TPI - Physical Event Simulation

Simulation d’expérience physique en C++ avec la bibliothèque SFML

Auteur : Lucas Charbonnier

Classe : MID4

Lieu : ETML

Temps à disposition : 90 heures (28 avril - 28 mai 2021)

Chef de projet : Dimitri Lymberis

Experts : Xavier Carrel, Charles-Henri Hayoz

Table des matières

[1 Analyse 2](#_Toc70930266)

[1.1 Méthodologie de projet 2](#_Toc70930267)

[1.2 Technologies à utiliser 3](#_Toc70930268)

[1.2.1 Langage de programmation 3](#_Toc70930269)

[1.2.2 Bibliothèques 3](#_Toc70930270)

[1.3 Convention de code/nommage 3](#_Toc70930271)

[1.4 Méthode de simulation : discrète ou continue 3](#_Toc70930272)

[1.4.1 Choix de la méthode et design pattern 4](#_Toc70930273)

[2 Planification initiale 4](#_Toc70930274)

[3 Conception 5](#_Toc70930275)

[3.1 Maquettes de l’interface 5](#_Toc70930276)

[3.1.1 Menu principal 5](#_Toc70930277)

[3.1.2 Interface de la première expérience : « chocs » 5](#_Toc70930278)

[3.1.3 Interface de la deuxième expérience « balistique » 6](#_Toc70930279)

[3.2 Formules mathématiques pour la gestion des collisions 6](#_Toc70930280)

[3.2.1 Détection et résolution d’une collision entre 2 billes 7](#_Toc70930281)

[4 Réalisation 10](#_Toc70930282)

[4.1 Mise en place de l’environnement pour compiler le code 10](#_Toc70930283)

[5 Tests 10](#_Toc70930284)

[6 Conclusion 10](#_Toc70930285)

[7 Bibliographie 10](#_Toc70930286)

# Analyse

## Méthodologie de projet

Pour mener à bien le projet, il a été décidé que la **méthode des 6 pas** serait utilisée pour structurer son déroulement. Cela veut dire que le projet sera séparé en plus ou moins 6 étapes :

* S’informer 🡪 prise de connaissances du cahier des charges
* Planifier 🡪 planification initiale détaillée
* Décider 🡪 analyse et conception des fonctionnalités
* Réaliser 🡪développement de l’application
* Contrôler 🡪 Effectuer les différents tests
* Évaluer 🡪 En général, l’évaluation d’un projet sert à savoir ce qui a bien fonctionné (ou non) dans le but d’améliorer le ou les processus de travail.

## Technologies à utiliser

### Langage de programmation

Le langage de programmation utilisé sera le C++. C’est un langage bas-niveau et adapté à la création de simulation de physique. De plus, c’est le langage avec lequel je suis le plus familier.

### Bibliothèques

Le langage C++ seul ne permet de créer des applications graphiques, c’est pour cela que la SFML sera également utilisée. La SFML (Simple and Fast Multimedia Library) est une bibliothèque multimédia permettant, entre autres, de créer des applications graphiques.

Il est important de noter que la SFML n’est pas un moteur de jeu à proprement parler. C’est une bibliothèque graphique relativement bas niveau qui permet d’afficher des formes (rectangles, cercles, etc…) sur une fenêtre Windows mais elle ne propose pas de fonctionnalités avancées telles que l’affichage de widgets ou le calcul vectoriel. Le site officiel de la SFML : <https://www.sfml-dev.org/index-fr.php>

La détection/résolution de collisions élastiques nécessitera l’utilisation de calcul vectoriel, c’est pour cela que, en plus de la SFML, une bibliothèque de calcul vectoriel sera utilisée. J’utiliserai ma propre bibliothèque que j’ai développée durant mon temps libre : <https://github.com/Raynobrak/Charbrary>.

## Convention de code/nommage

Les normes de l’ETML en matière de programmation définissent la manière dont les classes, méthodes et variables doivent être nommées.

Cependant, ces normes sont définies pour le langage C#, PHP et javascript, alors que le projet sera développé en C++. Pour cette raison, les modifications suivantes ont été apportées :

* Les fonctions membres d’une classe suivront le lowerCamelCase au lieu du UpperCamelCase. C’est une convention la majorité des développeurs C++ suivent.
* Les fonctions libres (non-membre) suivront le snake\_case. Cela permet de les différencier clairement des fonctions membres.
* Les variables membres privées d’une classe seront suivies (au lieu d’être précédées) d’un underscore : « foo\_ ». Les variables précédées d’un underscore sont réservées aux implémentations de la STL du langage C++. Le respect de cette convention facilitera la maintenance du code puisqu’elle permettra d’éviter d’éventuels conflits de noms.

## Méthode de simulation : discrète ou continue

Il existe deux grandes manières de simuler des systèmes physiques : de manière discrète ou de manière continue. Il est nécessaire d’examiner ces deux méthodes pour pouvoir choisir laquelle est la plus adaptée au développement du projet.

