Table de Billard Connectée

Rapport intermédiaire de projet

November 13, 2022

Par :
Thibault Meynier,
Florian Derlique

Tuteurs :
Xavier Redon,
Thomas Vantroys,
Alexandre Boé

Contents

1 Mathématiques et physique liée au billard 5
1.1 Généralités sur le mouvement d'une boule en deux dimensions 5
1.2 Estimation de l'accélération d'une balle 5
1.3 Calcul de collision entre deux balles 5
1.4 Réponse d'une collision entre deux boules 6

2 Méthodologie de développement 8
2.1 Modèle MVC 8

3 Développement de la simulation 9
3.1 Les modèles 9
3.2 Les vues 10
3.3 Les contrôleurs 11
3.3.1 Les classes de listeners 11
3.3.2 La classe Main 11

4 Résultats 13
4.1 Premier jet 13
4.2 Deuxième Jet 13
4.3 Troisième Jet 14

5 Poursuite des travaux 15
5.1 Simulation 15
5.1.1 Correction du bug de collision 15
5.1.2 Mise en place des règles 16
5.1.3 Amélioration de l'interface graphique et de la simulation 16
5.2 Traitement d'image 16
5.2.1 Choix des technologies 16
5.2.2 Méthodologie d'une solution 16
5.3 Application mobile 16
5.3.1 Choix des technologies 16
5.4 Autre solution 17

List of Figures

1 Principe de détection de collision entre deux boules 6
2 Principe de réponse d'un collision entre deux boules 7
3 Illustration du modèle mvc 8
4 Algorithme de mise à jour de la Table de jeu 9
5 Illustration du fonctionnement de l'affichage 11
6 Algorithme du thread d'actualisation de l'interface graphique 12
7 Algorithme de la méthode main 12
8 Rendu du premier Jet 13
9 Bug de collision 13
10 Rendu du deuxième Jet 14
11 Rendu du deuxième Jet après coup de casse 14
12 algorithme de calcul des collisions, version corrigée 15
13 Interface du troisième Jet après coup de casse 15
14 Diagramme de Gantt prévisionnel après mise à jour 18

Introduction

En 2015 des étudiants de la filière IMA de Polytech ont réalisé une table connectée comportant un écran tactile pour leur projet de fin d'études. Cette table est aujourd'hui inexploitée.

L'objectif de ce projet est donc de créer un jeu de billard en réalité augmentée en profitant des caractéristiques de la table. En plus d'offrir une expérience unique grâce à une caméra au dessus de la table qui permettra de jouer avec une queue adaptée similaire à celle d'un véritable billard. Il sera également possible de suivre les parties grâce à une application mobile sur un appareil connecté en Bluetooth à la table.

Ce projet sera le deuxième projet réalisé sur cette table connectée, après la réalisation en 2018 d'une table de bar connectée par des étudiants en IMA de Polytech Lille. La table a été créée en 2015 par les étudiants de Polytech Lille.

Ce rapport décrit l'avancement du projet à ce jour : ce qui marche, ce qui ne marche pas, ainsi ce qui a été fait pour corriger les problèmes le cas échéant. Ce rapport permet de faire le lien entre les méthodes et outils définies dans le cahier des spécifications et ce qui a été réellement mis en oeuvre. La conclusion portera sur le travail restant et sur un nouveau diagramme de Gantt basé sur l'avancement du projet.

On rappelle les fonctionnalités attendues :

- Possibilité de jouer une partie de billard dans un premier temps sur l'ordinateur;
- Intégration de cette simulation de jeu sur l'écran tactile;
- Pouvoir jouer au jeu avec une queue de billard modélisée pour cette application *(fonctionnalité attendue dans le cadre du projet)*;
- Ajout d'une application, pour permettre aux spectateurs de regarder la partie de jeu sur leurs portables en communication Bluetooth *(fonctionnalité attendue dans le cadre du projet)*;
- Aller au delà du jeu de billard en intégrant d'autres jeux *(optionnel)*;
- Enregistrement des parties pour pouvoir les revoir en replay. *(optionnel)*.

Matériel à disposition :

- Une table avec un écran tactile intégré créée par les étudiants IMA en 2015. La dalle tactile est de la marque Iiyama, gamme ProLite (TF3237MSC), avec une diagonale de 32". Il est possible d'utiliser cette dalle en multitouch;
- Un ordinateur Dell Précision T1700 doté d'un processeur Intel Core I7 de 4ème génération, 32 Go de RAM DDR3 et une carte graphique Intel Graphics. Il fonctionne actuellement sous Debian avec l'environnement graphique Gnome;
- Une caméra Logitech C920 pour la détection de la queue de billard sur la dalle.

