Table de Billard Connectée

Rapport de projet

3 février 2023

Par :

Thibault Meynier,

Florian Derlique,

SE5SC

Tuteurs :

Xavier Redon,

Thomas Vantroys,

Alexandre Boé

Table des matières

Table des figures

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 ou un site web.

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.

Dans un premier temps, la gestion du projet sera décrite avec un rappel du cahier des charges, une répartition des tâches entre les membres de l'équipe, des études sur les opportunités, les parties prenantes, les ressources mises à disposition pour le projet, ainsi que la méthodologie utilisée.

Dans un second temps sera décrite la recherche autour du projet avec l'étude de l'existant, des technologies de développement, les algorithmes mathématiques pour la physique liée au billard et enfin une sous partie sur les règles du billard.

Dans un troisième temps, on détaillera les travaux réalisés par l'équipe lors du temps mis à disposition pour le projet. On verra notamment la partie simulation, modélisation de la canne de billard, le traitement d'image, l'alternative qui a été prise à la partie développement d'application mobile pour observer les parties de billard et enfin une dernière sous-partie sur la restructuration du projet et son optimisation.

Dans un dernier temps, il y aura une partie dans laquelle l'équipe projet développera son point de vue sur le déroulement du projet, les points forts, les améliorations qui pourraient être apportées, et un point de vue davantage axé commercial (si le projet devait être commercialisé).

1 Gestion du projet

Dans cette partie, on abordera les grandes parties de la gestion du projet :

— Un rappel du cahier des charges ;
— Un rappel du planning prévisionnel déterminé en Septembre 2022 et Novembre 2022 et les comparer avec le planning réalisé ;
— Une étude d'opportunité de notre projet ;
— Détermination des partie prenante de notre projet ;
— Détermination des ressources mises en oeuvres pour le projet (budget utilisé, matériel, ...) ;
— Méthode de réalisation des tâches.

1.1 Rappel du cahier des charges

On rappelle le matériel à disposition au début du projet :

— 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, 16 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.

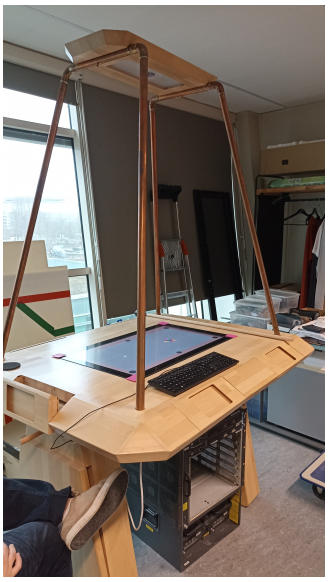

FIGURE 1 – Table en bois issue d'un projet des anciens IMA5

Et les fonctionnalités attendues :

— Possibilité de jouer une partie de billard dans un premier temps sur l'ordinateur ;
— Intégration de cette simulation de l'ordinateur intégré à la table avec 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)*.

1.2 Planning et répartition des tâches

Les principales tâches du projet ont été définies en septembre 2022, à travers la réalisation d'un planning de répartition de tâches (diagramme de Gantt) :

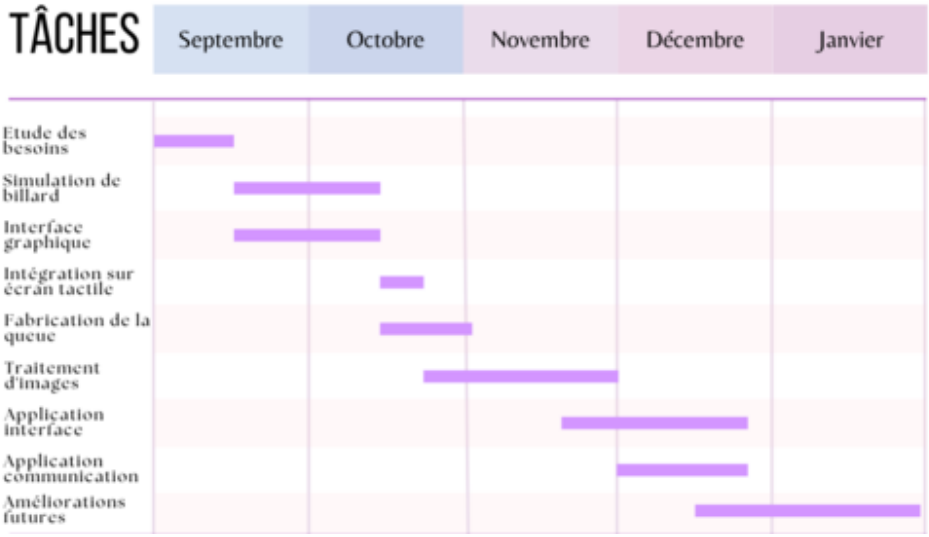

FIGURE 2 – Gantt prévisionnel du 6/9/2022

La répartition du personnel est la suivante :

— Florian : Simulation du billard, Intégration sur écran tactile, Traitement d'image, Application (communication) ;
— Thibault : Interface graphique, Fabrication de la queue de billard, Traitement d'images, Application (interface).

La révision du planning a été effectué après la soutenance de mi-parcourt de projet (en vert : ce qui est fait au 10/11/22, en rouge : ce qui n'est pas fait au 10/11/22, en violet : ce qui reste à faire).

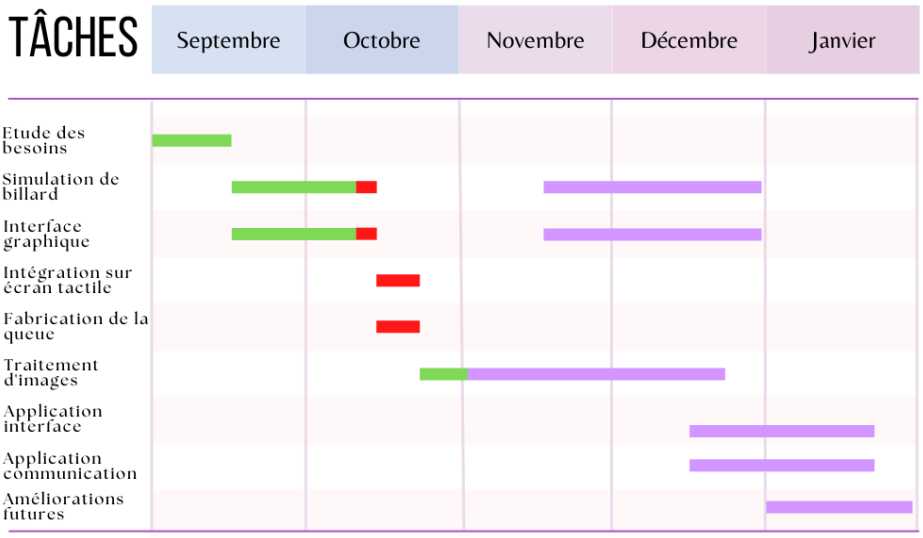

FIGURE 3 – Gantt prévisionnel du 10/11/2022

Et voici le planning effectif des réalisation à la fin du projet, avec répartition du personnel :

1.3 Détermination des parties prenantes

Une matrice RACI (Responsable, Approbateur, Consultant, Informés) a été créée pour déterminer les parties prenantes de notre projet. Cela a permis de déterminer quels sont les membres qui participent aux tâches et quel est leur rôle vis-a-vis de celles-ci :