Dans une simulation discrète, on calcule la simulation bout par bout jusqu’à ce qu’elle se termine. Pour cela, on définit un « pas de temps » (également appelé « time step » ou « ∆t ») qui correspond à la fréquence à laquelle la simulation sera mise à jour.

Un pas de temps petit (p.ex. 50 millisecondes) donnera un résultat plus précis qu’un pas de temps long (p.ex. 500 millisecondes) mais cela sera également plus couteux en puissance de calcul puisque, pour calculer 1 seconde de simulation, le premier ∆t nécessitera 20 mises à jour alors que le deuxième ∆t n’en nécessitera que 2. De plus, un pas de temps trop petit risque de donner un visuel saccadé.

Dans une simulation continue, le système physique est représenté comme une équation ou un système d’équations. On peut connaître l’état de la simulation à un certain moment en faisant varier le paramètre t dans l’équation.

C’est la méthode qui donnera le résultat le plus précis mais elle nécessite de disposer d’un modèle mathématique permettant de représenter le système sous forme d’équation, ce qui n’est pas toujours possible.

Prenons l’exemple d’un objet qui se déplace à une vitesse constante de gauche à droite. Dans une simulation continue, on pourrait calculer la position ***x*** de l’objet en fonction de la vitesse ***v*** et du temps écoulé ***t*** depuis le début de la simulation :

Alors que dans une simulation discrète, la position de l’objet à une certaine frame ***i+1*** sera calculée en fonction de sa position à la frame précédente ***i*** et en fonction du ***∆t*** et de sa vitesse :

En d’autres termes, dans une simulation continue on calcule directement la position absolue alors que dans une simulation discrète celle-ci se calcule en faisant la somme de tous les déplacements.

### Choix de la méthode et design pattern

Les simulations discrètes sont presque toujours plus simples à implémenter que les simulations continues parce que les formules à concevoir sont forcément plus simples (pour la même raison qu’il est plus facile d’approximer une intégrale plutôt que de calculer sa valeur exacte).

Une simulation continue aurait pu être considérée pour la deuxième expérience mais le fait qu’il faille gérer les collisions avec le « panier » compliquerait la formule. La première expérience, elle, est constituée de plusieurs objets et serait pratiquement impossible à simuler de manière continue, étant donné que c’est un système chaotique (une petite variation des paramètres de départ mènera à une situation finale totalement différente).

Il a donc été décidé que les 2 expériences seraient simulées de manière discrète.

## Création d’un installateur pour l’application

<à faire>

# Planification initiale

## Liste de tâches

Une fois l’analyse préliminaire du projet terminée, il a été possible de commencer la planification initiale. Elle consiste en la création d’une liste de tâches la plus exhaustive possible et en la répartition du temps du projet entre les différentes tâches.

Ci-dessous, vous trouverez la liste des tâches prévues pour le projet ainsi que le temps en heures prévu pour chacune. Certaines d’entre-elles sont déjà terminées puisque la planification initiale a été créée lors de l’analyse et que certaines de ces tâches concernent justement l’analyse.

|  |  |
| --- | --- |
| **Tâches - objectifs** | **Temps prévu** |
|
| Absences - imprévus | 7h10m |
|
| Réunion avec experts ou chef de projet | - |
|
| DOC - Journal de travail | 2h |
|
| DOC - Planification initiale | 7h |
|
| DOC - Analyse technologies à utiliser | 30m |
|
| DOC - Analyse méthode de simulation | 2h15 |
|
| DOC - Analyse gestion des collisions | 6h10 |
|
| DOC - Analyse et choix méthodologie de projet | 30m |
|
| DOC - Analyse et conception des maquettes | 3h05m |
|
| DOC - Réalisation | 9h15m |
|
| DOC - Analyse calcul collisions élastiques avec masse | 3h55m |
|
| DOC - Analyse calcul trajectoire balistique avec vent | 3h05m |
|
| DOC - Expliquer mise en place de l'environnement et prérequis | 45m |
|
| DOC - Conception des tests | 3h05m |
|
| DOC - Divers (rapport) | 1h25m |
|
| DOC - Préparer le rendu du projet | 1h |
|
| DEV - Mise en place de l'environnement | 30m |
|
| DEV - Implémenter un widget "bouton" | 2h10m |
|
| DEV - Implémenter le menu principal | 3h5m |
|
| DEV - Implémentation du lanceur de billes | 2h30m |
|
| DEV - Afficher la "trace" des billes | 30m |
|
| DEV - Implémentation des collisions avec les bords | 2h35m |
|
| DEV - Implémenter configuration des propriétés des billes 1-4 | 1h50m |
|
| DEV - Résoudre collisions élastiques avec masse entre deux billes | 5h35m |
|
| DEV - Simulation expérience "chocs" | 1h15m |
|
| DEV - Simulation expérience "balistique" | 4h30m |
|
| DEV - Implémentation de la simulation balistique sans le vent | 1h40m |
|
| DEV - Prendre en compte le vent dans la trajectoire balistique | 3h05m |
|
| DEV - Détecter les collisions avec le "récipient" | 3h05m |
|
| DEV - Création de l'installateur (setup) | 3h05m |
|
| DEV - Divers | - |
|
| TEST - Validation des tests | 3h55m |
|