1 Mathématiques et physique liée au billard

1.1 Généralités sur le mouvement d'une boule en deux dimensions

La boule sera soumise à une vitesse dans deux dimensions (axes x et y), où le vecteur vitesse se calcule de la sorte :

$$v = \sqrt{v_x^2 + v_y{}^2} \quad (1)$$

La nouvelle position de la boule à un intervalle de temps donné sera :

$$x_{t+1} = x_t + v_x \cdot \Delta t \quad (2)$$

$$y_{t+1} = y_t + v_y \cdot \Delta t \quad (3)$$

La nouvelle vitesse de la boule à un intervalle de temps donné doit être calculé selon son vecteur vitesse et l'accélération, alors :

1. on calcule son vecteur vitesse défini en (1)
2. on normalise les deux vitesses de l'axe des x et y (cela permet de garder en mémoire le sens ainsi que l'angle du vecteur vitesse):
$$intensity_X = \frac{v_x}{v} \quad (4)$$
$$intensity_Y = \frac{v_y}{v} \quad (5)$$
3. On applique l'accélération au vecteur vitesse calculé en (1) où :
$$v_{t+1} = v_t + a \cdot \Delta t \quad (6)$$
4. Finalement, on recalcule les vitesses en deux dimension en multipliant (6) par (4) et (6) par (5).

1.2 Estimation de l'accélération d'une balle

Considérons que l'accélération est soumise au vecteur vitesse de la balle calculée en (6), et que la balle est frappée uniquement en son centre (ne permet pas de prendre en compte les vitesses angulaires), alors la balle est soumise uniquement à une friction entre elle et la table. Cette friction est toujours de sens opposé à la vitesse. D'après la seconde loi de Newton :

$$ma = \sum F = -mg\mu_c \quad (7)$$

où :

- g est l'accélération de la pesanteur à la surface de la Terre égale à $9.80665 m.s^{-2}$;
- μ_c est le coefficient de frottement de la balle sur la table, comme celui-ci est difficile à connaître, la valeur de 0,1 a été choisie, partant du principe que le frottement entre la table et la boule était élevé.

On peut alors estimer une accélération égale à $-0.98 m.s^{-2}$ pour un coefficient de frottement de 0.1.

1.3 Calcul de collision entre deux balles

Il faut calculer si deux balles entrent en collision, il existe une façon simple. Soient deux objets circulaires :

- C_1 de centre $x1, y1$ et de rayon $r1$.
- C_2 de centre $x2, y2$ et de rayon $r2$.

Si on imagine une ligne entre les deux centres des cercles, la distance D qui sépare les deux centres est :

- $D < |r1 - r2|$si les deux boules sont en collision
- $D > |r1 - r2|$si les deux boules ne sont pas en collision

Pour déterminer si deux boules se touchent, il suffit de vérifier que :

$$(x2 - x1)^2 + (y2 - y1)^2 \leq (r1 + r2)^2 \tag{8}$$

Il est aussi possible de calculer l'angle α à partir de formules basique de trigonométrie :

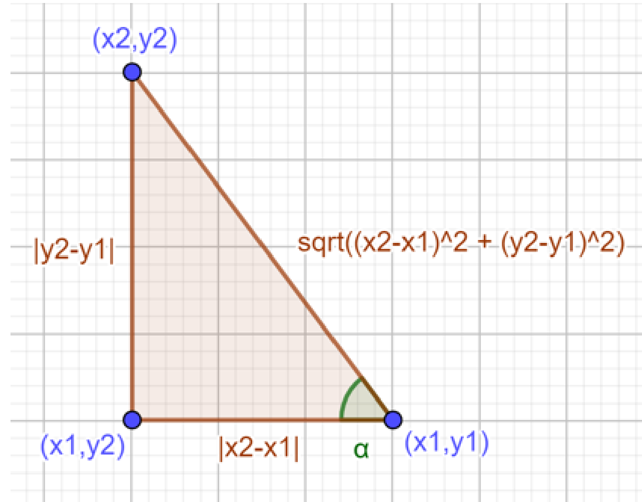

Figure 1: Principe de détection de collision entre deux boules

- $cos(\alpha) = \frac{|x2-x1|}{r1+r2}$
- $sin(\alpha) = \frac{|y2-y1|}{r1+r2}$
- $tan(\alpha) = \frac{|y2-y1|}{|y2-y1|}$

Et éventuellement le point de collision $x3, y3)$ à partir du centre de la boule C_1 :

- $x3 = x3 - r1 \cdot cos(\alpha)$
- $y3 = y3 + r1 \cdot sin(\alpha)$

1.4 Réponse d'une collision entre deux boules

On considère le cas où seule une boule possède une vitesse selon x et y. Soient deux objets circulaires :

- C_1 de centre $x1, y1$, de rayon $r1$ de vitesse v_x, v_y.
- C_2 de centre $x2, y2$ et de rayon $r2$.

Dans un premier temps, on calcule l'angle entre l'axe de collision et l'axe principal :

$$cos(\alpha) = \frac{x_1 - x_2}{\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}}, sin(\alpha) = \frac{y_1 - y_2}{\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}} \tag{9}$$

On calcule la valeur scalaire de la nouvelle vitesse dans l'axe de collision selon v_x, v_y et (9).

$$vecteurVitesse = cos(\alpha) \cdot v_x + sin(\alpha) \cdot v_y \tag{10}$$

Ainsi, les vitesses initiales seront décomposées en deux composantes:

- la vitesse transmise à la boule C_2 :

$$vC2_x = cos(\alpha) \cdot vecteurVitessse, vC2_y = sin(\alpha) \cdot vecteurVitesse \tag{11}$$

