# Rapport de Projet : PFA
# Paulhiac Kaiji, Leroy Antonin
# 2024-2025

## Vue générale

Le rythme générale du développement consister à discuter de quoi faire pendant les séances de PFA organisé par le professeur, et de s'y mettre à coder quand on on est arrivé chez soi.

Nous avons pendant très longtemps pas eu accès à une démo de ce qu'on programmer : cela n'était pas bon pour le moral, et on doutait fréquemment de ce qu'on écrivait.

L'objectif de notre projet n'était pas de créer un jeu vidéo avec un game-design intéressant, mais de créer un programme qui possède un code "intéressant" (L'aspect game-design du jeu a été même quasi-négligé).

Malheureusement, pour des raisons personnelles Leroy Antonin a eu très peu d' inputs sur le projet la grande majorité du temps (Anthonin a pus y mettre plus de travail vers la dernière semaine)  : la majorité du travail a été effectué par Paulhiac Kaiji.

Il y eut aussi quelques frustrations sur le niveau de qualité pauvre de la communication entre nous : Anthonin a souvent eu du mal à communiquer son avancée avec Kaiji.

Nous avions aussi créé un deuxième dépôt Git au lieu de mettre à jour le premier.

## Généralité sur la structure du code

Le projet est basé sur le TP3 se trouvant sur la page du professeur chargé du projet.

Le programme comporte en plus ces 3 systèmes : Système de dégât (“Destruction system), système de IA (“IA system”), système de physique naïve/très simple (“Physics system”)

Il comporte aussi quelques classes en plus, qui servent principalement à faire la démonstration de ces 3 systèmes : Les blocs, 1 classe d’ennemie possèdent une IA simpliste, 1 joueur, des balles à tirer pour le joueur, et 1 classe “viseur” qui permet au joueur de viser où tirer ses balles.

Plusieurs fichiers ont été retravaillés pour être adapter à ce qu’on veut faire : hormis le fichier cst.ml, input.ml et game.ml, global.ml a été assez lourdement retravaillé.

## Les 3 nouveaux systèmes

Nous allons discuter des 3 nouveaux systèmes :

- Physics
  Très simpliste (voir absurde), le système de Physics se contente juste de faire chuter avec une vitesse croissante toutes les instances inscrivant auprès d'elle. Elle utilise principalement 2 variables : position et masse.

- Destruction
  Ce système se base sur les concepts de "Hurttbox" et "Hitbox" dans les jeux vidéo : la Hurtbox est la zone dans laquelle il faut taper pour infliger des dégâts, et la Hitbox est la zone qui inflige des dégâts quand une Hurtbox entre en contact avec.

  Cependant, si jamais les points de vie chute à 0, ce n'est pas à ce système de gérer la "mort" de l'instance : c'est à d'autre système de le faire dans notre programme. Généralement, c'est au système d'IA de gérer ça.

  La Hitbox possède une valeur fixe "damage" qui indique combien de dégâts celle-ci inflige à une Hitbox quand elle rentre en contact avec elle, et respectivement Hitbox possède une variable "healthPoint" qui diminue quand elle se prend des dégâts.

  Chaque instance de Destruction possède obligatoirement une Hitbox et une Hurtbox.

  Les instances de Destruction possède aussi une liste "protectedTag" et une constante "tag" : une Hitbox ne peut pas infliger de dégât contre une Hurtbox possèdent un "tag" qui se trouve dans sa liste de "protectedTag"

  Le système possède aussi 2 variables "relativeHitbox" et "relativeHurtbox" qui ne sont pas utilisées : elles servent à décaler la position de la hitbox/hurtbox par rapport aux coordonnées de l'instance.

- IA
  C’est le système le plus complexe qu’on a ajouté : elle munit à chacune de ses instances une fonction “stateMachine” qui a pour objectif de faire exécuter une fonction propre à chaque classe de IA lors de l’exécution du système IA. Cette fonction se base sur la valeur d’une variable “behavior” qui est un entier qui indique à la stateMachine quelle block de code elle doit exécuter :

  Par exemple, un ennemi peut avoir 2 “behavior” distincts : Attaque et Dormir. Si behavior est à Attaque, alors stateMachine commandera à l'ennemie d’aller attaquer le joueur. Si behavior est à Dormir, alors stateMachine commandera à ce qui ne fait rien.

  Ensuite, le deuxième élément clé du système est que chaque instance de IA possède un attribut “id”, qui leur permet d’être identifiés de manière unique par le programme. Dans global.ml figure une liste d’instance de IA, cette liste permet notamment de retrouver certaines instances grâce à leurs “id”. “id” nous aide aussi à combler un défaut de la structure de notre code : on ne peut pas accéder directement aux champs des instances de classe même depuis leurs fichiers .ml associé. Par conséquent, on utilise leurs id pour rapidement les retrouver.

  Enfin, les instance de IA possèdent aussi une fonction “unregister” : c’est une fonction qui a pour objectif de donner à l’IA de s’auto détruire en retirant l’instance de chaque système

## Instances des systèmes

- Player
  Fonctionne sur le système Physics, Collision, Destruction, Move, Draw. Malgré qu'il possède un système de Destruction, le joueur est indestructible même si ses points de vies passent à zéro. Ses mouvements et son action de tir sont contrôlés par input.ml, et il peut sauter.

- Drone
  Fonctionne sur le système Destruction, Move, Draw, IA. Il possède une IA très basique : si le joueur est "loin", il reste statique. Sinon, il s'approche. Il peut être détruit et infliger des dégâts au joueur au contact. Il ne possède pas de système de Collision.

- Bullet
  Fonctionne sur le système Destruction, Move, Draw, IA. Possède une IA car c'est utile pour calculer les mouvements de la balles/quand la détruire. En l'occurrence, la balle se détruit quand elle tape dans une Hurtbox ennemie après lui avoir infligé des dégâts. Un timer global lui est associé pour éviter que le joueur puisse tirer sans délai son pistolet.

- Block
  Fonctionne sur Collision, Draw. Plateforme pour le joueur. Le joueur peut sauter dessus (ou rebondir).

- Crosshair
  Fonctionne sur Move, Draw. Permet au joueur de viser avec son arme à feu : la direction des balles est déterminée par le vecteur partant du joueur pointant la position du Crosshair.

## Fonctionnalités retiré

Un système de sauvegarde a été prévu et avait été développer : mais nous l'avons finalement abandonner pour 2 raisons :

- Comme on n'avait toujours pas de démo qui fonctionner jusqu'à la fin, on ne voulait pas implémenter des choses qui se base sur une fondation pas stable
- Avec la structure du code actuel, coder un système de sauvegarde est compliqué.

## Remarques

Le code pour créer les instances d'IA n'est pas forcément propre : il y a beaucoup de variables qui ne sont pas utiles dans le fichier.