## Commentaires

Tout d’abord, il est important de préciser que cette liste de tâches est commune au document de planification et au journal de travail. Ils font tout deux référence à cette même liste. Certaines tâches n’ont pas de temps planifié car elles n’auront lieu d’être que dans le journal de travail. Par exemple, la tâche « réunion avec experts ou chef de projet » ne peut pas être planifiée. On peut connaître l’heure et le date des visites mais il est difficile de planifier le temps qu’elles prendront. Pour la même raison, certaines tâches sont volontairement « vagues », telles que la tâche « DEV – Divers ». Ces tâches sont là pour pouvoir être référencées par le journal de travail lorsque le travail effectué n’entre dans aucune autre catégorie.

Vous aurez peut-être également remarqué que certaines tâches n’ont pas de temps attribué. Cela est dû au fait que certaines tâches étaient déjà terminées au moment de rendre la planification initiale. Je n’allais pas rétro-planifier des tâches déjà terminées (ce serait absurde) mais je les ai quand même ajoutées à ma planification pour pouvoir les référencer dans mon journal de travail.

Une absence est prévue le jeudi 6 mai 2021. C’est pour cela que la tâche « Absences – imprévus » a 7h10 de prévu.

### Tâches principales

L’un des points les plus difficiles du projet sera la gestion des collisions. La création d’un moteur physique permettant la gestion des collisions ne sera pas simple. La conception des formules sera complexe car il ne s’agit pas simplement de créer un résultat visuellement agréable mais bien de créer un résultat le plus réaliste possible. Il faudra donc concevoir ces formules en partant de lois et concepts physiques élémentaires (tels que le principe de conservation d’énergie et de quantité de mouvement).

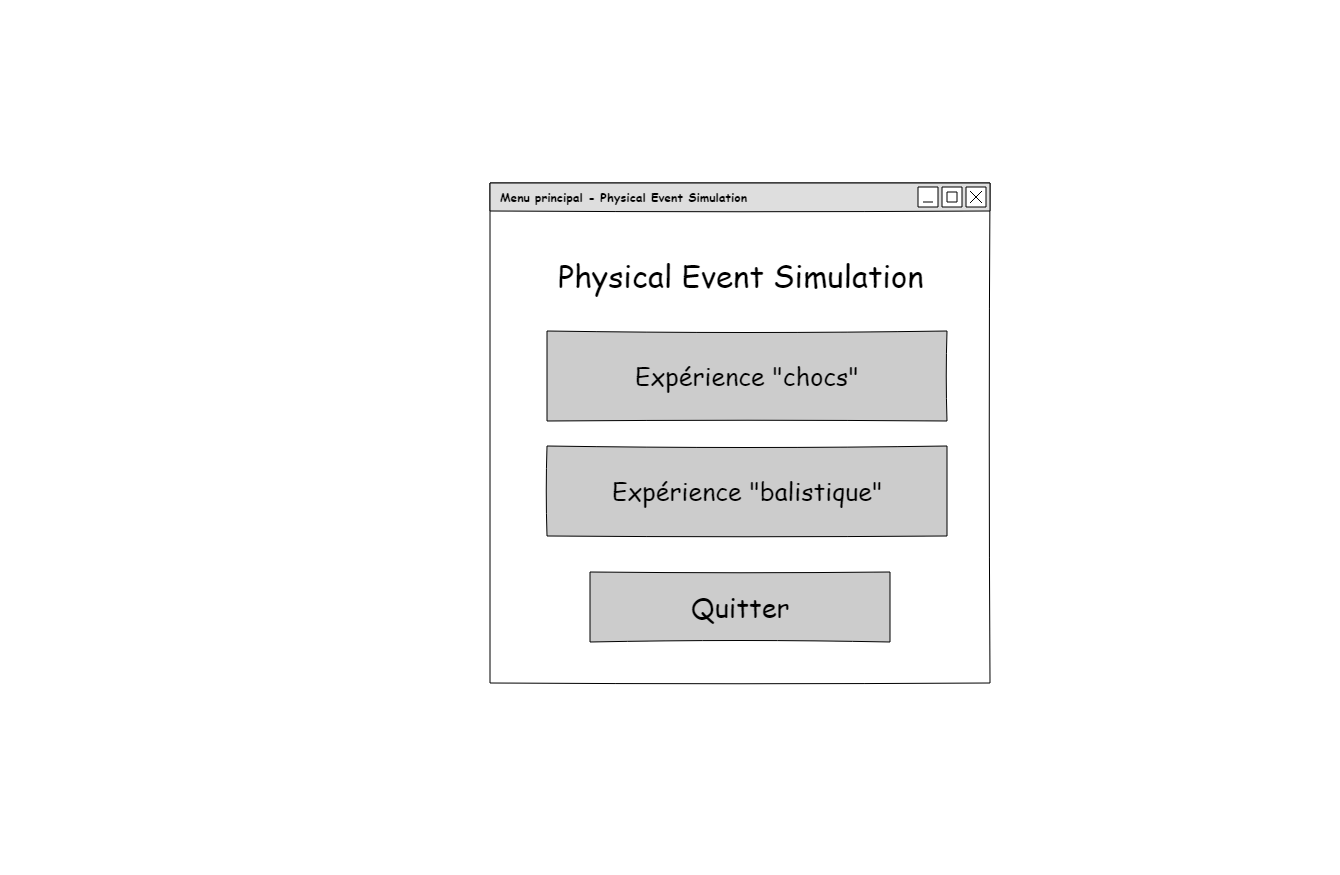
L’autre point qui prendra beaucoup de temps est la rédaction du rapport. Que ce soit pour mettre au propre l’analyse ou documenter le détail de la réalisation, la documentation du projet prendra beaucoup de temps. Ce choix est volontaire : il faut partir du principe que le projet va potentiellement être repris par un tiers un jour. Même si la documentation prend du temps, elle permet en réalité d’en gagner beaucoup en cas de reprise du projet.