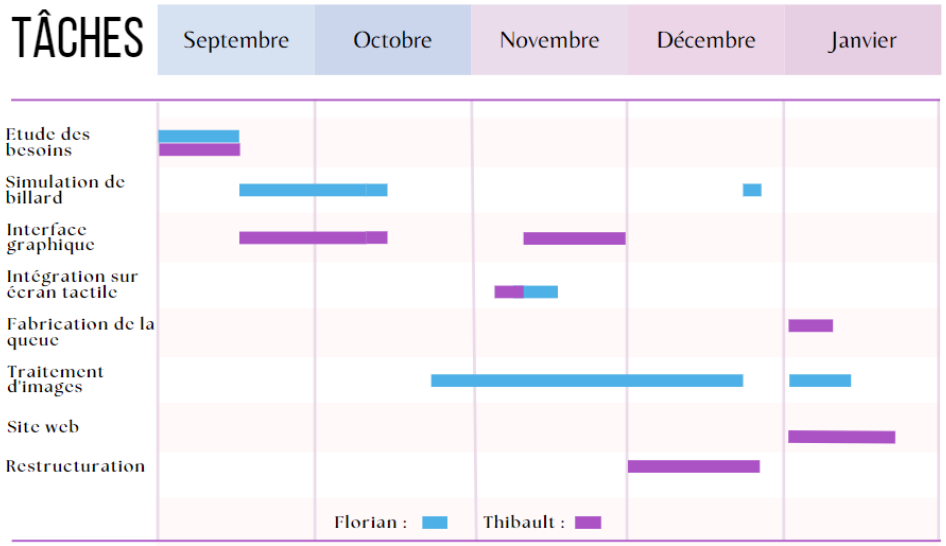

FIGURE 4 – Planning final

	Florian	Thibault	Tuteurs	Externes
Simulation billard	A, R	C, I, A	C, I	C
Interface graphique	C,I,A	A,R	C, I	C
Traitement d'image	A,R	C,I,A	C, I	C
Site web	C,I,A	A,R	C, I	C
Queue de billard	C,I,A	A,R	C, I	C
Restructuration et optimisation code	R,A	R,A	C, I	C

FIGURE 5 – Matrice RACI

Les externes peuvent être des enseignants ayant fourni des conseils, des amis ayant donné leur avis, etc. Ils ont donc un rôle de consultant. Les responsables sont ceux qui exécutent les tâches, les approbateurs valident les tâches. Pour finir, les informés sont tenus au courant régulièrement sur l'avancée de la tâche (les tuteurs et les autres membres du projet par exemple).

1.4 Détermination des ressources

Les éléments physiques utilisés sont :

— La table réalisée par les ancien étudiants IMA5, soit la structure en bois, son ordinateur et sa caméra
— une bâton en bois de 1m de longueur et 18mm de diamètre pour la queue de billard
— Plastique PLA pour l'objet à détecter (plus de détail dans la section 3.2)
— Élément de fixation (vis hexa, rondelles, écrous)

Les éléments logiciels utilisés sont détaillés dans les autres parties du rapport. Très peu d'élément matériel ont été utilisés puisque la plupart du projet consiste en du développement informatique.

1.5 Méthode de réalisation des tâches

Comme quasi-intégralité du projet est de la programmation, il faut mettre en place une méthodologie de développement efficace pour :

— Optimiser le temps de réalisation des tâches.

— Éviter d'oublier des fonctionnalités et facilité leur implémentation.
— Minimiser le temps passé à corriger les bugs.

1.5.1 Méthodologie

Pour chaque tâche, il faut suivre un processus simple. Dans un premier temps, une petite réunion entre membres du groupe est tenue pour décider des fonctionnalités à mettre en place, les librairies et technologies à tester, etc. Ensuite chaque membre du groupe développe les principales fonctionnalités de sa tâche assignée. Les membres réalisent alors un premier jet pour identifier les bugs. Ensuite, on procède à une phase de correction des bugs. Lorsque tout les bugs sont corrigé, soit on passe à la tâche suivant ou le développeur ajoute de nouvelles fonctionnalités. Certaines fonctionnalités présentes dans la version finale du projet ont, par exemple, été ajoutées car l'un des membres a jugé l'a jugée pratique ou ergonomique.

Cette méthode permet de n'oublier aucune des fonctionnalités de la simulation, tout en évitant les bugs. Cependant un problème se pose : Comment peut-on faire pour intégrer en continu le développement de différentes tâches si elle interagissent entre-elles (par exemple, pour tester la simulation, il faut que le développement de l'interface graphique soit un minimum avancé). C'est pour cette raison que Git à été utilisé.

1.5.2 Git

Git est un logiciel de versioning. C'est un système libre et gratuit, créé en 2005 par Linus Torvalds, connu également pour sa majeure contribution au noyau Linux.

Git permet notamment de garder plusieurs versions de fichiers sous forme de 'commit'. Une autre fonctionnalité très utilisée dans le projet est le système de branche de Git permettant à différents membres de travailler sur des fonctionnalités séparées puis lorsqu'une fonctionnalité est terminée, l'intégrer sur la branche principale en fusionnant le code.

Dans le cas particulier du projet, plusieurs branches ont été créées, une branche pour l'implémentation d'openCV, une pour la restructuration du projet ou encore une pour le découpage de l'interface graphique en vues.

2 Recherche bibliographique

Avant de présenter les travaux réalisés, l'équipe projet a fait un travail de recherche bibliographique. Les recherche se sont portées sur :

- Une étude de l'existant présentée dans le cahier des charges, dont un rappel se trouve dans la suite de ce rapport ;
- Une étude sur les compatibilités entre les technologies utilisées pour faire du traitement d'image et les langages de programmation les plus optimisés pour réaliser une simulation de jeu ;
- Une étude sur la physique et les mathématiques liée au jeu de billard ;
- Une étude sur les règles générales du billard dans le cadre de la réalisation de la simulation de billard.

2.1 Étude de l'existant

En 2016, un concept de table de billard à vu le jour, la Elysium Pool Table [1] est une table de billard connectée qui :

- Aide les personnes pour la trajectoire des coups à jouer ;
- Est dotée d'une caméra 4K sur le dessus pour enregistrer les mouvements ;
- Est dotée de LED intégrée pour savoir où placer le rack triangulaire et la boule blanche.

Cependant ce dernier est un table physique à part entière, elle se joue avec de vraies boules ce qui diffère totalement du notre projet.

La technologie AR a déjà été utilisée dans le cadre d'assistants de trajectoire [2] où une caméra sur le dessus détecte la position de la queue de billard et propose la trajectoire du coup à venir. Mais cette technologie n'a pas encore été utilisée pour contrôler la simulation sur l'écran à partir de la queue de billard.

2.2 Étude des technologies de développement pour la simulation

Dans un premier temps, l'équipe projet a étudié les possibilités de technologies pour réaliser du traitement d'image. Le plus simple semblait d'utiliser la technologie OpenCV codée en C++, qui offre une large gamme de classes et de méthodes facilitant le traitement d'images. De plus, cette technologie étant très utilisé elle est donc très documentée. Bien que cette technologie soit codée en C++, il existe également des moyens de l'intégrer dans du code Java ou Python, ce qui offre davantage de flexibilité.

Un langage orienté objet semble la meilleur option, puisque il est plus simple de s'y retrouver en fragmentant les programmes en classes. De plus un langage orienté objet semble particulièrement adapté car on peut imaginer assez facilement des classes pour chaque objet physique du billard (boules, trous, table, etc).

A la suite de cette première réflexion, il restait trois candidats potentiels :

- C++/C#
- Python
- Java

Le C++ n'a pas été retenu car les membres de l'équipe projet ne l'avaient jamais utilisé et bien qu'il s'agisse d'un langage moderne et un des meilleurs en termes de performances, celui-ci ne comporte pas de Garbage collector qui est une fonctionnalité bien pratique pour le développement informatique.

Le python n'a pas non plus été retenu malgré sa grande ergonomie car trop lent pour faire de la simulation et du traitement d'image en temps réel.

Le langage retenu est donc Java, car c'est un langage facile d'utilisation et qui reste rapide. C'est un bon compromis entre l'ergonomie de Python et les performances du C++. De plus, les membres de l'équipe projet ont déjà travaillé

1. https ://luxe.net/elysium-pool-table-billard-ne-lavez-jamais-pratique/
2. https ://www.researchgate.net/publication/266394007_PoolLiveAid_Augmented_reality_pool_table_to_assist_inexperienced_players

avec ce langage au cours de leur études.

En allant plus loin dans la réflexion, 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.