- ainsi que la vitesse gardée par la boule C_1 :

$$vC1_x = vecteurVitesse - vC2_x, vC1_y = vecteurVitesse - vC2_y \tag{12}$$

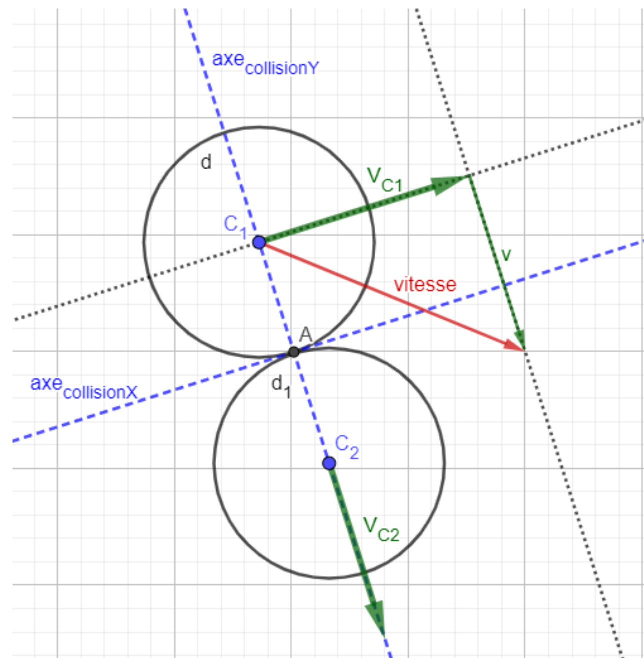

Figure 2: Principe de réponse d'un collision entre deux boules

On fait de même dans le cas où la boule C_2 possède une vitesse, on additionne toutes les composantes entre elles pour obtenir les nouvelles vitesses des deux boules.

2 Méthodologie de développement

2.1 Modèle MVC

Lors de la conception d'un logiciel graphique, il est facile de se perdre dans l'organisation du code. L'un des moyens pour s'organiser proprement est de bien découper chaque partie du projet pour séparer l'affichage de la modélisation par exemple. Une méthode qui va dans ce sens est le modèle MVC. Le concept est simple, Il y a trois types de composants : Le modèle, la vue et le contrôleur. Pour faire simple, le modèle représente un objet, une classe, les données de l'application.

Ce composant n'a pas d'existence graphique, n'interagit pas directement avec l'utilisateur. Par exemple, dans le projet, on a un modèle pour représenter les balles, un autre pour la table, etc.

Ensuite, il y a la vue qui s'occupe uniquement de l'affichage d'un modèle. Toujours dans le projet, on peut imaginer une vue pour chaque balle ainsi qu'une vue pour la table. Elle communique avec le contrôleur pour lui indiquer au travers des listeners comment l'utilisateur interagit avec l'interface.

Enfin, le contrôleur est le composant qui s'occupe de mettre ces deux entités en relation, il est comme le cerveau de l'application, c'est lui qui contient toute la logique de celle-ci. En effet, c'est lui qui récupère les interactions avec l'utilisateur au travers de la vue grâce aux listeners et met à jour le modèle puis la vue en conséquences.

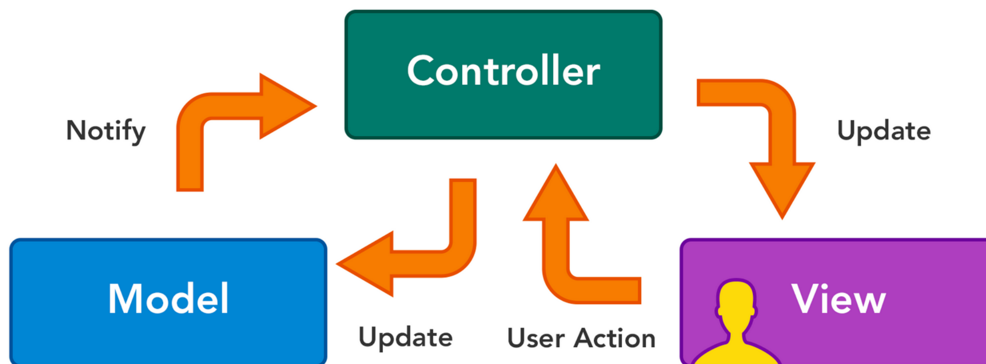

Figure 3: Illustration du modèle mvc

3 Développement de la simulation

3.1 Les modèles

Les différentes classes qui composent l'application sont :
La classe "**BallTable**" qui représente la table de jeu La table de jeu est composée de :

- 2 joueurs (instanciés dans la classe main !);
- d'une liste de boules;
- d'une liste de trous;
- d'attributs utiles pour récupérer le temps entre deux mises à jour, par exemple.

En plus des setters et des getters, cette classe comporte quelques méthodes intéressantes :

- Une méthode indicant si au moins une boule est toujours en déplacement;
- Une méthode qui effectue la mise à jour de la table pour un intervalle de temps entre 2 appels de cette méthode, l'algorithme de cette méthode est expliquée dans le diagramme ci-dessous.

