Physical Event Simulation

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

Table des matières

[Analyse 1](#_Toc70577573)

[Méthodologie de projet 1](#_Toc70577574)

[Technologies à utiliser 1](#_Toc70577575)

[Langage de programmation 1](#_Toc70577576)

[Bibliothèques 1](#_Toc70577577)

[Méthode de simulation : discrète ou continue 1](#_Toc70577578)

[Simulation à événements discrets 1](#_Toc70577579)

[Simulation continue 2](#_Toc70577580)

[Choix 2](#_Toc70577581)

[Gestion des collisions 2](#_Toc70577582)

[Interface graphique 2](#_Toc70577583)

[Menu principal 2](#_Toc70577584)

[Lanceur de projectiles 2](#_Toc70577585)

[Interface de la première expérience : « chocs » 2](#_Toc70577586)

[Interface de la deuxième expérience « ballistique » 2](#_Toc70577587)

[Planification initiale 3](#_Toc70577588)

[Réalisation 3](#_Toc70577589)

[Tests 3](#_Toc70577590)

[Conclusion 3](#_Toc70577591)

[Bibliographie 3](#_Toc70577592)

# 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>.

## 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

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.

## Gestion des collisions

Schémas fonctionnement

Que des billes donc simple

## Convention de code/nommage

La convention de l’ETML n’est prévue que pour le C#, pas le C++. <détailler>

## Interface graphique

Étant donné la nature de l’application à développer (une simulation), un soin particulier devra être apporté au visuel. Il s’agira donc de créer une interface qui donnera envie d’utiliser le programme. Cette partie pourrait prendre du temps vu que la bibliothèque graphique utilisée est d’assez bas niveau.

S’il s’avère nécessaire de mettre en place des widgets (boutons, checkboxes, curseurs, etc…) il sera nécessaire de les implémenter soi-même.

### Menu principal

À faire

### Lanceur de projectiles

À faire

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

À faire

### Interface de la deuxième expérience « ballistique »

À faire

# Planification initiale

screenshot

# 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. |