TP2 : Panda 3D Hello World

Objectif du TP

L'objectif de ce TP est de créer un monde 3D simple et d'y ajouter un personnage qui se déplace en gérant les collisions.

Mon premier monde avec Panda3D

1. Avec votre éditeur de texte préféré (Scintilla Text Editor installé lors du 1er TP, ou Pype, ou IDLE ou …), créez un nouveau fichier nommé `Hello3D.py` avec le contenu ci-contre. La ligne 1 permet d'importer la base nécessaire à l'exécution de Panda3D. La ligne 2 définit un nouveau type de données appelé `MyFirstGame` qui hérite de toutes les fonctionnalités du type de données `ShowBase`. Ensuite, en ligne 3, on définit une nouvelle fonctionnalité pour le type de données `MyFirstGame` qui s'appelle `__init__`. Cette fonction ou méthode est un peu particulière car elle correspond à un constructeur : c'est une méthode ou fonction qui est automatiquement appelée lorsqu'un nouvel objet du type `MyFirstGame` sera créé comme ca sera le cas en ligne 7. La ligne 5 fait appelle au constructeur du type de données auquel on hérite ; ainsi, un objet `MyFirstGame` se contruit, pour l'instant, de la même manière qu'un objet de type `ShowBase`. La ligne 8 fait appel à la méthode `run` de l'objet `app` du type de données `MyFirstGame`. Cette méthode a été définit par Panda3D dans le type de données `ShowBase`. Attention, respectez les indentations car le langage python utilise les tabulations comme moyen de définir un groupe d'instructions.

```
1 Hello3D.py
1  from direct.showbase.ShowBase import ShowBase
2
3  class MyFirstGame(ShowBase):
4      def __init__(self):
5          ShowBase.__init__(self)
6
7  app = MyFirstGame()
8  app.run()
```

2. Exécuter le programme saisi précédemment. Que constatez-vous ?

3. Nous souhaitons dorénavant télécharger notre environnement de jeu créé dans un logiciel de modélisation 3D comme Blender, Maya ou 3DS Max. Ce n'est pas l'objectif de ce cours de vous initier à cette modélisation. Néanmoins, il faut savoir que Panda3D possède son propre format de fichier 3D : les fichiers textuels lisibles (`*.egg`) ou les fichiers binaires (`*.bam`) qui se téléchargent plus rapidement. Panda3D propose néanmoins la possibilité de convertir des fichiers 3D (`vrml`, `x`, `x3d`, `maya` …) en fichier egg. Ajoutez les lignes ci-contre au constructeur de votre premier jeu et exécutez votre programme.

```
self.environ = self.loader.loadModel("models/groundPlane")
self.environ.reparentTo(self.render)
base.cam.setPosHpr(0,-210,135,0,327,0)
```

   La 1ère ligne demande au `loader` qui existe déjà dans le type `ShowBase` de charger un modèle 3D avec un nom et un répertoire (méthode `loadModel`). La variable `environ` contiendra un objet de type `NodePath`. Vous pouvez savoir ce que contient le type `ShowBase` en cliquant dessus. Toutefois, il ne suffit pas de charger le modèle 3D pour qu'il s'affiche dans la fenêtre. C'est l'objectif de la 2ème ligne qui s'occupe d'ajouter (`reparent`) le modèle ainsi chargé au graphe de scène. Que sa quo ? Panda3D contient une structure de données appelée *Graphe de Scène* qui se présente sous la forme d'un arbre contenant tous les objets devant être affiché ou rendu. Chaque objet possède un nom et la racine de l'arbre s'appelle le `render`. Par conséquent, pour ajouter le terrain de pelouse dans notre scène et ainsi le rendre visible, nous devons l'ajouter à l'arbre avec la méthode `reparentTo`.
   La 3ème ligne permet de positionner la caméra afin d'avoir une vision globale de la scène. Nous y reviendrons plus tard.

4. Ajoutez des commentaires à vos instructions (si possible un commentaire en anglais avant chaque instruction) avec un caractère # qui explique l'instruction suivante. Forcez-vous à le faire pendant tous les TPs.