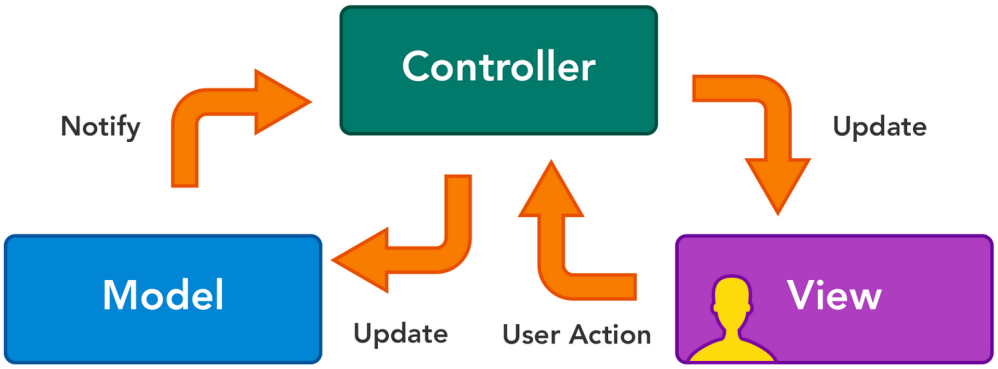

FIGURE 6 – Illustration du modèle mvc

2.3 Mathématiques et physique liée au billard

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

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

$$v = \sqrt{v_x^2 + {v_y}^2} \tag{1}$$

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

$$x_{t+1} = x_t + v_x \cdot \Delta t \tag{2}$$

$$y_{t+1} = y_t + v_y \cdot \Delta t \tag{3}$$

La nouvelle vitesse de la boule à un intervalle de temps donné doit être calculée 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) :

$$intensity_X = \frac{v_x}{v} \tag{4}$$

$$intensity_Y = \frac{v_y}{v} \tag{5}$$

3. On applique l'accélération au vecteur vitesse calculé en (1) où :

$$v_{t+1} = v_t + a \cdot \Delta t \tag{6}$$

4. Finalement, on recalcule les vitesses en deux dimension en multipliant (6) par (4) et (6) par (5).

2.3.2 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 :

$$ma = \sum F = -mg\mu_c \tag{7}$$

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, on choisit la valeur de 0,1, partant du principe que le frottement entre la table et la boule était élevé [3].

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

2.3.3 Collision entre une balle et un mur

Pour calculer la réponse de collision entre un mur et une boule : le mur du haut et du bas aura pour effet de changer le sens de la vitesse de la balle sur l'axe des x. En d'autres termes, si la vitesse était positive, alors elle sera négative et vice-versa. Le mur de gauche et de droite aura le même effet mais pour l'axe des y.

2.3.4 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 :

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

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

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

3. http ://www.pats.ch/formulaire/tables/tables1.aspx

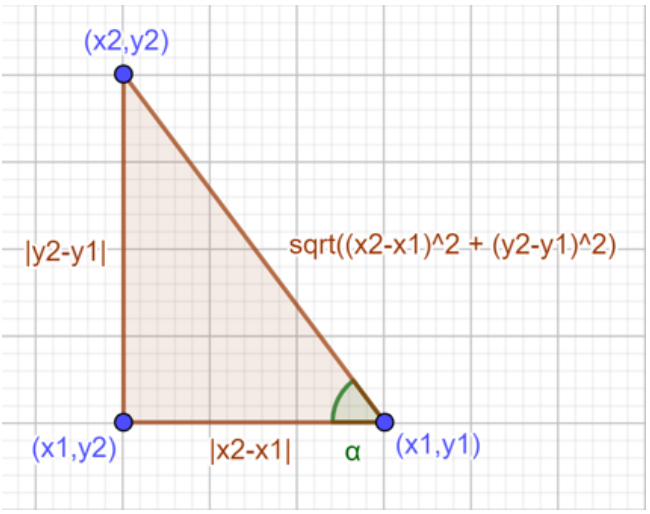

FIGURE 7 – Principe de détection de collision entre deux boules

Par ailleurs, les valeurs maximales et minimales qu'une boule peut avoir dans des conditions réelles, un coup très puissant peu d'atteindre 16m/s par exemple [4]

2.3.5 Réponse d'une collision entre deux boules

Le calcul de collision entre deux boules est très étudié par les physiciens, elle peux se ramener à une collision élastique [5]. 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 :

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

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.

4. https ://billiards.colostate.edu/faq/speed/typical/
5. https ://en.wikipedia.org/wiki/Elastic_collision

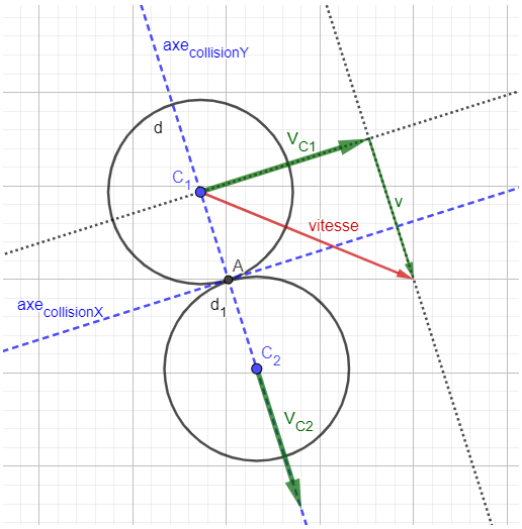

FIGURE 8 – Principe de réponse d'un collision entre deux boules

2.3.6 Pour aller plus loin

Il y a d'autre phénomènes liés à la physique du billard [6] qui ne sont pas pris en compte dans la simulation. Par exemple, l'analyse du "sweet spot" [7] où alors de l'impact des vitesses angulaires appliqué à une boule de billard.

2.4 Étude des règles du billard

Pour le projet, on étudie les règles du billard américain [8]. On aurait pu étudier d'autres modes de jeu, mais il est préférable de se concentrer sur la réalisation du cahier des charges avant d'ajouter des fonctionnalités supplémentaires.

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 avant de s'arrêter.
  - La boule blanche touche une boule adverse ou la boule noire en premier.
  - 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.

6. https ://www.real-world-physics-problems.com/physics-of-billiards.html
7. https ://www.billiardworld.com/sweet.html
8. https ://www.colorado.edu/umc/sites/default/files/attached-files/8-ball_rules_bca.pdf

3 Travaux réalisés

3.1 Développement de la simulation

La première étape est la création du début à la fin d'une simulation de billard. A première vue cela peut sembler une perte de temps mais il est plus simple de programmer une simulation entièrement plutôt qu'utiliser une simulation open source pour plusieurs raison :

- Il sera plus évident d'intégrer de nouvelle fonctionnalité dans la simulation pour la suite des travaux.
- Il sera difficile voire impossible de trouver une simulation qui correspond aux besoins.
- Il est difficile de comprendre les programmes complexes écrits par d'autres programmeurs.
- Il peut y avoir des risques de sécurité ou même simplement des codes obselètes.

La hiérarchie des classes de la simulation réalisé est la suivante :

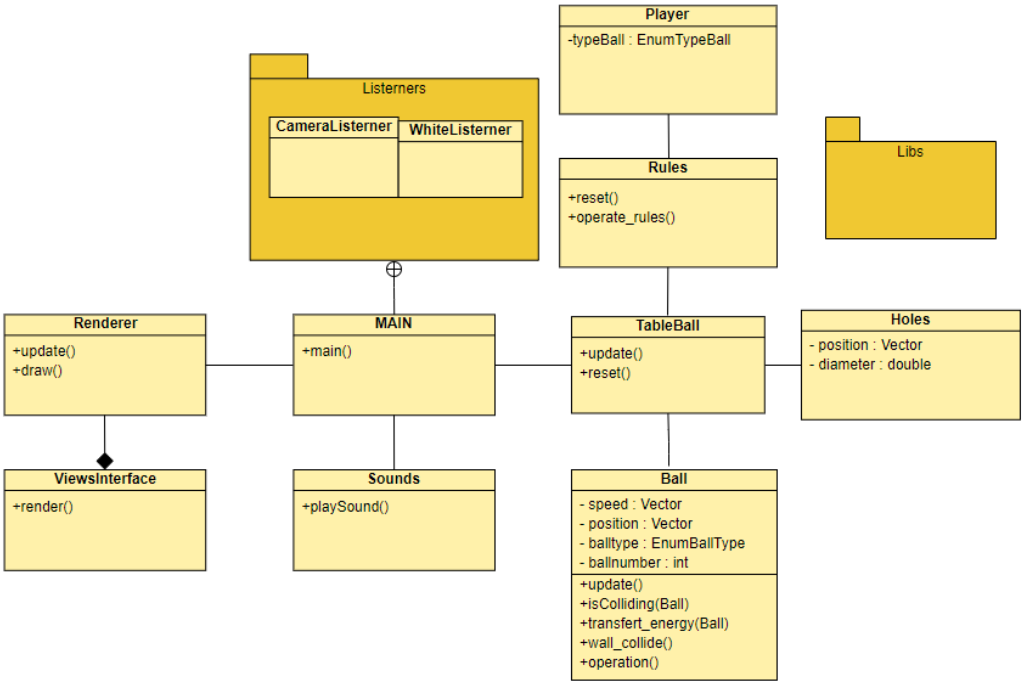

FIGURE 9 – Diagramme des classes

Cette partie explique, le fonctionnement de la simulation, ce qui a été fait, ce qui reste améliorable par la suite et ce qui a pris du temps. Cette partie montre également les résultats de la simulation.

