h e p i a

Haute école du paysage, d'ingénierie
et d'architecture de Genève

Jeu de tir de canon

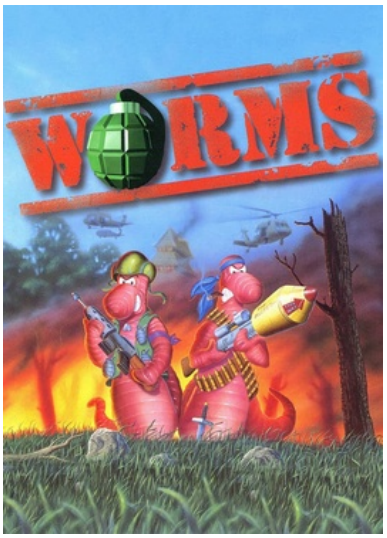
Orestis Malaspinas, Jonathan Lemus

2026

1 Introduction

La Physique est une science qui étudie par l'expérimentation et l'élaboration de concepts les propriétés fondamentales de la matière et de l'espace-temps. Ce n'est donc pas qu'une discipline permettant de construire des accélérateurs de particules, des bombes nucléaires ou juste de briller en société. Les concepts de la mécanique, de l'optique ou encore de l'électromagnétisme fournissent les briques de base afin de simuler numériquement des phénomènes souvent inaccessibles en laboratoire. Sans la Physique, pas de *Rocket League*, *Dragon Ball FighterZ* ou encore *Fortnite*. En effet, ces jeux vidéos tournent sous le moteur Unreal Engine, principal concurrent de Unity ou encore CryEngine. Ces moteurs de jeu vidéo, aussi puissants soient-ils, permettent une simulation de plus en plus fidèle à la réalité, grâce à l'intégration de principes physiques. A titre d'exemple, la mécanique permet de gérer les notions de mouvements alors que l'optique géométrique se concentrera sur les effets de *ray tracing*.

Sans prétention de créer un véritable moteur de jeu, l'objectif des travaux qui seront effectués durant l'année est d'appréhender des concepts physiques simples en les implémentant en C.



2 Tir ballistique

2.1 Votre mission

Vous rejoignez l'équipe technique d'un studio qui développe un jeu de tir d'artillerie façon *Worms*: un canon fixe doit toucher une cible posée quelque part sur le terrain. Votre mandat est d'écrire, en C, le module physique qui calcule et anime la trajectoire du boulet, de la sortie du canon jusqu'à l'impact.

Question directrice: connaissant uniquement la vitesse d'éjection du canon v_0 et la position de la cible, comment déterminer l'angle de tir qui permet de l'atteindre, puis animer image par image la trajectoire du projectile jusqu'à l'impact?

Le mandat se décompose en cinq étapes, à traiter dans l'ordre:

1. poser les hypothèses physiques et le modèle géométrique du problème;
2. établir les équations du mouvement du boulet;
3. en déduire l'angle de tir permettant d'atteindre la cible;
4. implémenter et animer le tir en C;
5. valider la trajectoire obtenue avec une seconde méthode de calcul.

2.2 Livrable et critères de réussite

À la fin de ce projet, vous devez remettre un programme C (basé sur le squelette fourni) qui, une fois compilé avec le `Makefile` donné:

- ouvre une fenêtre *gfx* et y dessine, avec l'API *gfx*, le canon, la cible et la trajectoire du boulet en temps réel;
- calcule automatiquement l'angle de tir à partir de v_0 et de la position de la cible;
- fait avancer le boulet à chaque pas de temps Δt selon les équations du mouvement, en travaillant en unités SI puis en convertissant en pixels uniquement pour l'affichage;
- détecte que la cible a été touchée et annonce si le tir est réussi ou manqué;
- recalcule la trajectoire avec la formule de Verlet et vérifie qu'elle coïncide avec la trajectoire analytique;
- reste lisible: constantes nommées, code commenté, structure proche du squelette fourni.

