PIN-2022, série 02, simulation & animation

Contents

Introduction

Nous introduisons ici la 2nde des 2 séries d'exercices, dont la réalisation est une condition préalable au projet, avec l'hypothèse que la série 1 a déjà été introduite et est au minimum en cours de réalisation.

NB : voir la donnée de la série 1 pour l'introduction des termes spécifiques de ce document, ainsi que pour une explication du contexte, de l'objectif des séries d'exercice, de l'organisation du cours et de l'évaluation des étudiants.

Série 2

L'objectif de la série 2 consiste d'abord à réaliser une application de setup qui génère les états successifs d'un monde (fichier timeline) sur la base d'un état de départ (fichier state). Il consiste ensuite à étendre l'application graphique de la série 1 pour que celle-ci affiche et anime les mondes ainsi générés.

Applications

Le schéma ci-après est une mise à jour de celui de la série 1. Il fait ressortir le fait que les applications de setup consomment également des fichiers JSON en entrée afin de générer les fichiers timelines :

- State : définit l'état de départ au temps 0 à partir duquel on va pouvoir calculer l'évolution du mode au fil du temps
- Constraint : impose des limites sur l'évolution dynamique du monde dans le temps

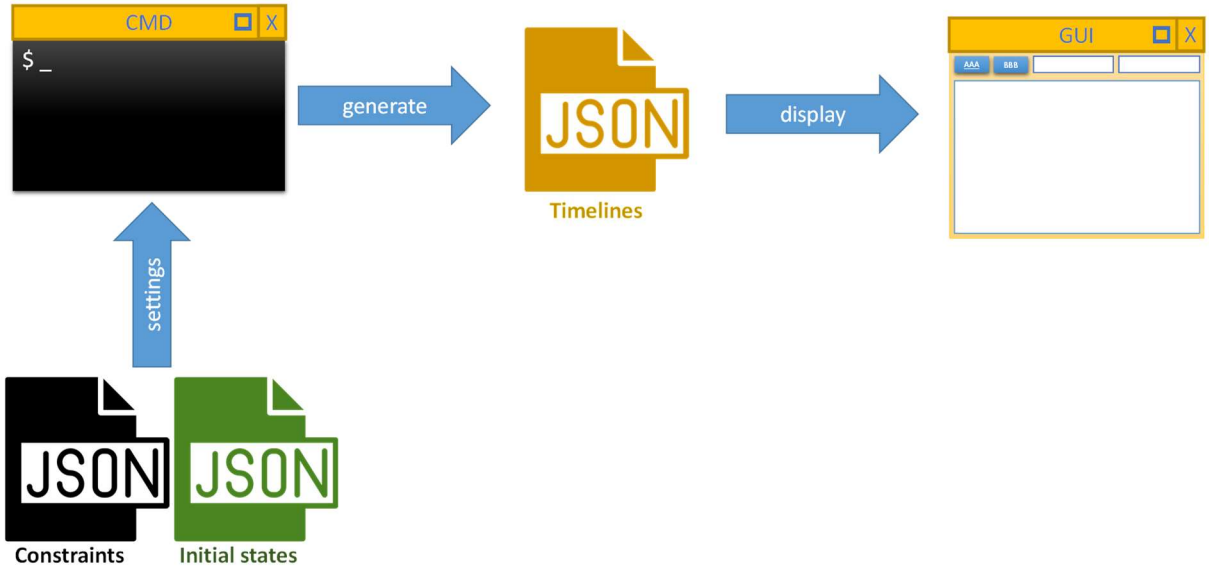


figure 1 Applications à réaliser et interfaces JSON

Objectif

Contrairement à la série 1 qui ne faisait qu'afficher les états du monde à un instant t, nous allons maintenant animer le monde de manière continue. Visuellement, la conséquence est que nous allons devoir calculer les déplacements des robots au rythme de rafraichissement de la zone d'affichage[1].

Nous allons aussi générer des timelines simples que nous pourrons ensuite visualiser. Nous allons, pour cela, calculer les états consécutifs aux explosions de particules que nous avons vu se produire lors de l'affichage dynamique des timelines dans la série 1. Nous allons également traiter les collisions entre robots, ou avec des particules.

NB : Des exemples de constraints, states et timelines vous sont fournis mais vous êtes encouragés à définir les vôtres également pour pouvoir tester plus à fond vos réalisations.