3.1.1 Les modèles

Les différentes classes de modèles qui composent l'application sont :

La classe "**BallTable**" qui représente la table de jeu La table de jeu est composée de :

- 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 indiquant 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.
- Une méthode de remise à zéro de la table.

L'algorithme de mise à jour de la table de jeu est développé de façon à ce que deux boules ne puissent pas se superpose. Pour réaliser ceci, il suffit de ne pas mettre à jour la position d'une balle si celle-ci est en collision avec

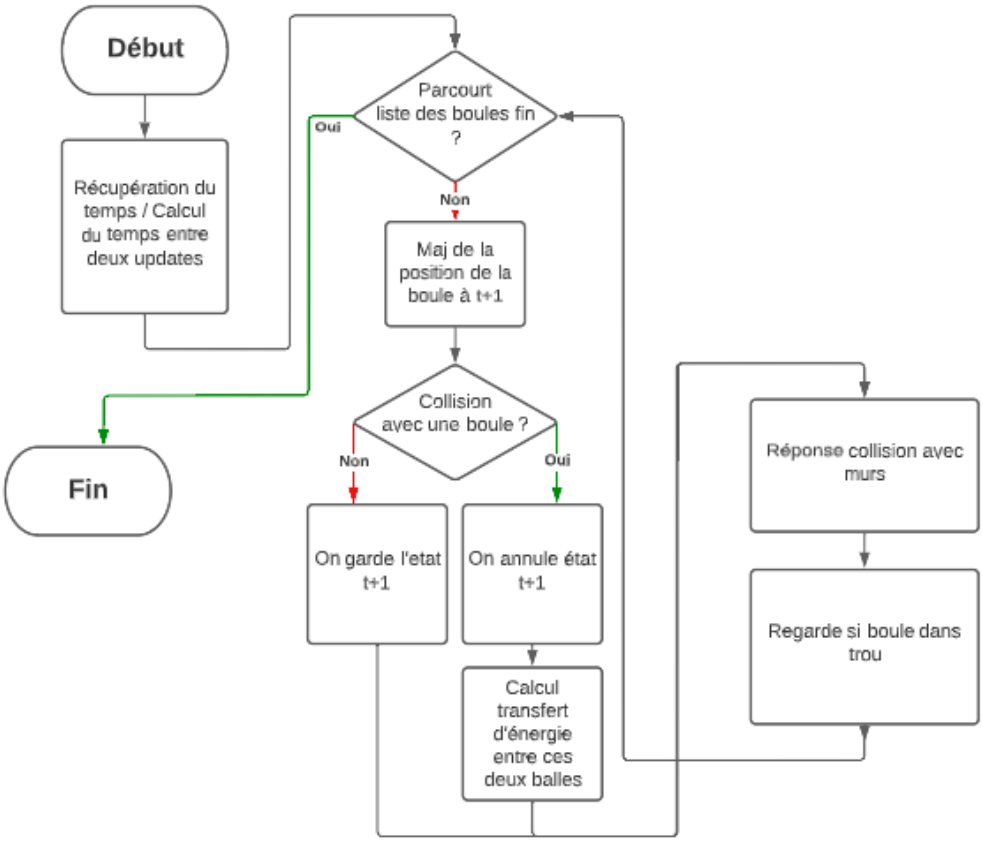

FIGURE 10 – Algorithme de mise à jour de la Table de jeu

une autre. Cette contrainte permet d'éviter les problèmes qui sont présent dans les premiers jet (voir section 3.1.4), les premiers algorithmes développés ne sont pas pris en compte pour ces raisons. Il y aura certes un léger retard mais, ce retard ne sera pas visible puisque le temps entre deux calculs est largement inférieur au temps entre deux rafraîchissements d'image.

En réalité, cette méthode de mise à jour est utilisée dans la plupart des simulations physiques. C'était une erreur de la part de l'équipe projet ne pas s'orienter sur cette méthode dès le début du développement de la simulation. Ce qui a coûté beaucoup de temps de recherche de solutions. C'est monsieur Alexandre Boé qui a fait part à l'équipe projet de cette solution.

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 des méthodes utilitaires ("getters" et "setters[9], une 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 (défini à la section 2.3).

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

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

Elle permet seulement de référencer deux joueurs dans la partie et leur attribuer une boule conformément aux règles du billard américain.

L'interface "**Hole**" et ses implantations gèrent le comportement des trous. Un billard possède plusieurs type de trous :

9. https ://www.geeksforgeeks.org/getter-and-setter-in-java/

des trous ronds sur les côtés et en Oblong [10] dans les coins. Chaque type de trou a sa forme mais possède une méthode de calcul imposée par l'implantation de l'interface "Hole" qui indique s'il y a collision avec une boule ou non.

Pour notre projet, il était possible de développer les différents type de trous si le temps le permettait. Seuls des tests de développement sont réalisés, et seul le trou circulaire est intégré dans la simulation finale.

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. Cependant, seulement les règles du billard américain est programmé conformément à la section 2.4

Cette classe récupère 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.

3.1.2 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 qu'elle dessine un élément à la fois.

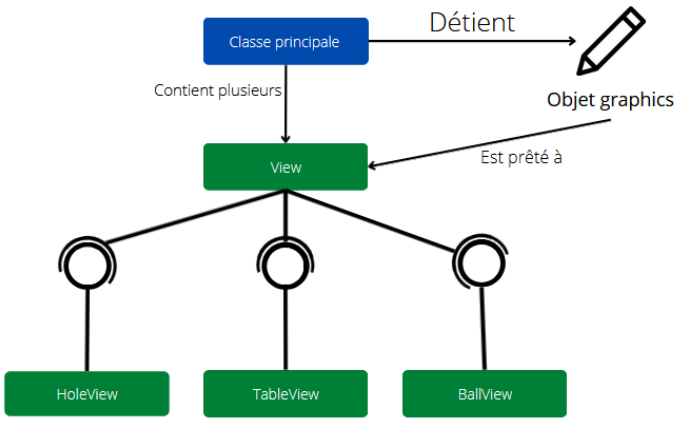

FIGURE 11 – Illustration du fonctionnement de l'affichage

Cette solution permet d'ajouter facilement de nouvelles vues si nécessaire, et assez facilement pour gagner du temps sur le développement (notamment pour afficher sur la simulation, la position de la queue de billard à la section 3.3.3).

On a développé toute les vues qui sont nécessaire pour l'interface de la table de billard. Pour aller plus loin sur la qualité de notre simulation, on aurait pu aussi développer des vues supplémentaires, comme par exemple, une vue pour représenter les bandes extérieurs de la table de billard. Des vues non-intégré dans la simulation sont tout de même crée :

— Une vue pour le trou Oblong (vue dans la section 3.1.1).

10. https ://fr.wikipedia.org/wiki/Oblong

— Une vue "bulle" pour afficher du texte par exemple pour afficher les règles du jeu ou le joueur qui doit jouer.

3.1.3 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.

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. On retient cette méthode pour intégrer plus facilement nos listeners dans la classe principale. De plus qu'il sera plus simple de s'y retrouver dans les phases de débugage du programme où bien si des développeurs tiers (ou nous même) voudraient reprendre notre projet pour l'améliorer.

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 "WhiteListener" qui permet à l'utilisateur de placer la boule blanche en cas de faute de l'adversaire.

Ces classes sont implémentés dans un premier temps pour tester la simulation sans la queue de billard, elles sont ensuite désactivés pour laisser place à la détection de la queue de billard. On laisse la possibilité à l'utilisateur de les activer.