## Maquettes de l’interface

Étant donné la nature de l’application à développer (une simulation), un soin particulier devra être apporté au visuel. Le but est de développer une interface simple et agréable à utiliser. Cependant, la bibliothèque graphique utilisée (SFML) est d’assez bas-niveau et c’est pour cela que j’ai choisi de partir sur un design « minimaliste ». Ci-dessous, vous trouverez les maquettes que j’ai conçues pour l’application.

### Menu principal

Le menu principal est très simple et ne contient qu’un titre avec 2 boutons permettant d’accéder aux différentes expériences et un dernier bouton pour quitter l’application.

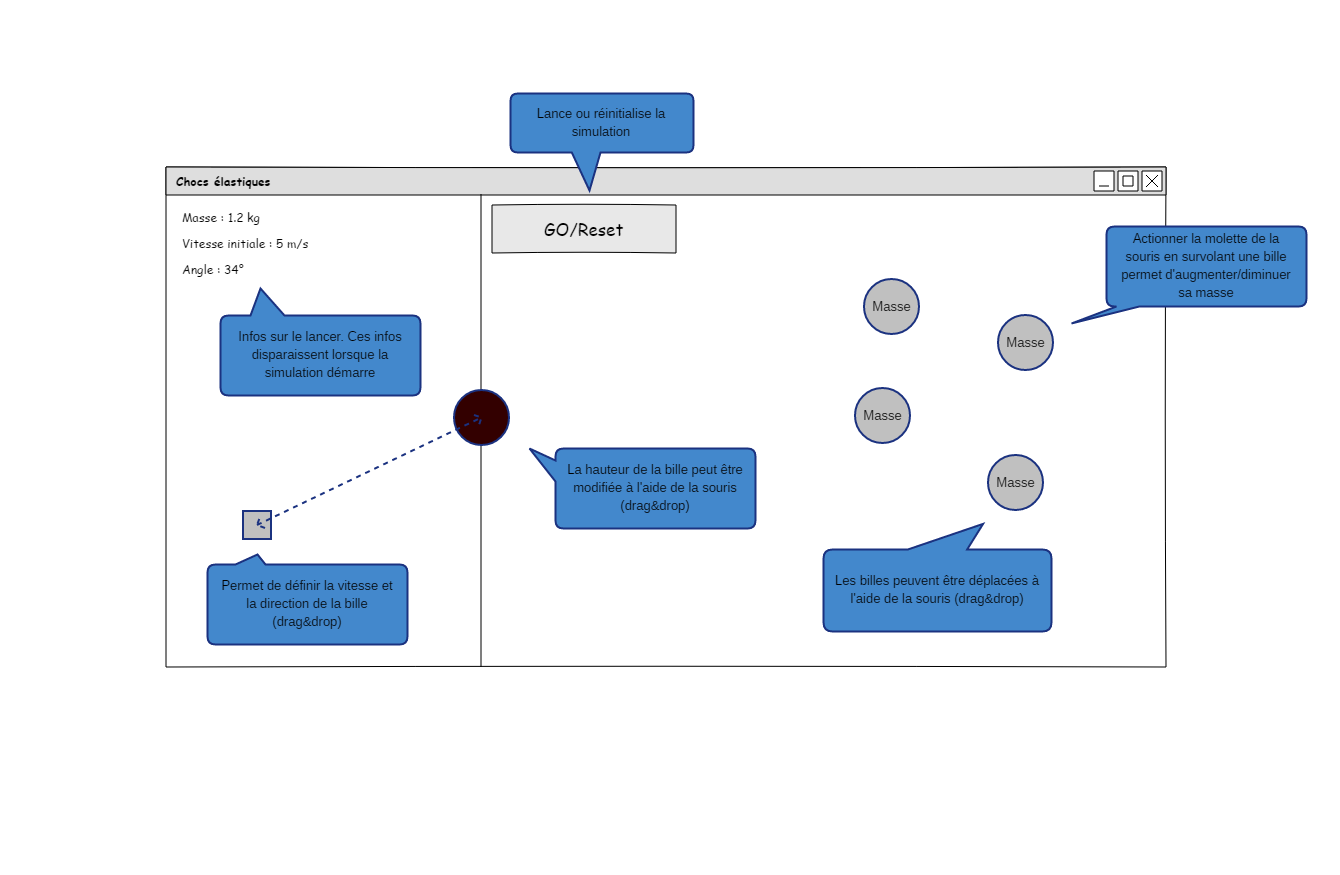


### Interface de la première expérience : « chocs »

Pour permettre de définir la vitesse et la direction de la bille principale, j’ai créé un « lanceur » (le petit carré connecté à la bille sombre). La vitesse de départ ainsi que la direction seront défini en calculant la distance relative séparant la bille et le carré.

Les billes pourront être déplacées en les glissant avec la souris (drag&drop). Il va de soi que la bille principale ne bourra bouger que verticalement et que les billes ne pourront pas se trouver en dehors de l’écran ou derrière le lanceur.

La masse des billes pourra être définie en actionnant la molette de la souris en les survolant (glisser vers l’avant pour augmenter la masse et vers l’arrière pour la diminuer). La masse exacte de chaque bille sera affichée dessus.