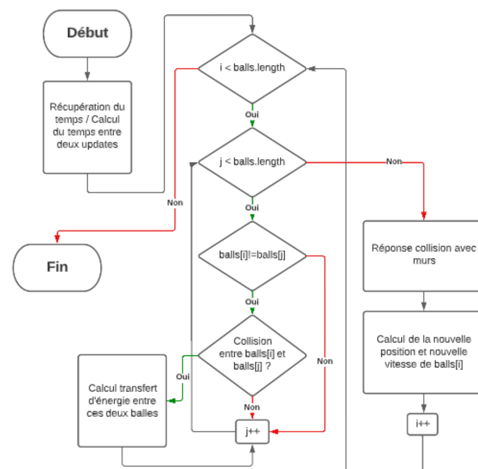

Figure 4: Algorithme de mise à jour de la Table de jeu

La classe "**Ball**" modélisant une boule de billard, elle prend comme attribut :

- Une position x, y et une vitesse v_x, v_y représentés par une classe Point.
- Un type de boule (blanche, noire, pleine ou rayée).
- Un chiffre (de 0 à 15).

Elle comporte méthodes utilitaires ("getters" et "setters[1], méthode pour savoir si la boule est toujours en mouvement, etc). Elle comporte aussi des méthodes complexe issues des algorithmes de mathématique et physique liée au billard (calcul de collision avec un mur, calcul de collision avec une boules, réponse à une collision entre boules). Finalement, la méthode de mise à jour de la position de la boule.

La classe "**Player**" modélise un joueur, elle possède comme attributs :

- Un ID (String).
- Un type de boule (celui que le joueur doit toucher).

[1] https://www.geeksforgeeks.org/getter-and-setter-in-java/

- Des drapeaux de contrôle (faute, gagne, à lui de jouer, etc).

Elle permet de dire si le joueur fait une faute ou non, s’il a gagné la partie, etc. Elle comporte différentes méthode utilitaires comme chaque classe.

L’interface "**Hole**" et ses implémentations gèrent le comportement des trous. Un billard possède plusieurs type de trous : des trous ronds sur les côtés et en Oblong[2] dans les coins. Chaque type de trou a sa forme mais possède une méthode de calcul imposée par l’implémentation de l’interface "Hole" qui indique s’il y a collision avec une boule ou non.

L’interface "**Rules**" modélise les règles du jeu. Chaque set de règles est différent, l’utilisation d’interface permettra de créer plusieurs jeux de billards comme le billard français par exemple. Voici la liste des règles applicables pour la simulation :

- Si un joueur commet une faute, le joueur adverse peut replacer la balle blanche n’importe où sur la table;
- Lorsqu’un joueur tire, il commet une faute si,
  - Il tire une autre boule que la blanche.
  - La boule blanche ne touche aucune autre boule ni bande avant de s’arrêter.
  - La boule blanche touche une boule adverse ou la boule noire en premier.
  - La boule blanche touche une boule alliée puis ne touche aucune bande. Le joueur ne commet pas de faute s’il empoche sa boule.
  - La boule blanche est empochée.
- Si un joueur empoche une de ses boules, il peut jouer un coup supplémentaire. Empocher deux boules d’un coup ne donne pas droit à deux coups supplémentaires. Mais tant que le joueur empoche ses boules, il peut continuer à jouer.
- Pour gagner, un joueur doit empocher toutes ses boules puis empocher la boule noire. Le premier joueur arrivant à ce résultat gagne;
- Si un joueur empoche la boule noire avant d’avoir empoché toutes ses boules, il a immédiatement perdu.

Ces classes récupèrent les informations nécessaires à travers les différentes autres classes afin de déterminer si le joueur actuel a commis une faute pendant son tour. Elles doivent savoir si la boule blanche a eu une collision pendant le coup du joueur. En cas de faute, le prochain joueur a la possibilité de placer la boule blanche pendant son prochain tour.

3.2 Les vues

Chaque élément qui doit être affiché à l’écran a sa propre vue. Par exemple, la table de jeu a sa vue, les balles ont une vue, les trous ont une vue, etc.

Avec Java, il était difficile de respecter cette partie du modèle MVC car l’architecture Swing ne s’y prête pas trop. Pour faire un rendu, swing met à disposition un objet qui permet de dessiner sur l’écran mais qui n’est utilisable que dans la classe représentant l’écran. Il est donc difficile de séparer l’affichage en plusieurs fichiers

La solution qui a été retenue a été de tirer parti des fonctionnalités d’interfaces offertes par Java. Une interface View a donc été créée. Chaque classe réalisant cette interface doit implémenter une méthode render qui utilise le fameux objet de dessin pour dessiner à l’écran. De cette façon, on peut segmenter l’affichage en plusieurs fichiers et on a plus qu’à créer un tableau avec toutes les vues et à appeler toutes leurs méthodes render pour rafraîchir l’affichage. Pour résumer, lors de l’exécution, la classe représentant l’écran "prête" donc son objet de dessin à chaque vue pour

[2] https://fr.wikipedia.org/wiki/Oblong

qu'elle dessine un élément à la fois.