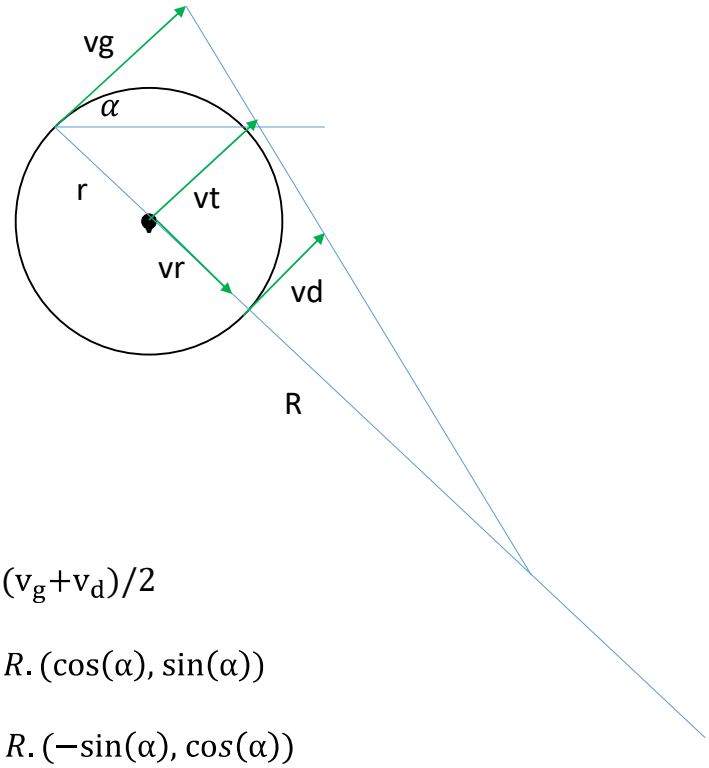
NB : D'autres événements sont visibles dans les timelines qui vous sont fournies. Il s'agit des modifications de trajectoire dans la chasse aux particules et des décontaminations (i.e. l'élimination des particules). Le traitement de ces événements est hors du scope de cette série.

[1] Voir à nouveau la vidéo montrant le déplacement d'un robot : https://youtu.be/LfpYF8NqJb8

Simulation des déplacements

Les trajectoires des robots sont des courbes qui suivent des arcs de cercle. L'orientation du robot suit la courbe, elle est constamment tangente à celle-ci. Pour arriver à ce résultat, il faut imaginer, comme illustré sur la figure 2, que les robots ont 2 vitesses, gauche v_g et droite v_d, qui sont des vitesses de déplacement linéaire des points extrêmes du diamètre transversal du cercle quand on se positionne dans le sens d'orientation du robot.

Quand les 2 vitesses sont égales, le déplacement est linéaire. La trajectoire s'incline du côté de la vitesse la plus faible, sinon, et dessine un arc de cercle dont le rayon R, par rapport au centre de l'objet en mouvement, selon la formule (1) sur la figure 2. L'objet va également tourner sur lui-même pour suivre la trajectoire selon une vitesse de rotation ω qui peut être calculée selon la formule (2). A noter, que l'objet va tourner sur lui-même dès lors que $v_g = -v_d$, les vitesses négatives étant autorisées. Au final, pour un intervalle de temps donné suffisamment court, on obtient la distance parcourue dans le monde virtuel selon la formule (3).



$$\frac{v_g}{v_d} = \frac{R + r}{R - r}$$

$$R.(v_g - v_d) = r.(v_g + v_d)$$

(1) $$R = r.\frac{v_g + v_d}{v_g - v_d}$$

(2) $$\omega = \frac{v_t}{R} = \frac{v_g - v_d}{2.r}$$

$$v_t = (v_g + v_d)/2$$

$$\overrightarrow{v_t} = R.(\cos(\alpha), \sin(\alpha))$$

$$\overrightarrow{v_r} = R.(-\sin(\alpha), \cos(\alpha))$$

(3) $$\vec{d} = \overrightarrow{v_t}.\sin(\omega.t) + \overrightarrow{v_r}.(1 - \cos(\omega.t))$$

figure 2 calcul des déplacements en arc de cercle

Simulation des explosions

Dans le monde que nous simulons, les explosions de particules se produisent de manière aléatoire. Une particule qui explose est remplacée par 4 sous-particules plus petites (décomposition) ou elle disparait tout simplement (désintégration).

