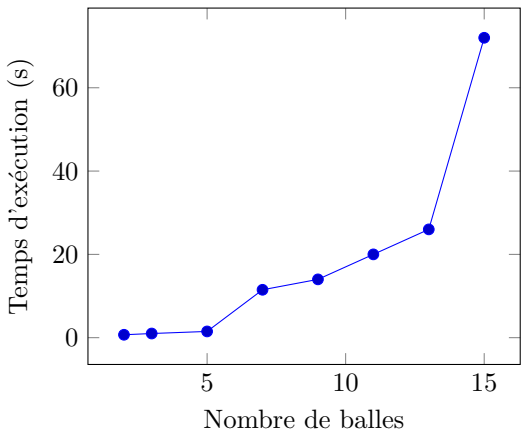

FIGURE 19

On constate que plus il y a de balles, plus le temps d'exécution est long. C'est logique : la simulation est plus complexe. Les algorithmes et le filtrage utilisés font aussi que plus le nombre de balles du joueur est élevé comparativement au nombre de balles adverses, plus les intervalles à parcourir seront nombreux. Par conséquent, ce sera plus long.

8 Discussion

Tout au long du développement, notre principal souci fut la précision. Nos implémentations de la prédiction de collision entre une balle en mouvement et un autre objet donnent des temps très précis : En moyenne $\pm 5\text{E}{-11}$. Un delta presque aussi petit a suffit pour s'assurer, par exemple, que la vitesse d'une balle était bien nulle, et donc que la balle était arrêtée. Une structure de données comme *BigDecimal* n'aurait pas réglé ce problème, tous les nombres réels ayant une représentation décimale non exacte auraient eu une imprécision.

Le réalisme fut notre seconde préoccupation. Les sorties du programme sont des solutions réalistes. Cependant, certains facteurs pourraient faire en sorte que des coups ne puissent pas être reproduits sur une vraie table, comme l'usure de la table et la difficulté de réaliser exactement la frappe voulue sur la blanche en alignant parfaitement la queue avec le centre de la balle.

9 Conclusion

Dans le cadre de ce travail, nous avons développé un solveur paramétrable appliqué à un problème bien précis. Pour cela nous avons conçu des algorithmes de filtrage, un heuristique et des stratégies de résolution. Nous avons étudié des approches différentes pour la recherche de solution ainsi que la détection de collision. Le résultat est satisfaisant ; les méthodes que nous avons choisies nous ont permis d'avoir des temps de calcul courts et les filtrages réduisent considérablement le domaine du vecteur vitesse.

Il aurait été excitant de valider les résultats du programme sur une vraie table avec une machine frappant précisément la blanche pour lui donner le vecteur vitesse désiré.

Remerciements

Nous voulons sincèrement remercier Raphaël Blais, étudiant en génie physique à l'Université Laval, qui nous a aidé à élaborer un modèle réaliste sur le plan des collisions entre balles.

Références

[1] David Alciatore, PhD, *Physics of Pool and Billiards*, http ://billiards.colostate.edu/threads/physics.html

[2] David Alciatore, PhD, *90 degrees and 30 degrees Rule Follow-up – Part III : inelasticity and friction*, http ://billiards.colostate.edu/bd_articles/2005/april05.pdf