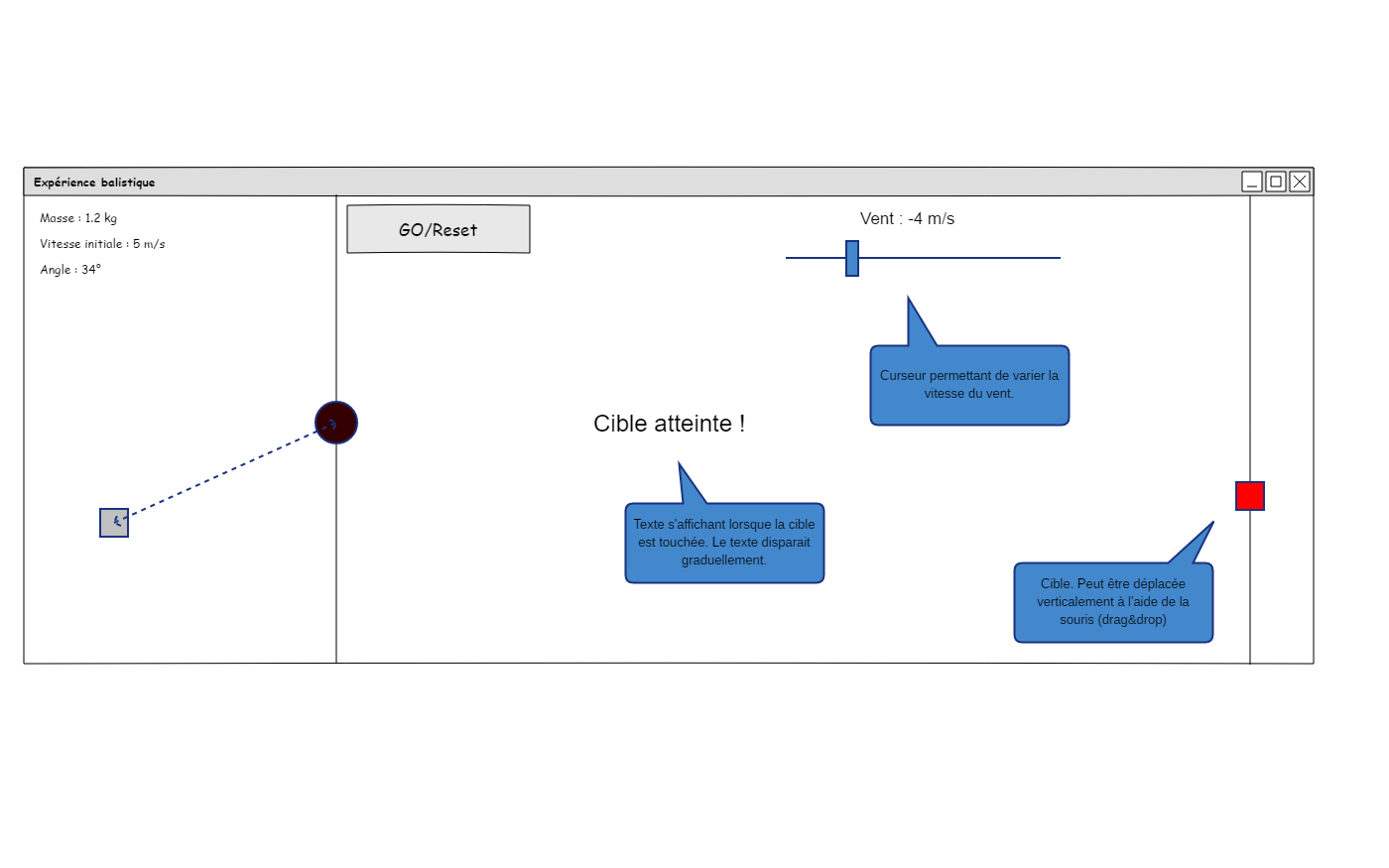


### Interface de la deuxième expérience « balistique »

La configuration de la bille de lancer est identique à l’autre expérience. La cible est positionnée de la même façon que la bille (en la glissant à l’aide de la souris).

Il est possible de modifier l’intensité et la direction du vent à l’aide du curseur. Une valeur négative indiquera un vent allant « contre » la bille et une valeur positive correspondra à un vent « poussant » la balle vers la droite.

Si la bille atteint la cible (collision), un message s’affichera au centre de l’écran. Le message disparaîtra graduellement (fade out).



## Formules mathématiques pour la gestion des collisions

La gestion des collisions sera une partie essentielle au fonctionnement de l’application. Dans une simulation de physique, on s’attend à ce que le résultat ressemble le plus possible à ce qu’on pourrait observer dans le monde réel, il faudra donc apporter un soin particulier à cette partie pour que le résultat paraisse le plus naturel possible.

Avant d’aller plus loin, il est important de bien expliquer en quoi consiste la « gestion des collisions ». Elle se divise en deux parties : la détection des collisions et la résolution des collisions.

La détection consiste à analyser la position, la forme et la taille de deux objets et à déterminer s’ils se superposent ou non. De plus, la détection des collisions englobe également le calcul d’informations telles que la normale de collision ainsi que la profondeur de la collision (nous reviendrons sur ces deux éléments).