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

1. l'instanciation de de la table de jeu ;
2. l'instanciation de la classe d'affichage (Renderer) ;
3. l'instanciation des listeners ainsi que leur liaison avec les vues ;
4. 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 ;
5. lancer la boucle principale qui gère le jeu (mise à jour de la table, respect des règles, etc).

3.1.4 Résultats de la simulation

Voici le résultat final lorsque qu'on active toutes les vues pour l'interface graphique :

L'exécution se fait sur une machine avec un système comportant une JVM [11], peu importe son système d'exploitation, d'où l'intérêt d'utiliser Java. La seule limitation est qu'il faut un écran connecté à la machine pour pouvoir afficher la simulation. Les collisions marchent correctement, on remarque seulement quelques fois un problème de boules qui se rentrent dedans :

11. Java Virtual Machine

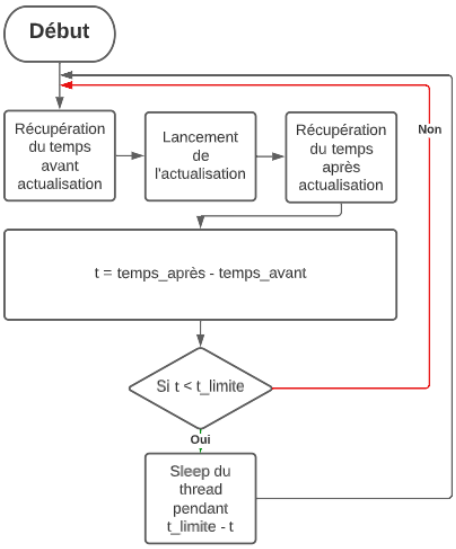

FIGURE 12 – Algorithme du thread d'actualisation de l'interface graphique

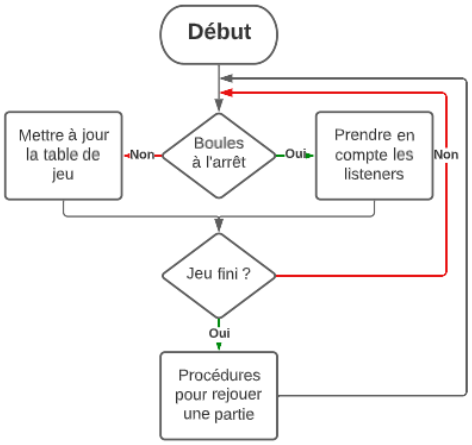

FIGURE 13 – Algorithme de la méthode main

Ce problème était due à la première version de l'algorithme d'actualisation de la table de jeu. L'ancienne version faisait se superposer les boules dans certaines conditions. Comme expliqué à la section 3.1.1, ce problème à été corrigé avec l'implémentation de l'algorithme décrit dans la figure 10 même si cela a pris beaucoup de temps.

Avant de réaliser ce nouvel algorithme, une semaine a été alloué pour trouver des solutions, 4 tests ont été réalisés pour comprendre le problème et tester des solutions :

- Utiliser un tableau de 16 booléens pour chaque boule afin de savoir si la collision entre les boules a déjà été calculée et ainsi ne calculer qu'une seule fois chaque collision.
- Augmenter la distance entre chaque boule lors de la casse puisque on remarque que ce bug se produit lors du premier coup (le coup qui réalise le plus de collision entre boules).
- Changer le moment où l'on actualise la position d'une boule (avant ou après le calcul de collision).
- Réaliser une ébauche de calcul de prédiction, mais cette solution était trop complexe à programmer et n'a pas été finalisée.

Au final, l'algorithme retenu est tout à fait performant et même s'il existe une probabilité non nulle qu'un bug apparaisse, l'équipe projet n'en a pas revu lors de la suite du développement. Voici un lien vers une vidéo de démonstration.

Du temps a été pris sur la correction d'un problème d'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

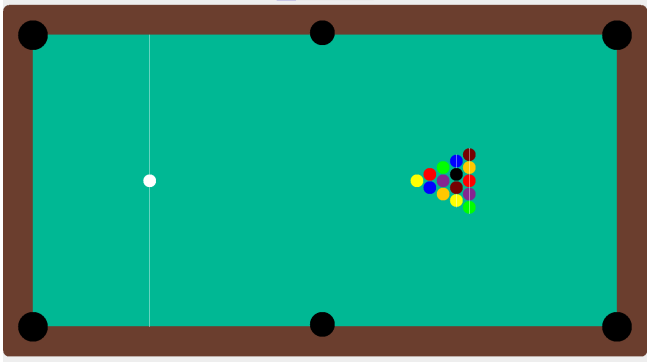

FIGURE 14 – Rendu de l'interface graphique

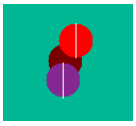

FIGURE 15 – Bug de collision

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 été résolu en utilisant une API différente pour mettre l'application en plein écran.

3.2 Modélisation de la canne de billard

Pour la modélisation de la canne, il y a quatre points essentiels :

1. La modélisation doit comporter une partie plate et de forme rectangulaire pour y installer un matériau réfléchissant ;
2. Elle doit aussi comporter une alcôlve en dessous de la partie plate pour y mettre une mousse afin de ne pas rayer l'écran lors de l'utilisation ;
3. Le modèle devra être imprimable avec les imprimantes du fabricarium de polytech. Autrement dit, le modèle devra pouvoir tenir dans un cylindre de 18cm de diamètre et de 20cm de hauteur.
4. Il devra être imprimable sans supports. Les imprimantes ne pouvant pas imprimer dans le vide, il faut donc toujours s'assurer qu'il y ait des pentes entre les différentes couches et que ces pentes aient un angles inférieur ou égal à 45°.

3.2.1 Premier jet

Au départ, la totalité de la canne avait été modélisée d'un coup avec une surface pour le support réfléchissant de 3x2cm et une longueur de 25cm. Il a donc fallu réduire la taille de tout le modèle pour que celui-ci puisse être imprimé en une seule fois.

Ce premier jet n'est pas utilisable car la surface pour le matériau réfléchissant est trop petite pour être utilisable par openCV (après réduction la surface fait à peine plus de 1x1.5cm) et le manche de la canne est trop petit ce qui la rend peu ergonomique.

De plus, pour pouvoir être imprimé sans support, il a fallu imprimer le modèle sur la tranche ce qui a aplati le manche le rendant peu esthétique.

3.2.2 Deuxième jet

Suite au premier jet, il était clair qu'il fallait une surface bien plus grande pour le matériau réfléchissant. il a été décidé que 5x4cm conviendrait.

FIGURE 16 – Premier jet de la modélisation de la queue

Pour éviter de devoir réduire le modèle comme pour le premier jet, plusieurs solutions ont été envisagées. La première était de découper le modèle en plusieurs modules de sorte à pouvoir les connecter bout à bout pour former la canne. Cette solution n'a pas été retenue pour plusieurs raisons. Tout d'abord, l'équipe projet n'avait pas les compétences pour créer un tel système. Ensuite, cela aurait pris bien plus de temps à imprimer, à assembler et cela aurait rendu la canne fragile.

La solution à laquelle est arrivé l'équipe projet a été de seulement modéliser la tête de la canne avec la surface pour le support réfléchissant ainsi que l'alcôve pour la mousse. De cette manière, en rajoutant deux trous dans le modèle, il est possible de fixer l'objet à un cylindre en bois à l'aide de vis à bois. L'équipe s'en est procuré un et y a fixé l'objet une fois imprimé.