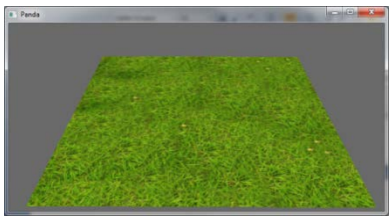
5. Exécutez votre programme et bougez votre modèle avec la souris :

Bouton Gauche	:	Translation du modèle suivant les axes X et Z
Bouton Droit	:	Zoom avant ou arrière
Bouton Droit et Gauche	:	Rotation autour des axes

```
class MyFirstGame(ShowBase):
    def __init__(self):
        ShowBase.__init__(self)
        #load the environnement model
        self.environ = self.loader.loadModel("models/groundPlane")
        #Add the model to Graph Scene (render)
        self.environ.reparentTo(self.render)
        #modify camera position & direction
        base.cam.setPosHpr(0,-210,135,0,327,0)
app = MyFirstGame()
app.run()
```

Remarquez qu'il est possible de ne plus rien voir : cela signifie que votre modèle 3D est uniquement constitué d'une texture qui s'affiche que d'un seul coté.
Il est également possible de désactiver l'utilisation de la souris avec l'instruction `base.disableMouse()`. Essayez-la.

6. Comme nous l'avons précisé précédemment, il est possible de télécharger un modèle 3D plus rapidement en le convertissant en fichier bam. Pour cela, appliquez la méthode `writeBamFile('pelouse.bam')` sur votre environnement et exécutez le programme. Vérifiez que vous avez bien un nouveau fichier `pelouse.bam`. Maintenant, modifiez (en commentant) votre programme pour ne plus écrire de fichier bam mais plutôt charger le fichier bam au lieu du fichier egg. Dans notre cas, vu que le modèle chargé est très léger, vous ne verrez pas de différences. Mais plus vous avancerez dans le TP, plus nous allons charger de modèles (joueurs, objets 3D du monde…), plus le chargement initial prendra du temps.



Ajout et Animation d'un personnage

7. Nous allons utiliser la classe `Actor` pour charger des modèles 3D animés contrairement à précédemment où la méthode `loadModel` était utilisée pour charger des modèles statiques. Ainsi, il faut importer un nouveau module qui contient la définition du type `Actor` avec l'instruction `from direct.actor.Actor import Actor`.

8. Avec l'instruction suivante `self.ralph = Actor("models/ralph",{"run": "models/ralph-run","walk":"models/ralph-walk"})`, nous ajoutons un nouvel acteur nommé `ralph` dont le modèle se trouve dans le répertoire `models` de votre répertoire courant. Par ailleurs, l'instruction permet de définir deux animations `run` et `walk` avec leur modèle respectif. Afficher `ralph`.

9. Une fois l'acteur ajouté, vous pouvez animé Ralph en demandant à ce qu'il boucle sur une des animations `walk` ou `run` avec l'instruction `ralph.loop("walk")`. Que remarquez-vous ?

10. Vous pouvez arrêter toute animation avec l'instruction `stop()` sur l'`Actor`. Vous pouvez également lancer l'animation qu'une seule fois avec la méthode `ralph.play("run")`. Mettez en commentaire l'animation, nous y reviendrons plus tard.

Déplacer le personnage

11. Afin que le joueur soit un peu plus immergé dans le jeu, il faut qu'il puisse interagir avec le jeu. Pour cela, nous devons lui donner la possibilité de déplacer Ralph. Ceci nous permettra également de mieux utiliser les animations vu précédemment. Panda3D propose la gestion de saisie de touche clavier avec les fonctions `accept(<caractère souhaité>, <méthode à appeler>, <paramètres de la méthode appelée>)`. Par exemple, on utilisera l'instruction `self.accept('escape', sys.exit)` si on veut quitter l'application lorsque l'utilisateur appuie sur la touche `Escape`. La méthode `exit` se trouve dans le module `sys` qu'il faut importer (`import sys`). Modifier votre programme pour pouvoir quitter en appuyant sur `Escape`.

12. Pour déplacer Ralph vers la gauche, nous devons ajouter la gestion de la touche de direction gauche avec `self.accept('arrow_left', self.DeplaceRalphGauche)`. Par conséquent, nous devons définir la méthode `DeplaceRalphGauche` qui déplace Ralph vers la gauche de la façon suivante :
```
def DeplaceRalphGauche(self):
    self.ralph.setPos(self.ralph.getPos()+Point3(-1,0,0))
```
Cette fonction ne pourra marcher qu'en important le module qui connait le type `Point3` (`from panda3d.core import Point3`).
Ajoutez la gestion des autres touches de direction.