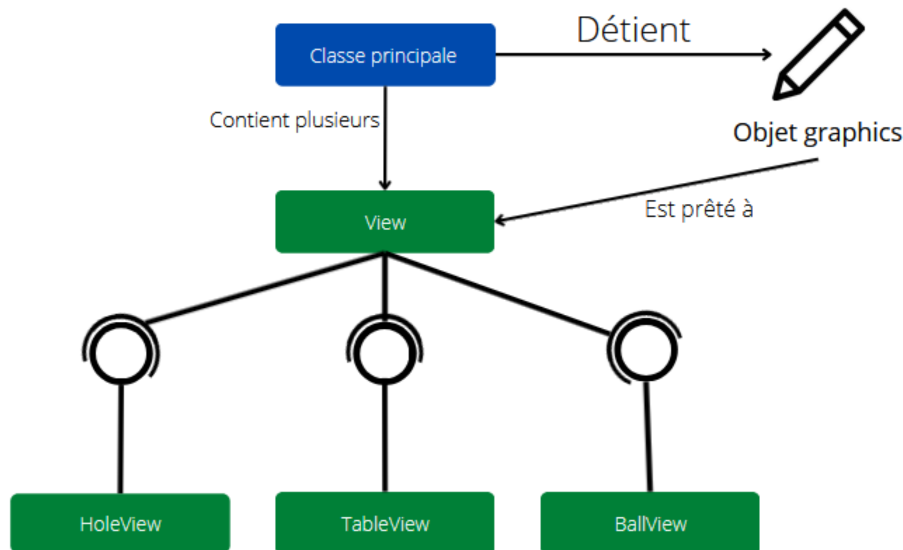

Figure 5: Illustration du fonctionnement de l'affichage

3.3 Les contrôleurs

Le contrôleur principal utilisé dans l'application est la classe Main qui boucle jusqu'à ce que l'utilisateur quitte l'application, elle gère le déroulement du jeu et met à jour les vues en utilisant les données récupérées par les listeners.

3.3.1 Les classes de listeners

Les listeners sont des interfaces du package java.awt permettant de capturer les actions d'un périphérique (souris, clavier, etc). Dans ce cas, ce sont les listeners de souris qui sont utilisés. Il y a 3 interfaces de listeners de souris :

- L'interface "MouseListener" permettant de capturer le moment où l'utilisateur clique, appuie, ou lâche un bouton de la souris.
- L'interface "MouseMotionListener" permettant de capturer les mouvements de la souris, lorsque un bouton est appuyé ou non.
- L'interface "MouseWheelListener" permettant de capturer les mouvements de la molette de la souris.

Plutôt que de réaliser des implémentations des interfaces MouseListener et MouseMotionListener. Il a été décidé de simplement réaliser une classe qui hérite de la classe abstraite MouseAdapter qui comporte des méthodes vides pour tous les évènements de ces trois interfaces. On peut donc simplement réécrire celles dont on a besoin. Pour le projet, les classe suivantes ont été créées :

- la classe "AimListener" qui s'occupe de détecter lorsque le joueur essaye de tirer. C'est également lui qui s'occupe de mettre à jour une vue qui trace une ligne entre le lieu du clic et le pointeur pour aider l'utilisateur à viser.
- la classe "FaultListener" qui permet à l'utilisateur de placer la boule blanche en cas de faute de l'adversaire.

3.3.2 La classe Main

La classe "main" est la colonne vertébrale du projet la méthode principale de cette classe réalise dans l'ordre :

1. l'instanciation des joueurs;
2. l'instanciation de de la table de jeu;
3. l'instanciation de la classe d'affichage (Renderer);
4. l'instanciation des listeners ainsi que leur liaison avec les vues;
5. la création du thread d'affichage. Auparavant, la mise à jour de l'interface graphique était réalisée lorsqu'une des boules avait changé de position pendant la mise à jour de la table de billard. Cette méthode était certes efficace, mais inutilement coûteuse en termes de ressources. Il a donc été décidé de séparer l'affichage et la mise à jour de la table de jeu. De cette façon, on peut contrôler le nombre d'affichages par seconde et énormément réduire l'utilisation de ressources;

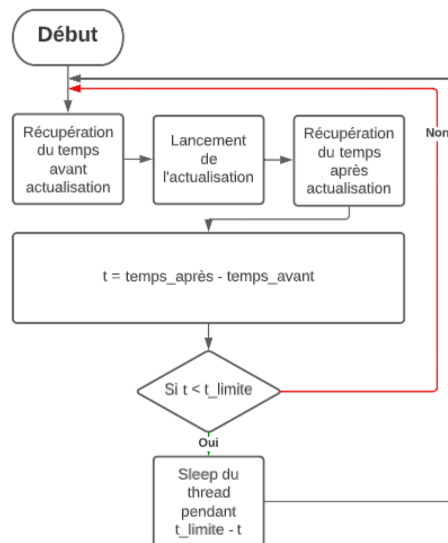

Figure 6: Algorithme du thread d'actualisation de l'interface graphique

6. lancer la boucle principale qui gère le jeu (mise à jour de la table, respect des règles, etc).

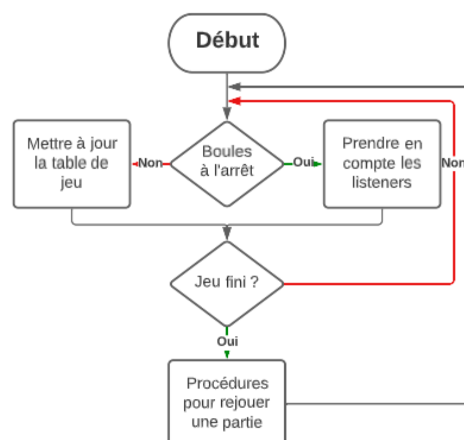

Figure 7: Algorithme de la méthode main

4 Résultats

4.1 Premier jet

Le premier jet, utilisait l'ancienne méthode d'affichage, le freinage de la table n'était pas non plus pris en compte, les boules ne décéléraient donc jamais. Les listeners n'étaient pas encore implémentés. Et les trous n'étaient pas non plus fonctionnels.