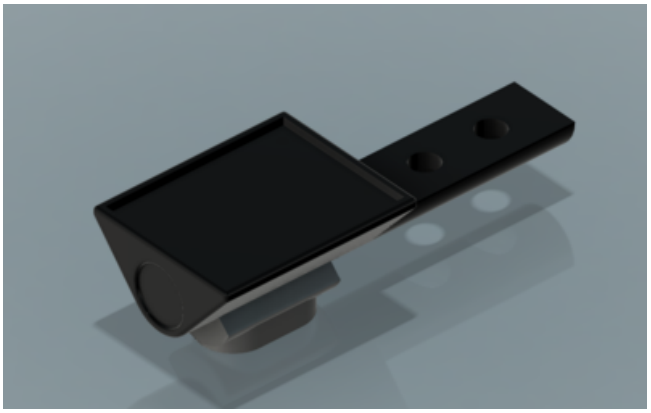

FIGURE 17 – Deuxième jet de la modélisation de la queue sur fusion 360

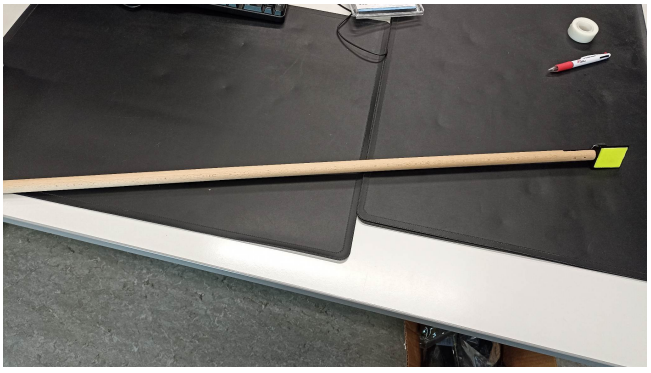

FIGURE 18 – Assemblage du deuxième jet sur canne en bois

3.3 Détection d'objet

Pour la détection d'objet, conformément à la section 2.2, l'équipe projet a décider d'utiliser la technologie OpenCV.

OpenCV [12] est une bibliothèque open source, initialement développée par Intel, spécialisée dans le traitement d'images en temps réel. Elle permet, entre autres, de travailler sur des projets :

- De détection d'objet
- De machine learning et de deep learning
- De traitement de photos ou vidéos
- etc

Cette bibliothèque remplit parfaitement les exigences du projet et permet de satisfaire les objectifs du cahier des charges. C'est donc cette solution qui a été retenue.

3.3.1 Rappel de l'objectif avec la détection d'objet

Le principal objectif de l'équipe projet est de contrôler la simulation de billard avec la queue de billard, décrite à la section 3.2. Il faut alors être capable de détecter l'objet au bout de la queue, transférer sa position dans la simulation et de développer des méthodes de calcul de vitesse lorsque la queue et la boule blanche seront superposées.

Pour commencer et prendre en main la librairie, l'équipe projet à d'abord réalisé un petit projet : Faire bouger une boule de la simulation avec un objet coloré réfléchissant similaire à celui fixé sur la canne.

3.3.2 Prise en main d'OpenCV

Dans un premier temps, pour avoir le résultat de la détection d'objet, une application en utilisant JavaFX est réalisée pour avoir en temps réel les résultats de la détection d'objet.

JavaFX est un peu similaire à Swing, il permet de réaliser des interface graphique en Java avec la même architecture. Comme les méthodes pour convertir les objets OpenCV en JavaFX sont déjà implémentées nativment dans la librairie, JavaFX est retenue pour réaliser cette interface.

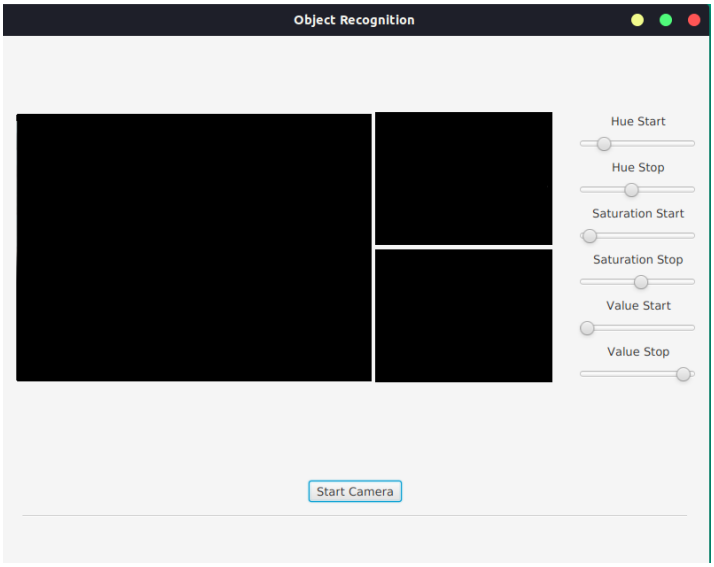

FIGURE 19 – Application JavaFX pour la détection d'objets

L'équipe projet a décidé de réaliser un algorithme de détection d'objet morphologique. La détection morphologique se déroule en plusieurs étapes :

12. https ://opencv.org/

1. Une préparation de l'image, avec OpenCV, il est possible de modifier le flux vidéo avant de traiter cette dernière, il est possible de modifier notamment :
   — La saturation, valeur, hue de base du flux vidéo
   — La rotation du flux vidéo
   — Le zoom sur le flux vidéo pour recadrer l'image
   Il est aussi conseillé de réaliser une réduction de bruit sur l'image en appliquant un filtre de flou gaussien.

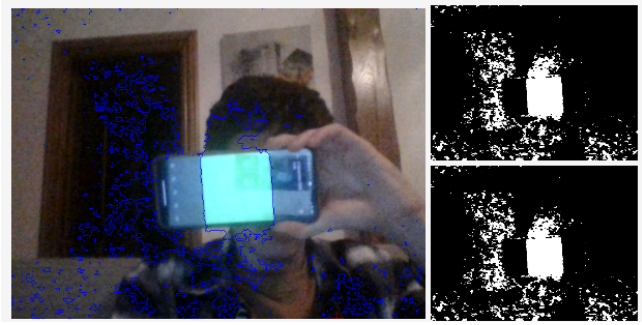

FIGURE 20 – Image avec bruit

FIGURE 21 – Image avec réduction du bruit

   On obtient une matrice d'éléments en format HSV.

2. Application d'un masque HSV sur l'image :
   — "Hue" contrôle la couleur
   — "Saturation" contrôle le niveau de gris de l'image
   — "Valeur" contrôle le niveau de luminosité de l'image

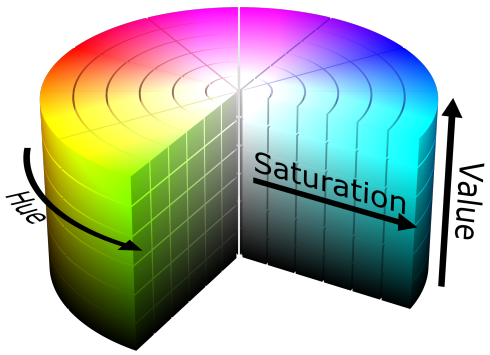

FIGURE 22 – Niveaux HSV

   On passe l'image du format RGB à HSV puisqu'il est possible de gérer facilement la suppression des arrières plans, ou bien gérer la détection de luminosité de l'objet et surtout de pouvoir supprimer facilement certaines couleurs en changeant la Teinte ("Hue").

   Le programme produit une matrice de points binaires (noir ou blanc).

3. Application des opérations de morphologie, soit une érosion (diminution de la taille des objets détecté) et une dilatation (augmentation de la taille des objets détecté), l'érosion à pour but de supprimer les objets de petites tailles, pour en garder les plus grosses. La dilatation permet de revenir à l'état précédent sans les objets les plus petits.

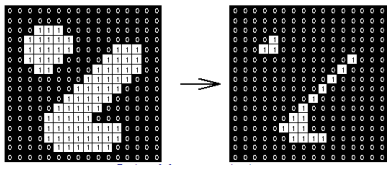

FIGURE 23 – Principe de l'érosion

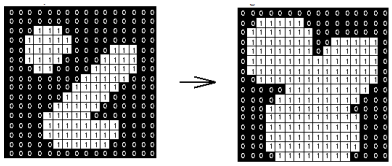

FIGURE 24 – Principe de la dilatation

Après application des opérateurs morphologique, voici le résultat :

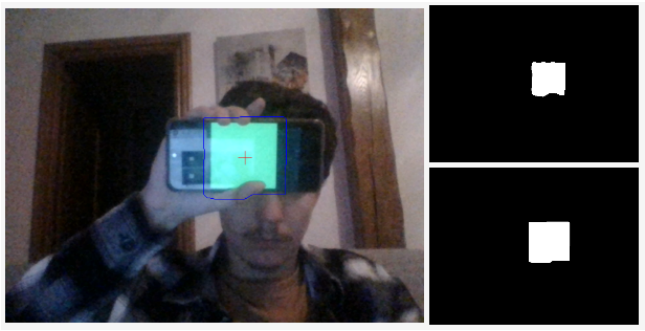

FIGURE 25 – Résultat de la détection d'objet

Voici ce que l'on peut faire avec la détection d'objet obtenue :

— On peut afficher les contours des objets détecté.
— On peut calculer les rectangles détecté à partir des contours, puis calculer leur centre, taille, ...