13. En programmation, il est beaucoup plus intelligent d'utiliser des constantes plutôt que des valeurs numériques sans savoir exactement à quoi elles servent. Pour cela, définissez une variable intitulée `VitesseDeplacementRalph` à `1` juste après la définition du type de données `MyFirstGame`. De cette façon, si vous souhaitez modifier la vitesse de déplacement de Ralph, il suffit de modifier la valeur qu'à un seul endroit dans le code source. Pour cela, il faut modifier les méthodes `DeplaceRalphXXXX` en remplaçant les `1` par `self.VitesseDeplacementRalph`.

14. Nous avons défini 4 méthodes pour le déplacement de Ralph. Nous allons maintenant utiliser la notion de paramètre pour transformer ces méthodes en une seule. En effet, quelque soit la direction, nous allons appeler la même méthode de déplacement de Ralph sauf que cette fois, nous allons passer en paramètre la direction dans laquelle Ralph doit se déplacer. Ainsi, commentez les anciennes méthodes de déplacement et remplacez les par l'unique méthode `DeplaceRalph(self,dir)` en sachant que `dir` correspond à un `Point3`. Il faudra également modifier les lignes qui précise les touches que l'on doit gérer ; par exemple : `self.accept('arrow_left', self.DeplaceRalph, [Point3(-self.VitesseDeplacementRalph,0,0)])`. Python demande à ce que les paramètres de la méthode soit mis sous forme de liste, c'est pour cela que la direction est mise entre crochets (`[]`).

15. On souhaite orienter Ralph dans la direction du déplacement. Pour cela, il faut tester les valeurs de la variable `dir` pour savoir dans quelle direction on va aller et ensuite modifier l'orientation de Ralph à l'aide de la méthode `setHpr()` de la façon ci-contre. Nous reviendrons plus tard sur cette méthode.

```
if (dir[0]<0):
    rot=-90
elif (dir[0]>0):
    rot=90
elif (dir[1]>0):
    rot=180
else:
    rot=0
self.ralph.setHpr(rot,0,0)
```

ANIMER LE PERSONNAGE

16. Afin d'avoir plus de réalisme, nous souhaiterions utiliser les animations `walk` et `run` qui ont été définis lors du chargement du modèle3D de Ralph. Pour cela, ajoutez tout d'abord un nouveau paramètre `isRunning` à la fonction `DeplaceRalph` pour savoir si Ralph est en train de courir ou marcher. Vous devez également modifier les appels à cette méthode (gestion des touches clavier) en précisant qu'il ne court pas (`False`). Ensuite, pour animer Ralph, nous allons utiliser la méthode `play` sur Ralph en précisant quelle animation on souhaite jouer (`self.ralph.play("walk")` ou `self.ralph.play("run")` en fonction de la valeur de `isRunning`). Testez votre programme. Que remarquez-vous ?

17. Vous avez surement remarquez que lorsque vous laissez votre doigt appuyé sur une touche de direction, Ralph ne se déplace qu'une seule fois. En effet, cette action correspond à un autre événement que Panda3D propose de gérer en détectant la répétition d'une touche de la façon suivante : `self.accept('arrow_left-repeat', …`
Modifiez votre programme pour que Ralph court quand vous laissez votre doigt appuyé sur une touche de direction. On utilisera toujours la même méthode de déplacement `DeplaceRalph` mais avec des valeurs de paramètres différents ...

18. Pour éviter que Ralph fasse de la lévitation ou le moonwalk, nous ne devons pas lancer une animation alors qu'elle est déjà en train de s'exécuter. Modifiez votre programme en utilisant l'instruction qui détecte l'animation en cours d'exécution : `if (self.ralph.getCurrentAnim()<>"run"):…`

19. Afin que le constructeur ne soit pas trop compliqué à lire, déplacez toutes les instructions de gestion des touches dans une nouvelle méthode intitulée `setKeys` et appelez cette méthode dans le constructeur. Nous avons enfin un personnage un peu plus réaliste qu'au début même si c'est encore perfectible…

UN PEU PLUS D'IMMERSION

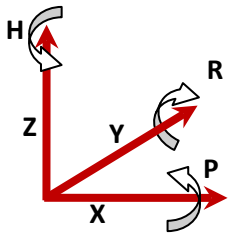
20. Maintenant que nous avons un personnage qui se déplace sur de la pelouse, ca serait bien qu'il ne voit pas les bords du monde que nous venons de créer. Pour cela, nous allons ajouter le ciel. Ajouter un nouveau modèle nommé `sky` à votre monde et appelons ce modèle le `skydome`. Modifier la taille du dôme avec la méthode `setScale()` et utilisez comme valeur la limite du terrain. Pour cela, commencez par définir une donnée nommée `LimiteTerrain` égale à `70` qui correspond à la taille du terrain. Cette valeur permet à ce que les bords du dôme soit dans la pelouse afin que Ralph ne vois que du bleu ou du vert.

