Projet Pogo (danse)

CLASSE PRE-ING1 • 2021-2022

DESCRIPTION

Réaliser un programme qui va simuler le comportement de personnes qui pratiquent le pogo pendant un concert de rock.
La piste sera représentée par une matrice 2D dans laquelle chaque case pourra contenir au plus 1 personne
Chaque personne sera modélisée au minimum par un vecteur vitesse et par une information sur son type de danseur «classique » ou « pogoteur » (mais surement d'autres informations supplémentaires)
A chaque étape de la simulation, les personnes se déplaceront : si 2 ou plusieurs personnes entrent en collision, il faudra modifier leurs vecteurs vitesse afin de simuler le choc .
Ce programme affichera chaque étape de la simulation à l'écran et stockera dans un fichier les données de ces étapes.
Ce projet met l'accent sur la gestion d'un tableau de structures, sur l'algorithmique afin de tester les collisions entre N éléments, sur la gestion des fichiers, ainsi que sur l'affichage d'une matrice à l'écran.

CAHIER DES CHARGES MINIMAL

- Initialisation des danseurs à des positions aléatoires sur une matrice 2D
- Modélisation des mouvements de danse par défaut (pas de collision) : un danseur peut bouger (vecteur vitesse minimum) de droite à gauche, ou de haut en bas, ou en diagonale, selon des critères aléatoires à définir
- Un pogoteur peut bouger de plusieurs cases par défaut (norme du vecteur vitesse minimum plus grande qu'un danseur «classique»)
- Quand 2 danseurs entrent en collision, chacun récupère une partie du vecteur vitesse de l'autre : un danseur classique récupèrera la moitié de la vitesse de l'autre personne, alors qu'un pogoteur récupèrera la totalité
- Si plus de 2 personnes entrent en collision, chacune récupèrera la somme des contributions des vecteurs des autres danseurs
- Si la norme du vecteur vitesse d'un danseur est supérieure à sa vitesse minimum, il faudra le réduire de X% à chaque étape (inertie)
- Affichage de chaque étape de la simulation : utilisation des séquences d'échappements pour afficher la piste de danse au même endroit sur le terminal (et donner ainsi une illusion d'animation sur l'écran)
- Vitesse d'animation modérée pour pouvoir observer le comportement de la simulation
- Stockage des informations de la simulation à chaque étape dans un fichier texte avec un format spécifique décrit ci-après.

BONUS POSSIBLES

- Position initiales des danseurs passées en paramètre au programme dans un fichier avec le format de votre choix
- Couleurs des personnes affichées en fonction de leurs types de danse, et/ou de leurs vitesses
- Gestion des danseurs qui sortent de la piste (bloqués sur les bords, éliminés de la simulation, remplacés par de nouveaux danseurs aléaoires, …)
- Réglage du terminal sur une police de caractères de taille très petite pour afficher une piste de danse plus grande (à indiquer dans votre *ReadMe*)
- Paramètres de la simulation passés en paramètres de votre exécutable
- Rejouer une simulation en chargeant un fichier .pogo précédent
- Possibilité de modifier la vitesse de simulation, faire du pas à pas, …
- Positions et vitesses intiales des danseurs compatibles avec le schéma « Wall of Death » ou « Mosh »

RESSOURCES UTILES

GitHub

- Site web : https://github.com

Format JSON

- Wikipédia : https://fr.wikipedia.org/wiki/JavaScript_Object_Notation

Le pogo

- Wikipédia : https://fr.wikipedia.org/wiki/Pogo_(danse)
- Mosh : https://fr.wikipedia.org/wiki/Mosh
- Wall of Death : https://fr.wikipedia.org/wiki/Mur_de_la_mort

Les séquences d'échappement d'un terminal

- Wikipédia : https://en.wikipedia.org/wiki/ANSI_escape_code

FORMAT .POGO

➢ Ce format permet de stocker l'ensemble des étapes de la simulation

➢ Le format interne du fichier sera compatible avec le format JSON. L'extension du fichier sera « .pogo »

➢ Ce fichier contiendra une norme (nimbre réel) de vecteur vitesse minimale pour le danseur classique (***classicMinSpeed***) et le pogoteur (***pogoMinSpeed***), ainsi qu'un nombre d'étapes (***nbSteps***).

➢ une liste d'étapes (***steps***) suivra, encadrée par des crochets, et les étapes séparées les unes des autres par des virgules.

➢ Le bloc de simulation sera encadré par des accolades, et chacun des champs cités sera séparé des autres par des virgules

```
{
  "classicMinSpeed" : 1,42,
  "pogoMinSpeed" : 3,5,
  "nbSteps" : N,
  "steps" : [
        BLOC_ETAPE_1,
        BLOC_ETAPE_2,
        …
        BLOC_ETAPE_N
  ]
}
```

➢ Le bloc pour une étape est défini ci-dessous

ETAPE .POGO

- Une étape est définie par bloc encadré par des crochets
- Ce bloc contient un champ « numéro d'étape » (***stepID***), un champ « nombre de danseurs » (***nbDancers***), et une liste des danseurs (***dancerList***) avec leurs caractéristiques
- Le champ ***stepID*** est une valeur entière positive
- Le champ ***nbDancers***, est une valeur entère positive
- Le champ ***dancerList*** est une liste de danseurs, encadrée par des crochets, et les danseurs séparés les uns des autres par des virgules :

```
{
   "stepID" : 1,
   "nbDancers" : N,
   "dancerList" : [
         BLOC_DANSEUR_1,
         BLOC_DANSEUR_2,
         ...
         BLOC_DANSEUR_N
   ]
}
```

- Le bloc pour un danseur est défini ci-dessous

DANSEUR .POGO

- Un bloc danseur est défini par plusieurs propriétés que vous aurez jugé utile d'ajouter. On peut déjà lister certaines indispensables comme la position (***x*** et ***y***) qui seront des entiers positifs ou nuls, indiquant l'index de colonne et de ligne du danseur)
- Il y aura un vecteur vitesse (***vx*** et ***vy***), qui seront des valeurs réelles
- Et enfin le type de danseur (***type***) classique ou pogoteur
- Eventuellement un identifiant (***id***) qui sera généré aléatoirement (pour pouvoir suivre le danseur au cours de la simulation)
- Le bloc danseur sera donc le regroupement de tous ces champs, encadré par des crochets, les champs séparés les uns des autres par des virgules

```
{
   "id" : "dancer001",
   "type" : "pogo",
   "x" : 5,
   "y" : 12,
   "vx" : -3.25,
   "vy" : 1.47
}
ou
{
   "id" : "dancer027",
   "type" : "classic",
   "x" : 37,
   "y" : 25,
   "vx" : 2.47,
   "vy" : -0.72
}
```