Pour faire bouger la boule, l'équipe projet doit alors calculer les contours des objets détectés puis donner forme à des rectangles pour connaître le centre et la taille des objets détectés. Avec ça, il est possible de donner une position à la boule pour la déplacer.

Voici un lien vers une vidéo de démonstration.

3.3.3 Réalisation avec OpenCV sur la simulation

Comme la caméra ne capture pas que l'écran, il faut mesurer la taille en pixels de l'écran tactile (e.g la table de billard) sur le flux de la caméra.

Pour pouvoir détecter les bords de l'écran, l'équipe a imprimé quatre rectangles en PLA rose, qui sont ensuite détectés à l'aide du programme de détection d'objet, avec les 4 positions récupérées, il est possible de calculer les vecteurs définis à la figure 26. On appelle alors le décalage entre l'écran et le bord du flux vidéo des offsets.

Cette partie du projet gérant le traitement d'image permet également de transmettre la position de la queue à la simulation en tenant compte des offsets précédemment décrits grâce à des conversions.

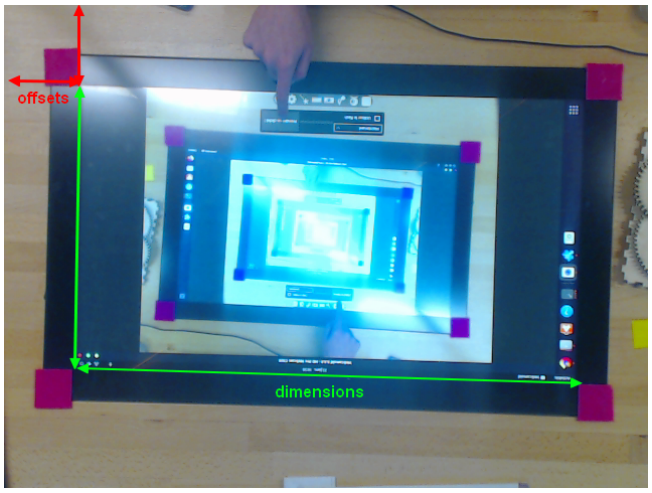

FIGURE 26 – Sortie vidéo

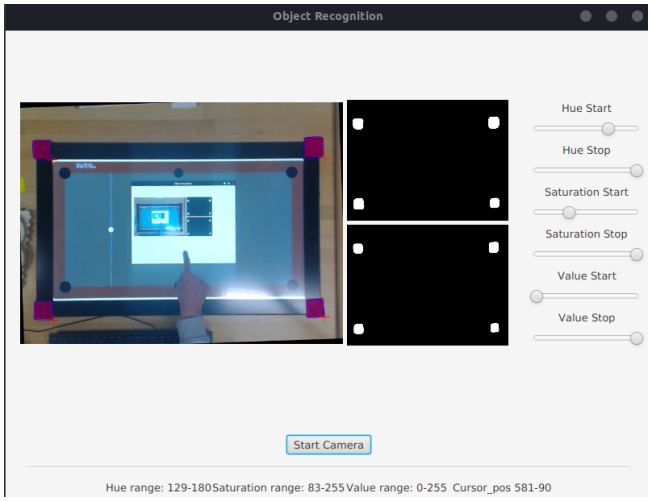

FIGURE 27 – Recherche des paramètres optimaux des offsets

L'autre application est la détection de la queue de billard et le contrôle de la simulation avec cette dernière. L'objectif était de chercher les paramètres optimaux pour détecter du jaune et de retransmettre la position de la queue à la simulation en utilisant l'application précédente.

Il a été difficile de trouver les paramètres optimaux pour détecter le jaune (ou autre couleur) puisque la couleur perçue par la caméra varie beaucoup selon la luminosité de la pièce, qui change selon l'heure de la journée. De plus, la lumière émise par l'écran a une forte influence sur le traitement de l'image. Pour palier à ce problème, l'équipe projet a envisagé de fermer les volets de la salle et de contrôler la luminosité en allumant toujours les mêmes lumières mais la salle étant utilisée pour d'autres activités par Polytech, l'équipe projet a préféré recalibrer la caméra lorsque les performances se dégradaient.

Dans la simulation, les 10 dernières positions identifiées par le programme de traitement d'image sont stockées dans une liste. Dès que la queue se superpose avec la boule blanche, on applique une force selon la vitesse de la queue et l'orientation de son vecteur qu'on calcule grâce aux précédentes mesures de la position de la queue. Ceci fait, voici un lien vers une vidéo de démonstration.

Une piste d'amélioration serait de rendre possible le fait de déplacer la boule blanche en cas de faute en utilisant l'application de traitement d'image, ce qui n'est pas possible à l'heure actuelle. On peut imaginer un autre objet avec un matériau réfléchissant comme un gant, par exemple.

3.4 Diffusion de la partie

Le cahier des charges spécifiait qu'il devait être possible de suivre les parties en direct depuis un smartphone en Bluetooth avec une application mobile.

Après avoir beaucoup réfléchi au sujet, l'équipe projet a envisagé une autre approche. En effet, créer une application mobile exploitant des flux vidéos au travers du Bluetooth n'est pas une compétence qu'il était simple d'acquérir en si peu de temps.

L'équipe projet a donc plutôt pensé à retransmettre le flux vidéo de l'écran sur un serveur web afin de rendre le flux disponible à tous les appareils sur le même réseau. Installer le serveur web sur une machine connectée au web permet de voir les parties de billard de n'importe où pour peu qu'on ait un accès à internet.

Pour ce faire, l'utilitaire ffmpeg a été installé afin de créer un flux vidéo de l'écran. Sur une machine de TP connectée à internet, le serveur web Nginx avec un module permettant de rediffuser les flux vidéos en rtmp ont été installés.

Après une longue session de configuration, il est possible de voir les parties en direct en se connectant au flux vidéo suivant avec un utilitaire adapté tel que VLC : `rtmp://brochette.site/live/poolstream`

Non seulement cette solution permet à l'utilisateur final de ne pas avoir à installer d'application supplémentaire s'il a déjà un lecteur de vidéo ; mais elle permet également de rendre disponible les parties à tous les systèmes d'exploitations. On peut ainsi voir les parties sur Android, IOS, Linux, Windows, MacOS, etc.

Un autre format de vidéo a été exploré par l'équipe projet : le format HLS. Ce format permet, en théorie, de diffuser les parties dans une page web directement sans passer par un lecteur de flux. Cela permettrait de rendre disponible les parties sur n'importe quel type d'appareil si celui-ci possède un navigateur web comme des consoles de jeux.

Malheureusement, cette solution n'a pas abouti par manque de temps, tout d'abord, le format HLS est supporté nativement par très peu de navigateurs excepté sur smartphone ou la plupart des navigateurs sont compatibles. Ensuite, pour des raisons inconnues, le stream HLS renvoyé par le serveur NGINX ressort tout noir et sans le son. Ce qui est bizarre car il a été possible de streamer des vidéos enregistrée dans des fichiers. L'origine du problème reste donc à déterminer, l'équipe projet n'ayant pas pu le faire par manque de temps.

3.5 Restructuration du code et optimisation

Au fur et à mesure du projet, les nouveaux fichiers ont commencé à s'accumuler et il a été important de passer par cette étape pour :

- Rendre le code plus lisible, plus compréhensible afin de faciliter la reprise du code par d'autres développeurs où l'équipe projet.
- Limiter la consommation des ressources CPU / RAM

Cette étape consiste à :

- Découper le code en plusieurs classes.
- Optimiser le programme pour pouvoir ajouter des phases de mise en sommeil du programme.
- Faire du multi-threading pour exécuter plusieurs tâches en même temps.