Un état de collision est un état non désirable et c’est surtout un état abstrait qui ne peut exister que dans une simulation informatique. Dans la vraie vie, les objets ne peuvent pas occuper le même espace, une superposition est donc impossible. Pour que la simulation apparaisse réaliste, il est donc nécessaire de réarranger les objets pour qu’ils ne soient plus en collision. Ce processus s’appelle la résolution de collisions.

La résolution de collision consiste à déplacer et/ou appliquer une impulsion aux objets en état de collision dans le but qu’ils ne se collisionnent plus lors de la prochaine vérification.

### Détection et résolution d’une collision entre 2 billes

Nous allons voir comment détecter et résoudre une collision élastique entre 2 billes de masses différentes. Une collision élastique est une collision ou l’énergie cinétique totale des 2 billes est conservée et qui implique donc l’application d’une impulsion aux deux billes lors de la collision.

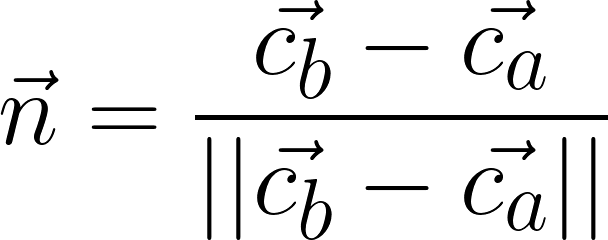
#### Détection de la collision

Nous sommes en 2D donc les billes sont analogues à des cercles. Les collisions entre deux cercles sont très simples à vérifier : il suffit de vérifier si la distance entre les centres des cercles est inférieure à la somme de leurs rayons. Si c’est le cas, les deux cercles se superposent. Pour simplifier et retirer la racine carrée (très couteuse au niveau du CPU), on peut élever au carré :

collisions billes

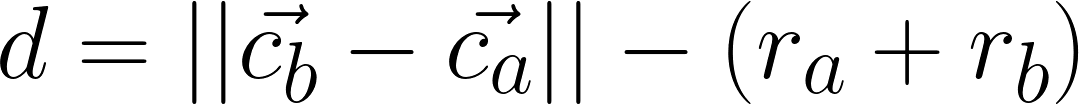
S’il n’y pas de collision, il va de soi que le programme n’ira pas plus loin. En revanche, si une collision est détectée, il va être nécessaire de déterminer deux informations essentielles : La normale de collision et la profondeur de collision.

