Présentation de mon PPP

Développeur web

Plan de la présentation :

I/ Description globale de mon projet (motivations)

II/ Présentation des sites web que j’ai déjà réalisé

III/ Recherches faites dans le domaine du développement web

IV/ Mes projets actuels

I/ Description globale de mon projet

Description générale :

- jeu de combat 2D

-se joue à deux ou trois

-se passe dans l’espace sur des planètes

-le but est de tuer son adversaire dans un temps imparti

I/ Description du jeu

Le début de la partie :

-plusieurs niveaux de difficulté (défini par la fréquence d’apparition des astéroïdes, (autres paramètres))

-apparition aléatoire des joueurs sur une planète différente

-chaque joueur a une barre de vie (pleine au début)

-bouclier protecteur de 3 secondes

I/ Description du jeu

Se déplacer au cours de la partie :

-masse des joueurs négligée par rapport à celle des planètes (n’influence pas leur trajectoire)

-impossible de léviter dans l’espace

-on peut tourner autour des planètes

-on peut dasher (il y a un temps de rechargement du dash)

-on peut changer de planète en sautant

-le changement se fait automatiquement si l’on est trop proche de l’autre planète

I/ Description du jeu

Comment perd on de la vie ?

-on peut se faire toucher par un astéroïde (qu'il faut éviter)

-on peut toucher son adversaire au corps à corps (avec un couteau)

-on peut toucher son adversaire à distance (en lançant un proton-tige)

-pour lancer un proton-tige, on peut orienter sa ligne de mire (il y a une aide à la visée)

-il n'est pas possible de se soigner, il faut donc être vigilant !

-on a tout de même le bouclier protecteur après chaque perte de vie

II/ Présentation des éléments s'inscrivant dans le cadre du programme

Le moteur physique, la gestion de la gravité :

-création d’une classe Champs pour gérer les interactions entre les objets (planètes et astéroïdes)

-on assimile tout objet à son centre de gravité

-on utilise le modèle standard de la gravitation universelle

-on calcule à chaque instant l'accélération des objets grâce au second principe

-on applique l’action gravitationnelle

II/ Présentation des éléments s'inscrivant dans le cadre du programme

Passages du code correspondant à la mise en place de la gravité :

```
    def computeAcc(self, A, B):
        "Calcule les accélérations entre les Points Matériels A et B. Renvoie le couple (accélération de A,
accélération de B)."
        ## calcul du vecteur unitaire en coordonnées cylindro-polaires
        r = abs(A.pos - B.pos)
        Ur = (A.pos - B.pos)/r
        ## on ne calcule pas la même accélération pour A et B. Elles sont dans la même direction, mais dans des
sens opposés, et on utilise respectivement les masses de B et A dans la formule de la force d'attraction
gravitationnelle, sinon le résultat serait faussé (A noter qu'ici l'expression de la force n'est pas utilisée.
En effet, l'ensemble des objects massifs présents dans le champs ne subissent que les forces d'attraction
gravitationnelle des uns sur les autres, la masse de l'objet attiré n'a donc pas d'importance dans l'expression
de l'accélération qui'il subit, contrairement à la masse de l'objet attracteur qui elle apparaît)
        return -G * B.mass / r**2 * Ur, G * A.mass / r**2 * Ur
```

```
    def applyGravity(self):
        "Calcule toutes les accélérations que subissent les objets massifs présents dans le champs les uns avec
les autres. La méthode applique également ces accélérations aux objets massifs concernés."

        ## simple algorithme de calcul des accélérations
        for i in range(len(self.objects)):
            Ai = self.objects[i]
            ## la boucle commence à i+1 pour ne pas calculer plusieurs fois les mêmes accélérations, ce qui
serait inutile, une perte de temps et faux
            for j in range(i+1, len(self.objects)):
                Aj = self.objects[j]
                acci, accj = self.computeAcc(Ai, Aj)
                Ai.applyAcc(acci)
                Aj.applyAcc(accj)
```

II/ Présentation des éléments s'inscrivant dans le cadre du programme

La gestion des planètes (et astéroïdes) :

-on considère les planètes comme étant homogènes (même densité partout) et indifférenciées (même matière partout)

-les planètes sont toutes supposées telluriques

-on ne prend pas en compte l’atmosphère que pourraient avoir les planètes

-on prend en entrée un intervalle de masses voulues

-on déduit un intervalle de rayons

-on calcule une densité de référence à l’aide d’une moyenne cubique

II/ Présentation des éléments s'inscrivant dans le cadre du programme

Passage du code correspondant au calcul de la densité :

```
def RADIUS(mass, density):
    return ((3*mass)/(4*pi*density))**(1/3)

def DENSITY(mass_min, mass_max, radius_min, radius_max):
    "Calcule la densité optimale, pour que le rayon, prenant des valeurs dans [mass_min, mass_max] soit à
valeurs dans [radius_min, radius_max]."
    ro1 = (3*mass_min)/(4*pi*radius_min**3)
    ro2 = (3*mass_max)/(4*pi*radius_max**3)
    ro = ((ro1**3 + ro2**3)/2)**(1/3)
    print("r(m) = {} (valeur attendue : {})".format(int(RADIUS(mass_min, ro)), radius_min))
    print("r(M) = {} (valeur attendue : {})".format(int(RADIUS(mass_max, ro)), radius_max))
    return ro
```

II/ Présentation des éléments s'inscrivant dans le cadre du programme

La gestion des personnages :

-on repère les personnages en polaire par rapport au centre de la planète où il se trouve

-on transforme ces coordonnées en cartésien car python fonctionne comme cela

-on applique son poids au joueur (seule force qu’il subit)