Après des sessions d'optimisation, des tests de performance de la simulation on été réalisés. Pour rappel, l'équipe projet est dotée d'une unité centrale avec un CPU 4 coeurs cadencés à 3.1GHz, un processeur graphique intégré et 16Go de RAM.

Tests	Utilisation CPU	Utilisation RAM	OpenCV
Billard avant optimisation Billard après optimisation	30% constant 30% maximum, 1% minimum	235Mo 230Mo	
Billard avant optimisation + OpenCV Billard après optimisation + OpenCV	50% constant entre 20% et 40%	250Mo 250Mo	1 frame / 120ms 1 frame / 60ms
Flux vidéo (HD/30fps) Flux vidéo (SD/20fps)	70% 17%	400Mo 182Mo	
Ensemble optimisé	60%	500Mo	

Il est évident que le CPU est moins utilisé après la phase de restructuration et d'optimisation. De plus, après optimisation, il est possible de capturer une image toutes les 60ms avec OpenCV contre une image toutes les 120ms avant optimisation du code.

Même si cette étape prend du temps à réaliser, elle est primordiale pour limiter l'utilisation du processeur.

4 Prise de recul

4.1 Points forts du projet

L'un des point fort de la simulation est sa structure simplifiée, un gros effort a été fait pour rendre le programme facile à comprendre en structurant le mieux possible le code avec différente classes. Il est très facile d'ajouter de nouvelles classes dans le projet.

Le choix du langage de programmation a été judicieux. On remarque que la simulation est fluide et ne consomme pas trop en RAM et processeur malgré que nous utilisons un langage orienté objet.

Lorsque tout le projet est en marche (e.g simulation + détection d'objet + flux vidéo) l'ordinateur n'est pas surchargé et les applications ne se terminent pas anormalement (sur un test de 4h sans interruption).

La technique retenue pour la retransmission en direct permet à l'utilisateur de regarder la transmission vidéo sans télécharger d'applications tierce. Ce qui est un point fort pour l'expérience utilisateur.

La gestion de projet est également un point fort du projet. Les tâches étaient bien réparties dès le début du projet, ce qui n'a laissé aucun temps libre à l'équipe. De plus, avec les "mini-réunion", l'équipe s'est adaptée aux limites de temps pour la prise de décision (technologies, structures de programme, etc). Pour finir, avec la gestion de version Git, nous avons pu éviter les problèmes de travaux simultanés avec le système de branche.

4.2 Améliorations future

Dans un premier temps, nous avons fait l'intégralité de ce que était prévu dans le cahier des charges. Il y a cependant quelques étapes à optimiser :

- Pour l'interface graphique, cette dernière est améliorable, notamment en ajoutant les nouveaux types de trous (voir section 3.1.1) et les bandes extérieures comme sur les vraies table de billard.
- Pour la physique du billard, ajouter de la physique de billard avancée comme la détermination du "sweet spot" ou des vitesses angulaires pour pouvoir donner des effets aux balles.
- Pour la partie détection d'objet, il aurait été bien ajouter une méthode pour bouger la balle blanche avec la main à l'aide de la caméra et de OpenCV, nous avons seulement réfléchi à comment l'implémenter. Cependant, il est possible de bouger la boule blanche en cas de faute en touchant l'écran directement. Des problèmes de détection subsistent encore, une amélioration possible est de trouver les paramètres idéaux (couleur de détection, taille érosion, dilatation, ...), pour ne pas avoir de problèmes de détection.
- Essayer de mettre la retransmission des parties en direct sur une page web plutôt qu'en générant un flux rtmp afin que les utilisateurs n'aient pas à utiliser un visionneur de flux vidéo comme VLC.
- Optimiser encore le code pour limiter l'utilisation du CPU, ce qui a été fait en grande partie mais qui reste améliorable.

D'un point de vue gestion de projet, on peut remarquer que le diagramme de Gantt initial n'a pas été respecté. Par exemple, nous avions prévu de réaliser la partie "Détection d'objet" à deux alors qu'une seule personne s'en est chargé. Nous avons, en réalité, vite compris qu'il était plus simple d'assigner une seule personne à chaque tâche pour que ce dernier soit plus concentré dans celle-ci et pour ne pas se gêner.

D'un point de vue technique, on aurait pu intégrer de l'électronique sur la queue de billard pour la détecter plus facilement (e.g capteurs de distance par exemple ou accéléromètres). Nous n'avons pas étudié la possibilité plus en détail puisque cela semblait difficile en termes de délais et puisque nous avions, dès le début, une caméra à notre disposition. On remarque qu'il n'est pas évident de jouer avec la queue de billard et qu'il est plus facile de jouer avec l'écran tactile. Mais en améliorant encore un peu le projet, cela pourrait changer.

4.3 Point de vue commercial

Nous nous sommes s'est demandé si le projet était vendable ou non. Si nous reprenons le projet de l'Elysium Pool Table [13], ce projet est vendu 160 000€. Ou bien, des projets similaire utilisant des caméras en détection d'objet se vendent près de 10 000€. Une ébauche du budget utilisé pour ce projet a été réalisé :

Élément	Prix
Unité centrale	600€
Caméra	109€
Table en bois	800€
Queue de billard	4€
Total	1513€

Certains détails n'ont pas été pris en compte, comme le salaire des techniciens si ce projet était fait dans le cadre d'un projet en entreprise, le prix du matériel de développement (ordinateurs, etc), prix des technologies (logiciels, etc). De plus, la table n'a pas de prix fixe, le prix ci-dessus est une estimation.

Cependant, le projet est sensible à la luminosité de la pièce. Actuellement, le produit n'est pas vendable, mais en corrigeant ces problèmes, en optimisant encore les programmes, on peut parfaitement imaginer vendre le projet. On peut, par exemple, imaginer des caméras plus performantes et des objets à détecter avec des couleurs optimisées pour la détection d'objets.

4.4 Pour aller plus loin

Notre but était de développer un billard virtuel augmenté, mais les possibilités de projet avec la table en bois issue d'un projet des anciens IMA5 sont infinies.

Voici quelques idée pour l'avenir :

- Réalisation d'un ice-hockey virtuel augmenté à la façon du jeu "Pong" où les plate-formes pour faire rebondir la balle seraient remplacées par un objet réel détecté par la caméra.
- Réalisation d'un flipper virtuel augmenté où les "bumpers" seraient remplacés par des objets imprimés à l'imprimante 3D détectés par la caméra.
- Réalisation d'un jeu de carte virtuel, puisqu'on remarque des encoches pour installer des téléphones sur la table.
- Réalisation d'un jeu de rôle virtuel type Donjon et Dragon.
- Réalisation d'un jeu à rôles cachés type Loup-Garou
- etc, les idées se limitent à l'imagination.

13. https ://luxe.net/elysium-pool-table-billard-ne-lavez-jamais-pratique/

Conclusion

Nous avons eu des résultats positifs sur l'ensemble des tâches que nous avons définies dans le cahier des charges.

La simulation du billard est terminée, il reste quelques amélioration de "confort" et d'expérience utilisateur à apporter. Surtout que grâce au modèle MVC, il est facile d'ajouter de nouveau objets dans la simulation.

La détection d'objet a donné des résultats très satisfaisants, nous pu faire de nombreuses parties de jeu. Il y a cependant quelques aspect de la détection de la canne à améliorer.

Il est également possible de regarder la partie de jeu en direct grâce à la génération du flux vidéo.

Nous avons fait face à beaucoup de problèmes et d'imprévus mais nous avons su à chaque fois trouver des solutions pour continuer à avancer.

Grâce à l'importante phase de réflexion au début du projet, nous avons pu trouver les bonnes solutions dès le début de la réalisation des différentes tâches, ce qui nous a permis de gagner du temps.

En plus des améliorations qu'il est possible de faire sur notre projet, nous avons imaginé d'autres projets réalisable sur la table en bois réalisée par les anciens IMA5. Le fait d'avoir une machine avec un écran à laquelle d'autres appareils peuvent se connecter offre de très nombreuses possibilités en termes de divertissement.

Ce projet était très enrichissant d'un point de vue technique et managérial, nous avons pu utiliser toute nos connaissances puis nous avons pu nous enrichir sur de nouvelles technologies comme OpenCV, ffmpeg, Nginx. De plus que ce projet était très ludique et amusant à réaliser.