La normale de collision correspond à l’axe le long duquel la collision se produit. D’une certaine manière, elle représente la « direction » de la collision. C’est une droite mais on la représente généralement sous forme d’un vecteur unitaire. Pour calculer ce vecteur unitaire, il faut diviser la position relative des deux cercles par la distance les séparant :



On a maintenant une normale de collision pointée en direction du cercle Ca. Elle indique la direction vers laquelle le cercle Ca doit être « poussé » pour résoudre la collision. Pour le cercle Ca, on prendra l’opposé de la normale.

La profondeur de la collision d peut être calculée en soustrayant la distance entre les centres des deux cercles par la somme de leurs rayons :



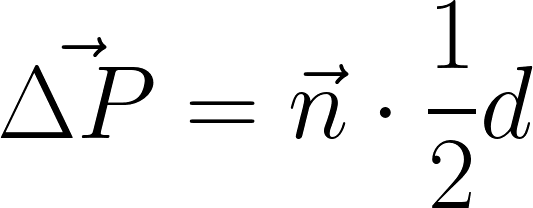
#### Résolution de la collision : déplacement

On dispose maintenant de la normale de collision et de la profondeur de la superposition des deux cercles. Il est temps de passer à la résolution. La première étape de la résolution consiste à repositionner les cercles pour qu’ils ne se superposent plus.

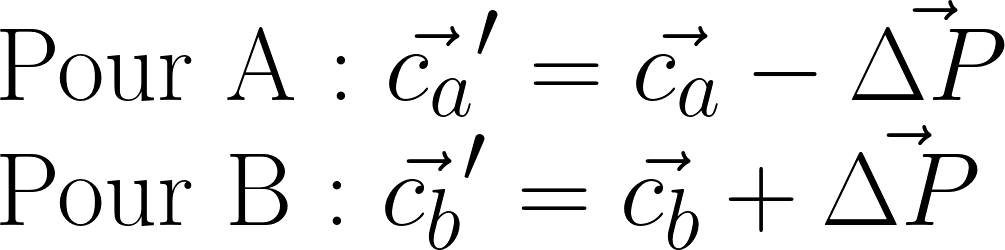
Pour cela, il faut calculer le vecteur de translation qui sera additionné ou soustrait en fonction du cercle concerné. Les propriétés de ce vecteur seront identiques pour les deux cercles, à une différence près : son sens. Pour repositionner le cercle Ca, il faudra soustraire le vecteur de déplacement à sa position alors que pour Cb, il faudra l’additionner. Le calcul de la nouvelle position après un déplacement se fait de la manière suivante :

C:\Users\luccharbonnier\AppData\Local\Microsoft\Windows\INetCache\Content.Word\déplacement.png

Pour calculer le déplacement (delta P), on amplifie la normale de collision par la moitié de la profondeur de la collision (d) :



On prend la moitié car ce vecteur sera appliqué pour les deux cercles, ce qui donnera un déplacement total égal à la profondeur. On peut maintenant calculer les nouvelles positions des cercles A et B :

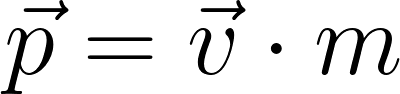


#### Résolution de la collision : impulsion

Lorsque deux billes se collisionnent, elles ne sont pas juste déplacées, elles rebondissent l’une contre l’autre. Il y a donc un changement au niveau de leur vitesse et direction de déplacement.

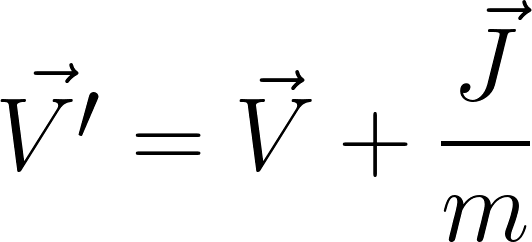
C’est ici que les choses se compliquent un peu. En plus de prendre en compte la normale de collision ainsi que la position des cercles, il va également falloir tenir compte de leurs vélocités et de leurs masses pour pouvoir leur appliquer une impulsion. Pour rappel, la vélocité est un vecteur représentant la vitesse et la direction de déplacement d’un objet.

Avant d’aller plus loin, il est nécessaire d’introduire le concept d’impulsion. Une impulsion est définie comme un changement de quantité de mouvement instantané. La quantité de mouvement d’un corps correspond à sa vélocité (un vecteur) amplifié par sa masse :



Dans la suite de cette réflexion, nous noterons l’impulsion avec la lettre J.