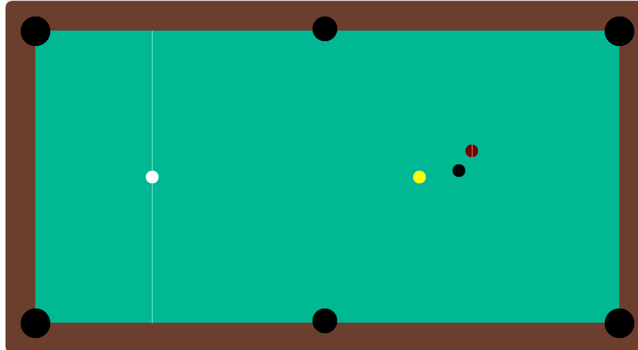

Figure 8: Rendu du premier Jet

L'exécution se fait sur un ordinateur avec Windows, les collisions marchent correctement, on remarque seulement quelques fois un problème de boules qui se rentrent dedans :

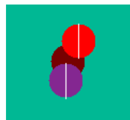

Figure 9: Bug de collision

Ce problème peut-être du à :

- Un problème sur le calcul de collision
- Un problème sur le temps d'exécution, un temps trop élevée et la boule peut rentrer dans une autre à la prochaine mise à jour des positions

Dès l'exécution sur un PC Debian de l'école, la simulation s'ouvre 1 fois sur 3, le cas échéant, la simulation est très différente de l'exécution sur Windows. La barre des tâches en haut de l'écran n'est pas recouverte par l'application en plein écran, de plus, les éléments graphiques sont tous décalés comme si l'application ne pouvait afficher que sur la partie en dehors de la barre des tâches mais qu'elle prenait la largeur et la hauteur totale de l'écran pour les coordonnées.

Ce bug a vite été résolu en utilisant une différente API pour mettre l'application en plein écran.

4.2 Deuxième Jet

Dans ce deuxième jet, le problème de plein écran sur les machine Debian est corrigé. Les trous sont fonctionnels. On implémente les listeners pour pouvoir tirer dans la boule blanche, le but est de savoir si la simulation répond bien lors du coup de casse (le premier coup dans une partie de billard est celui où il y a le plus de collision). Sur ce jet, le problème des collisions de boules n'est pas encore entièrement corrigé, pour voir si le problème persiste dans ce cas. On remarque que le bug de collision est très présent. De plus, il est impossible de jouer le prochain coup quand ce bug est présent. En effet, même si les boules ne bougent pas, la simulation considère que leur vitesse n'est pas nulle et elle empêche donc que le prochain coup soit joué. La prochaine étape est de corriger ces problèmes.

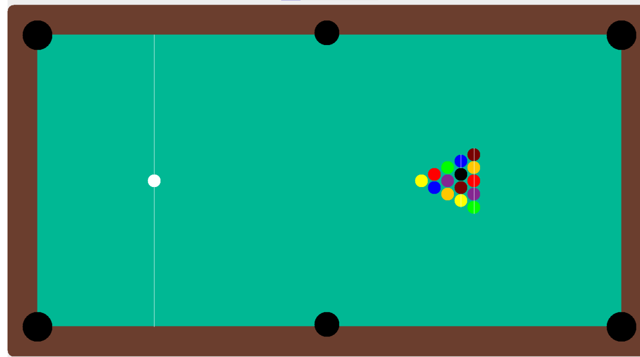

Figure 10: Rendu du deuxième Jet

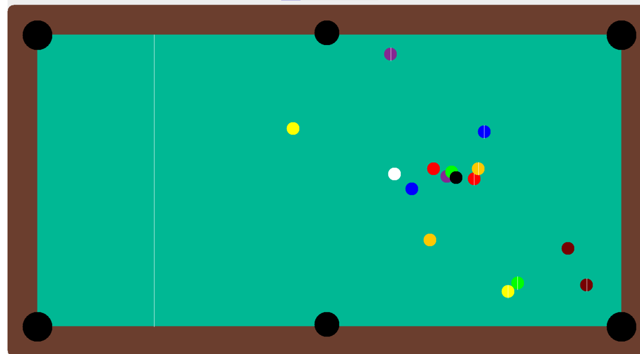

Figure 11: Rendu du deuxième Jet après coup de casse

4.3 Troisième Jet

On remarque que certaines collisions se calculent deux fois dans la même boucle de calcul. L'idéal serait de n'avoir qu'un seul calcul de collision, on donne alors un tableau de 16 booléens à chaque boule pour savoir si la collision entre les boules a déjà été calculé. A chaque fin de boucle, on remet à zéro le tableau de booléen de la boule correspondante.