2.3 Étape 1: Contexte, hypothèses et modèle géométrique

Un tir de canon propulse un objet vers une cible distante à atteindre (pour la détruire… potentiellement…). L'objet est mis en mouvement par une explosion dans le canon, ce qui lui confère donc une vitesse et un angle initiaux. Une fois dans les airs, le projectile est soumis à la gravité et aux frottements de l'air.

Avant de continuer, posez-vous les questions suivantes (elles conditionnent tout le reste du projet):

- Quelles forces agissent réellement sur le boulet une fois qu'il a quitté le canon?
- Que se passe-t-il si on néglige les frottements de l'air? Est-ce raisonnable pour une première version du modèle?
- Dans quel repère allez-vous décrire le mouvement? Combien de dimensions vous faut-il?

Pour ce premier projet, on se placera en deux dimensions dans le plan (x, y) et les frottements de l'air seront négligés par souci de simplicité. Seule l'action de la gravité a un effet sur le boulet.

Modèle géométrique du boulet et de la cible. Le boulet n'est pas tout à fait ponctuel: on le modélise par sa position (x, y) (son centre) munie d'un rayon r_b. La cible, fixe, est modélisée de la même façon: un centre (x_{target}, y_{target}) muni d'un rayon r_t. Avec ce modèle, la cible est touchée dès que les deux disques se chevauchent, c'est-à-dire dès que la distance entre les deux centres devient inférieure ou égale à la somme des deux rayons:

$$(x - x_{target})^2 + (y - y_{target})^2 \leq (r_b + r_t)^2.$$

C'est cette condition que vous devrez tester à chaque pas de temps dans votre programme pour savoir si le tir est réussi.

2.4 Étape 2: Mettre le mouvement en équations

Dans le cas où on néglige les frottements de l'air et en intégrant la seconde loi de Newton, il est facile de déduire les équations de la trajectoire du boulet de canon.

En 2 dimensions, ces équations horaires donnent les positions verticale (hauteur) *y(t)* et horizontale *x(t)* (distance par rapport au canon) du projectile en fonction du temps écoulé par rapport au départ du tir. Elles sont données par:

$$y(t) = -\frac{1}{2}gt^2 + \sin(\alpha)v_0 t + h$$
$$x(t) = \cos(\alpha)v_0 t$$

avec dans cette équation:

- $y(t)$ la hauteur du projectile en mètres
- $x(t)$ la distance horizontale par rapport au canon en mètres
- t le temps écoulé par rapport au début du tir en secondes
- g l'accélération de la pesanteur en m/s^2
- α l'angle de tir en radians
- v_0 la vitesse d'éjection en m/s
- h la hauteur du canon en mètres

2.5 Étape 3: Trouver l'angle de tir

Admettons que vous êtes des artilleurs chevronnés et que vous maîtrisez parfaitement la puissance du canon, il est donc très probable que vous connaissiez la vitesse d'éjection v_0. Cependant, pour une cible placée à une distance d donnée, quelle est l'angle α optimal à donner au canon pour l'atteindre?

Essayez de répondre avant de lire la formule ci-dessous, à l'aide de la propriété de symétrie de l'équation de la trajectoire (indice: à quel instant le boulet retombe-t-il à la hauteur h de départ?). On trouve:

$$\alpha = \frac{1}{2}\arcsin\left(\frac{gd}{v_0^2}\right).$$

(On rappelle quand même que d peut se calculer comme la différence entre la position sur l'axe horizontal de la cible $p_{x,target}$ et celle du canon x_0)

À ce stade, vous avez tout ce qu'il faut pour calculer, sur papier, une trajectoire complète pour des valeurs numériques données de v_0, d et h. Avant de passer au code, vérifiez votre formule à la main sur un petit exemple: c'est le meilleur moyen de repérer une erreur de signe ou d'unité avant qu'elle ne se cache dans 200 lignes de C.