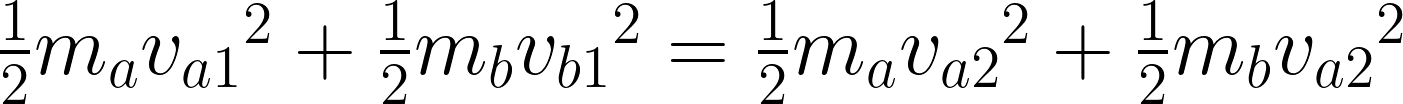
À partir de cette formule, on peut montrer que la vélocité finale d’un corps après l’application d’une impulsion J peut être calculée de la manière suivante (où v et v’ sont respectivement les vélocités de départ et de fin et et m correspond à la masse) :



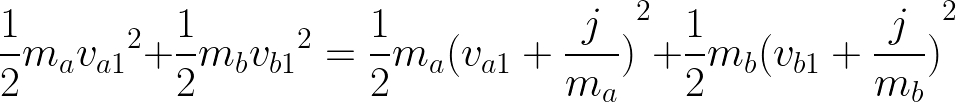
Selon la troisième loi de Newton, si un objet exerce une force sur un deuxième objet, ce dernier exercera une force de **direction opposée** mais de **grandeur égale** sur le premier objet. Concrètement, cela veut dire que les deux billes recevront la même impulsion mais dans des sens différents.

La direction dans laquelle l’impulsion sera appliquée est connue, c’est la normale de collision. Comme pour le vecteur de déplacement, on l’inversera ou non en fonction de la bille concernée. Étant donné que l’on connait la direction du vecteur, il ne nous reste qu’à calculer le scalaire qui servira à amplifier ce vecteur. Nous noterons ce scalaire j.

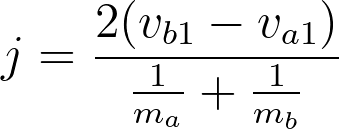
Pour déterminer j, nous allons appliquer le principe de conservation d’énergie qui nous dit que l’énergie initiale totale du système est égale à l’énergie finale totale du système. En d’autres termes, il n’y a aucune perte d’énergie. Étant donné qu’il n’y a que de l’énergie cinétique dans notre système, on peut poser l’équation suivante :



Elle montre simplement que l’énergie cinétique totale du système est conservée. Ici, va1, va2, vb1 et vb2 correspondent respectivement aux vitesses de départ et de fin des deux billes. Comme vu précédemment, la vitesse de fin peut être calculée à partir de la vitesse de départ si on connait l’impulsion et la masse de l’objet. Nous allons remplacer les deux vitesses finales (va2 et vb2) par cette équation :

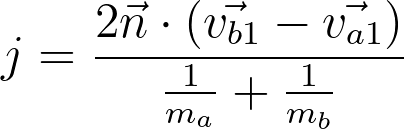


Vue comme ça, l’équation peut paraître compliquée mais si l’on isole notre inconnue j, elle se simplifie énormément :

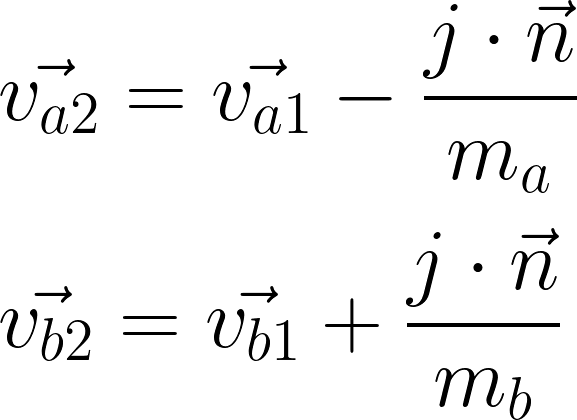


C’est presque fini, mais il reste quelque chose de très important à clarifier. Vous aurez sûrement remarqué que nous avons utilisé les vitesses et non les vélocités des billes dans l’équation. En fait, cette équation permet de résoudre des collisions en 1 dimension, c’est-à-dire, avec des billes qui vont exactement l’une contre l’autre. Cependant, ce n’est pas toujours le cas en 2 dimensions et il va falloir tenir compte de cela. Le calcul et l’application de l’impulsion ne changent pas, seul les paramètres changent. Au lieu de prendre les vitesses absolues des deux billes, nous allons prendre leur **vitesse le long de la normale de collision**. Pour calculer cette vitesse, il faut simplement effectuer le produit scalaire entre la normale de collision et la vélocité.

Si l’on substitue nos deux paramètres Va1 et Vb1 par cette nouvelle formulation, on obtient l’équation finale suivante :



On peut maintenant calculer le scalaire j de l’impulsion, il ne reste plus qu’à l’appliquer aux vélocités des deux billes :



Après le repositionnement des deux billes et l’application de l’impulsion, la collision peut être considérée comme résolue.

# Réalisation

À faire

## Mise en place de l’environnement pour compiler le code

À faire : link sfml

# Tests

À faire

# Conclusion

# Bibliographie

|  |  |
| --- | --- |
| [1] | «Simulation à événements discrets,» [En ligne]. Available: https://fr.wikipedia.org/wiki/Simulation\_%C3%A0\_%C3%A9v%C3%A9nements\_discrets. |
| [2] | «Simulation informatique,» [En ligne]. Available: https://fr.wikipedia.org/wiki/Simulation\_informatique. |