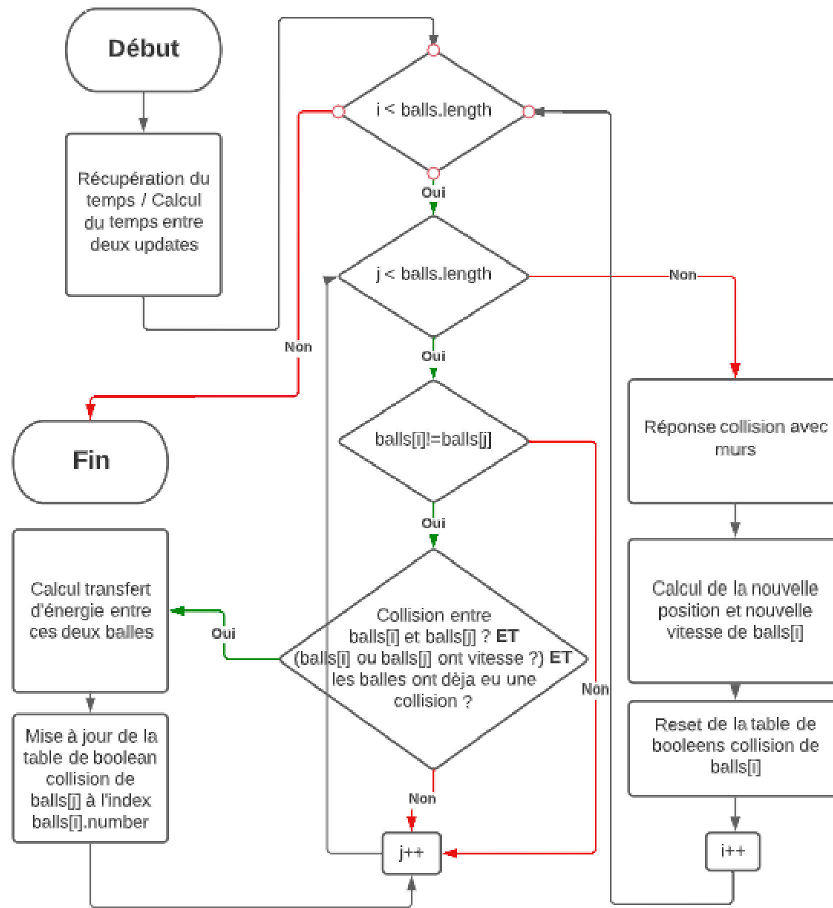

Figure 12: algorithme de calcul des collisions, version corrigée

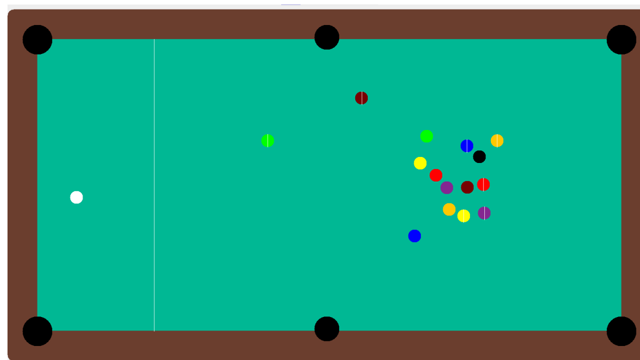

Figure 13: Interface du troisième Jet après coup de casse

On remarque une amélioration que ce bug ne s'est pas produit, cependant il existe toujours, il va donc falloir le supprimer définitivement. Avec la nouvelle méthode de calcul, les boules se détachent plus facilement lorsque le bug se produit. Ce patch corrige également l'impossibilité de jouer le coup quand le bug de collision est présent.

5 Poursuite des travaux

5.1 Simulation

5.1.1 Correction du bug de collision

Le bug n'est pas définitivement corrigé, cependant un nouveau patch peut résoudre ce problème :
Ce problème arrive quand le temps de calcul entre deux mises à jour du jeu est trop grand. La solution est de mettre à jour la position de la boule avant le calcul de collision. Avant le calcul de collision entre deux boules, en calculant

la distance entre ces deux boules, il est facile de savoir si celle-ci se rentrent dedans (voir la méthode de calcul de distance entre deux boules). Le cas échéant, le temps entre deux mises à jour dépasse le temps limite à la boule pour ne pas rentrer dans une autre. Il faut alors calculer ce temps limite et corriger la position de la boule pour qu'il ne soit pas dans une autre boule. On calcule la réponse à la collision et on remet à jour la boule avec le temps restant (soustraction entre le temps entre deux mises à jour et le temps limite).

5.1.2 Mise en place des règles

La correction des bugs ayant été plutôt longue, les règles du jeu ainsi que l'algorithme de fin n'ont pas encore été complètement implémentés. Cette mise à jour devrait arriver peu après la rédaction de ce rapport.

5.1.3 Amélioration de l'interface graphique et de la simulation

Même si les 2 précédents travaux permettent d'avoir une simulation fonctionnelle, il est toujours appréciable d'améliorer la simulation. Il est donc prévu de :

- Optimiser le temps de calcul;
- Diminuer la RAM utilisée;
- Améliorer l'interface graphique et ajouter les bandes en forme de trapèze sur les côtés;
- Éventuellement prendre en compte la rotation horizontale des boules pour plus de réalisme;
- Faire en sorte que l'utilisateur puisse donner un coup de canne n'importe où et pas simplement au centre de la boule blanche.