2.6 Étape 4: Implémenter en C

Votre travail maintenant consiste à utiliser ces connaissances en mécanique pour développer une petite application en C permettant de tirer avec un canon pour atteindre une cible.

Pour mettre en place une telle simulation, il faut procéder de manière itérative. En effet, en simulation numérique, le temps n'est pas continu au sens mathématique du terme. Le temps écoulé à partir du début du tir se calcule de façon incrémentale avec un "pas de temps" Δt. Ainsi par exemple, la première itération que l'on peut calculer après le tir est donnée par $t_1 = t_0 + \Delta t$, celle d'après est donnée par $t_2 = t_1 + \Delta t$ et ainsi de suite.

Vous avez à disposition un squelette de code dans le fichier *main.c* à partir duquel vous devrez implémenter les parties manquantes:

- définissez les constantes et variables utilisées pour les équations (positions et vitesse initiales, gravité, pas de temps, rayons du boulet et de la cible…);
- écrivez la boucle qui calcule, à chaque itération, la position du boulet à partir des équations de l'étape 2 et de l'angle de l'étape 3, puis teste la condition de touché de l'étape 1;
- affichez la scène à l'aide de l'API *gfx* (voir *gfx/gfx.h* et *gfx/gfx.c*): à vous de choisir comment dessiner le canon, la cible et la trajectoire avec les fonctions mises à disposition.

Remarque: Les calculs que vous effectuerez se font dans les unités du système international (mètre, seconde, kilogramme…). Pour afficher vos résultats dans la fenêtre *gfx*, il faudra procéder à une conversion permettant de passer des mètres aux positions de pixels.

2.7 Étape 5: Retrouve-t-on la trajectoire balistique avec Verlet?

Une fois votre canon capable de toucher la cible avec les équations analytiques de l'étape 2, vous allez calculer la même trajectoire d'une seconde façon, purement numérique cette fois: l'**intégration de Verlet**. C'est une méthode que nous détaillerons en profondeur dans un prochain chapitre, mais en voici déjà la formule pour le cas sans frottement, qu'on vous donne telle quelle:

$$x_{n+1} = 2x_n - x_{n-1} + a_x \Delta t^2, \qquad y_{n+1} = 2y_n - y_{n-1} + a_y \Delta t^2$$

avec $a_x = 0$ et $a_y = -g$. Cette formule a besoin de deux positions précédentes pour démarrer; on amorce donc le calcul avec:

$$x_1 = x_0 + v_{x,0}\Delta t + \frac{1}{2} a_x \Delta t^2, \qquad y_1 = y_0 + v_{y,0}\Delta t + \frac{1}{2} a_y \Delta t^2.$$

Ajoutez ce second mode de calcul dans votre programme (une simple bascule entre les deux méthodes suffit) et vérifiez que la trajectoire obtenue par Verlet se superpose, aux erreurs d'arrondi près, à celle obtenue avec les équations analytiques de l'étape 2. Si ce n'est pas le cas, c'est probablement le signe d'une erreur dans l'une des deux implémentations: c'est exactement le rôle d'une étape de validation.

Cette vérification n'est pas un simple exercice de style: dès que l'on ajoute des frottements fluides (comme l'air), il n'existe plus d'équation analytique de la trajectoire, et l'intégration de Verlet devient la seule méthode disponible. Le fait de savoir qu'elle reproduit correctement le cas sans frottement est ce qui vous permettra de lui faire confiance au prochain chapitre.

2.8 Pour aller plus loin (bonus)

Si votre canon touche déjà sa cible à tous les coups et que la validation de l'étape 5 passe, voici de quoi continuer à creuser:

- Faites varier la position de la cible ou v_0 au clavier et vérifiez que l'angle recalculé touche toujours la cible.
- Affichez plusieurs trajectoires (plusieurs angles) simultanément pour visualiser la parabole de sécurité.