La décomposition en 4 sous-particules est calculée comme illustré sur la figure 3. Chaque sous-particule occupe l'espace maximal possible en conservant toutes les 4 le même rayon, sans sortir du disque occupé par la particule parente[2].

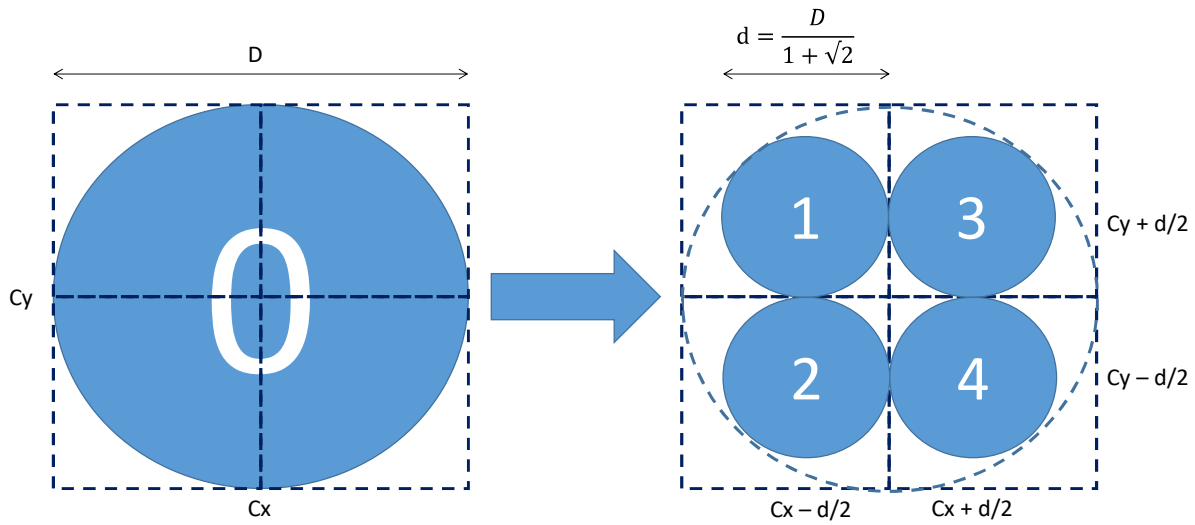


figure 3 décomposition en 4 sous-disques

Nous numérotons les particules séquentiellement dans leur ordre d'apparition, avec l'hypothèse qu'elles apparaissent systématiquement dans l'ordre indiqué sur la figure 3. Pour le reste, le rythme des explosions suit une loi stochastique.

Simulation des collisions

Les trajectoires des robots aboutissent en principe sur les particules, et elles peuvent avant cela se croiser entre elles (voir figure 4), provoquant dans les 2 cas des collisions. Dans notre simulation, cela se traduit par une mise à l'arrêt du ou des robots concernés à la périphérie de l'objet percuté. En d'autres termes les cercles dessinant la périphérie des objets doivent être tangeants et les vitesses des robots doivent être mises à 0.

Comme illustré en figure 4, nous allégerons le calcul des collisions en prévoyant une marge d'erreur paramétrable ε. Ce paramètre sera défini comme une constante du code source.