21. Modifier la méthode `DeplaceRalph` pour qu'il ne puisse pas marcher au-delà du dôme. Vous aurez surement besoin de la méthode `getPos()` qui s'applique sur un acteur et renvoyant un triplet contenant les positions X, Y et Z. Pour accéder à X par exemple, utiliser l'instruction `getPos()[0]`.

22. Si on souhaite avoir un jeu avec une vision de FPS, il faut déplacer la caméra. Pour cela, mettez en commentaires la ligne qui modifie le positionnement de la caméra et exécutez à nouveau votre programme. Que voyez-vous ? Quittez le programme et ajouter l'instruction `base.oobeCull()`dans le constructeur; relancez votre programme…
    Vous avez eu peur … hein, avouez !!!!!!
    Manipulez la souris pour avoir une vision plus large. Que constatez-vous ?
    En réalité, le module ShowBase propose cette nouvelle instruction qui signifie « Out Of Body Experience » et permet d'avoir une vision divine (God Eye's view).

23. Pour résoudre le problème précédent, le plus simple est de modifier l'orientation initiale de Ralph à l'aide de l'instruction `setH(angleEnDegréCelcius)`afin d'aligner la vision de Ralph avec la vision de la caméra. Que voyez-vous dorénavant ?

24. Il existe en réalité 2 caméras : la caméra divine, représentée par `base.cam` (caméra pour laquelle on a commenté le code dans la question 21) et une autre caméra accessible par `base.camera`. L'instruction `base.oobeCull()` permet donc de rendre visible la caméra `camera` ainsi que son champ de vision ; lorsque l'instruction `base.oobeCull()` sera utilisée, on considérera être en *débogage visuel*. Néanmoins, par défaut Panda3D exécute une tâche qui vous permet de bouger la caméra avec la souris.
    Si vous souhaitez modifier `camera` et ainsi l'observer lors du débogage visuel, il faut avant tout désactiver la tâche par défaut avec l'instruction `base.disableMouse()` car les instructions qui modifieraient la position ou l'orientation de `camera` rentre en conflit avec la tâche par défaut et donne l'impression que les modifications ne fonctionnent pas. Modifiez la position de `camera` pour qu'elle se trouve à la hauteur des yeux de Ralph et un peu derrière lui. Pour cela, utilisez la méthode `setPosHpr()`qui modifie la position et l'orientation HPR (*H*eading, *P*itch, *R*oll) suivant le schéma ci-contre. Vérifiez avec et sans la vision de débogage visuel.



25. Le problème d'avoir modifié la position de la camera et de ne plus avoir la vision divine est que lorsque Ralf se déplace, la camera reste figée et ainsi on ne voit plus Ralph. Une solution à cela est d'utiliser la notion de *placement relatif* à un autre objet dans le graphe de scène. De cette façon, si un objet est déplacé, alors

l'autre objet relatif au premier sera déplacé automatiquement. L'idée est de positionner la caméra `camera` comme nous l'avons fait dans la question 24 mais de telle sorte que si Ralph se déplace et se tourne, alors la camera également. Pour cela, il suffit simplement de préciser que `camera` dépend de Ralph ou que Ralph est le *parent* de `camera` avec la méthode `reparentTo`. Dans le constructeur, précisez que le père de `base.camera` est Ralph. Attention, il y a des chances que vous deviez modifiez l'orientation initiale de Ralf et/ou de la camera, modifiez plutôt celle de la caméra.

26. Afin d'avoir un code structuré, créez une nouvelle méthode `loadModels()` et déplacez toutes les instructions relatives au chargement des modèles 3D ou personnages dans cette nouvelle méthode.