5.2 Traitement d'image

5.2.1 Choix des technologies

Comme expliqué dans le cahier de spécification, c'est le framework OpenCV qui a été retenu. il permet de faire du traitement d'image, a initialement été développé en C/C++ mais est aussi disponible pour Java.

5.2.2 Méthodologie d'une solution

On doit faire en sorte de capturer une image (sous forme de matrice de valeurs) tous les X ms, faire un pré-rendu sur cette image pour mettre en évidence l'objet à capturer (e.g la pointe de la queue dans le cas du projet). Ensuite, capturer les points maximum et minimum en x et y pour calculer le milieu de la pointe de queue. Si la position capturée entre en collision avec la boule blanche alors on calcule la trajectoire de la queue à l'aide des positions anciennement calculées. Avec le temps et la taille du segment de trajectoire on peut calculer la force émise sur la boule blanche, ainsi que sa trajectoire.
Cette solution est une première approche et est susceptible de changer. Pour la suite du projet, de petits projets seront réalisés pour prendre en main la technologie. Puis cette première solution sera mise en place en observant les résultats, il faut changer de méthode ou continuer sur celle ci.

5.3 Application mobile

5.3.1 Choix des technologies

Comme indiqué dans le cahier des spécifications, c'est la technologie Android Studio qui a été retenue. Celle-ci présente l'avantage d'être écrite en java. La partie communication en Bluetooth ne devrait pas poser de problème.

5.4 Autre solution

Il a également été envisagé, plutôt que redessiner à l'écran toute la table de billard, etc, et de transmettre la position des boules et toutes les informations par Bluetooth; de simplement faire en sorte que la machine sur laquelle se passe la partie diffuse un flux vidéo que tous les appareils connectés puissent capter. On peut aussi imaginer que l'appareil source crée un hotspot wifi pour faciliter la diffusion.

Conclusion

La simulation du billard est quasiment terminée, il reste quelques bugs à corriger et quelques améliorations de "confort" et d'expérience utilisateur à apporter mais dans l'ensemble cette partie est presque terminée.
L'équipe projet a fait face à beaucoup de problèmes et d'imprévus mais a su à chaque fois trouver des solutions pour continuer à avancer.

Il y a eu quelques manquements au diagramme de Gantt qui avait été établi en Septembre, voici une nouvelle version modifiée prenant en compte l'avancement actuel du projet :

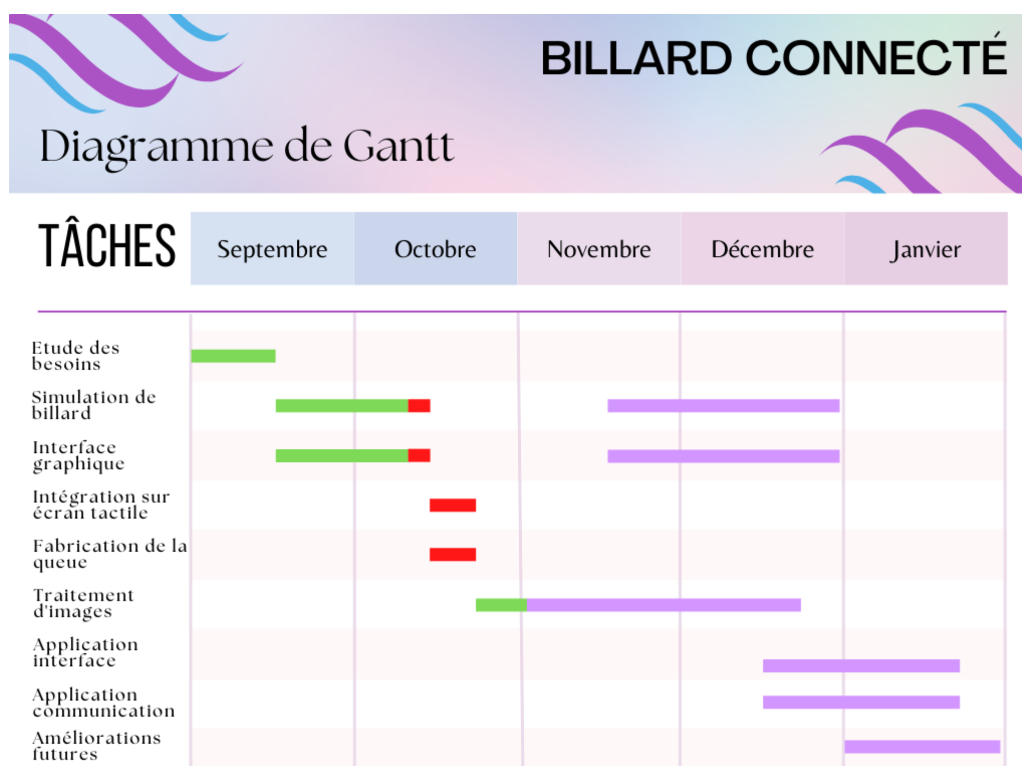

Figure 14: Diagramme de Gantt prévisionnel après mise à jour

Pour la deuxième partie du projet, l'équipe projet va se focaliser sur l'intégration de la solution de traitement d'image pour jouer avec une vraie queue de billard et la diffusion de la partie sur application mobile.