[2] https://debart.pagesperso-orange.fr/seconde/empilements_dans_plan.html

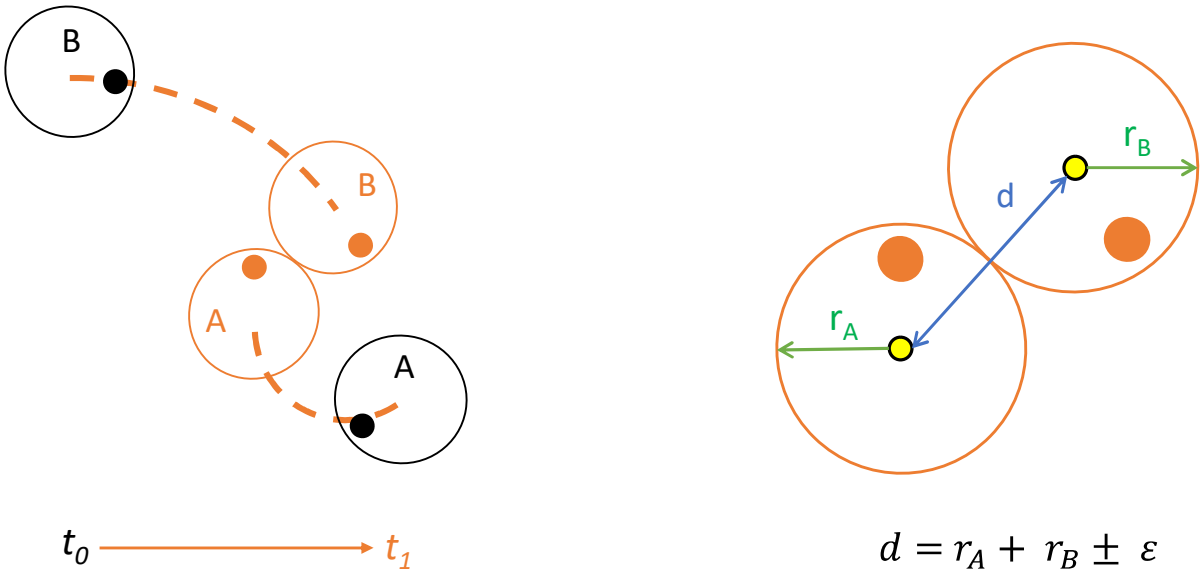


figure 4 collisions

Animation du monde

L'animation est conditionnée par le rythme de rafraichissement de l'écran. Ce dernier définit le nombre de rafraîchissements de l'image qu'il faut effectuer pour que l'œil humain perçoive un mouvement continu. Il n'est donc pas nécessaire d'aller au-delà, sous peine de saturer la machine avec des traitements inutiles.

Nous définissons un paramètre supplémentaire pour le taux de rafraîchissement sans aucun lien avec l'accélération de l'animation que nous avons prévu dans la série 1 avec les champ « Speed ». Ce paramètre sera défini comme une constante du code source et régit seul le rythme de rafraichissement de la zone d'affichage du monde des robots et particules. Il est fixé à 24 par défaut[3].

Directives

Vous trouverez dans le dossier */Explications* toutes les explications complémentaires (readme.md) et exemples (fichiers de données et images de rendu graphique) nécessaires.

En plus des explications déjà vues pour la série 1, vous aurez besoin du contenu des sous-dossiers suivants pour cette série :

- */Command* : ce dossier contient les informations sur les applications de setup et, en particulier, sur les contraintes à respecter. Les timelines générées doivent respecter ces contraintes.
- */Collisions* : ce dossier contient une explication détaillée de la recherche et du traitement des collisions.

[3] https://fr.wikipedia.org/wiki/Images_par_seconde

NB : les temps aléatoires auxquels les explosions de particules se produisent ont déjà été calculés dans les fichiers state qui vous ont fournis.

Livrables

Vous réaliserez votre travail dans un fork du GitHub du cours. L'accès à ce fork en lecture doit être ouvert aux membres de votre groupe et aux professeurs, mais fermé au reste des étudiants. **Le résultat final doit être taggé avec le nom de la série réalisée.**

Sources

Tous les sources **.cpp*, **.h*, **.ui*, ainsi que les directives de build (*CMakeLists.txt*) sont contenus dans le répertoire */Src*, lui-même structuré par application. **Le build doit être fonctionnel**.

Données de test

En plus des données fournies dans le dossier */Explications*, vous trouverez dans le dossier */Data*, les données de test pour vérifier la bonne implémentation de vos applications. Le dossier est structuré pour distinguer les séries 1 et 2, puis le projet et la compétition finale.

Directives

La série est valide lorsque vous pouvez démontrer le bon fonctionnement de votre réalisation sur les jeux de données fournis et que vos réponses aux questions des évaluateurs satisfont ces derniers.

L'évaluation vérifie le fonctionnement sur les différents jeux de données ainsi que la conformité du code aux directives présentées ci-avant et aux meilleures pratiques découvertes dans les tutoriaux ou dans d'autres expériences. Le compte-rendu est rédigé en **anglais** et tient sur une **page A4**. Il décrit, en **10 points obligatoires**, les principaux faits, qu'il s'agisse de problèmes lors des tests, d'un fait notoire comme une solution élégante ou, au contraire, à améliorer. Le compte-rendu est déposé sur le git dans le dossier */Documents/Rendus* avant la séance plénière.