GESTION DE COLLISIONS

27. Afin d'agrandir le monde dans lequel Ralph se déplace et lui autoriser plus d'accès dans le monde que nous avons créé, nous allons ajouter des objets tout autour du terrain et autoriser Ralph à se déplacer n'importe où :

    a. Commentez les lignes qui restreignent le déplacement de Ralph (cf. question 21) ;

    b. Affichez le monde dans le mode de vision divine ;

    c. Ajoutons des boites afin de délimiter le terrain. Dans la méthode `loadModels`, chargez le modèle `box` et affichez-le. Redimensionnez-le avec la méthode `setScale(10,0.5,0.5)`. Ensuite, positionnez-le au bord du terrain (utiliser la valeur `LimiteTerrain + 3` qui correspond à la dimension du terrain + la moitié de la boite). Faites de même avec les autres cotés (il suffit de changer l'échelle dans une autre direction).

```
self.box1 = self.loader.loadModel("models/box")
self.box1.setScale(10,0.5,0.5)
self.box1.setPos(0,self.LimiteTerrain+3,0)
self.box1.reparentTo(self.render)
```

    d. Augmentez la taille du dôme pour qu'il aille au delà des boites ;

    e. Testez à nouveau le déplacement de Ralph. Est-ce qu'il peut traverser les boites ?

28. Afin que Ralph ne se prenne pas pour un fantôme pouvant aisément traverser les murs, nous devons ajouter la gestion de collisions entre les objets du monde 3D et Ralph. Mais avant de configurer ce gestionnaire, nous devons construire et ajouter dans le graphe de scène les objets 3D pour lesquels nous souhaitons détecter les collisions ; ce sont les solides de collision. En effet, il faut distinguer les objets 3D à visualiser des objets 3D représentant les collisions à détecter. Pour que Ralph ne puisse pas traverser les murs, le plus simple est de créer des solides correspondant à des plans (`CollisionPlane`). Les types de données permettant cette gestion se trouvent dans le module `panda3d.core`. La démarche à suivre est la suivante :

    a. Commencez par créer une méthode `setCollisions(self)` que vous appellerez dans votre constructeur et qui s'occupera de gérer toutes les collisions.

    b. Dans cette nouvelle méthode, on crée un nœud du graphe de scène correspondant à un nœud de collision en lui donnant un nom pour l'identifier éventuellement plus tard : `CollWorld = CollisionNode('collworld')`.

    c. Il faut associer à ce nœud de collision une géométrie correspondant à 4 plans perpendiculaires en utilisant la méthode `addSolid` sur ce nœud et en passant en paramètre un solide de collision : un plan dans notre cas. Un plan peut être créé de plusieurs façons, mais la manière la plus simple est de préciser une direction vers laquelle le plan est orienté ainsi qu'une position où se trouve le plan. Ainsi, nous utilisons l'instruction `collWorld.addSolid(CollisionPlane(Plane(Vec3(1,0,0), Point3(-self.LimiteTerrain,0,0))))` pour ajouter le plan de gauche. Ajoutez les 3 autres plans.

d. Nous venons de créer la géométrie correspondante aux limites du monde dans lequel Ralph peut évoluer, mais nous devons ajouter ce nœud particulier au graphe de scène avec l'instruction : `collWorldNodePath = self.render.attachNewNode(collWorld)`. Il est possible d'afficher le nœud de collision ainsi créé avec l'instruction `collWorldNodePath.show()` pour vérifier que le solide créé est correct.

29. Nous devons maintenant ajouter le solide de collision correspondant à Ralph. Comme Ralph est un objet 3D compliqué et pour que Panda3D gère plus facilement et plus rapidement les collisions entre Ralph et son environnement, nous allons créer une sphère de collision autour de Ralph (cf. ci-contre) :

a. Pour cela, nous allons récupérer les dimensions de Ralph avec l'instruction `bounds = self.ralph.getChild(0).getBounds()`.

b. Ensuite, nous pouvons récupérer le centre et le rayon de ces dimensions avec les instructions : `center = bounds.getCenter()` et `radius = bounds.getRadius()`.

c. Enfin, inspirez vous de la question 28 pour ajouter un nouveau solide de collision au graphe de scène correspondant à une sphère. La création de la sphère se fera simplement comme cela : `CollisionSphere(center, radius)`.

30. Maintenant que les géométries de collision sont créées, il est nécessaire de configurer la gestion de collision. Il existe plusieurs façons de gérer les collisions mais dans le cas présent, celui qui nous intéresse et le gestionnaire `CollisionHandlerPusher`. Ce gestionnaire facilite les collisions avec les murs. Quand un objet rentre en collision (Ralph par ex.) avec un autre objet (un mur par ex.), ce gestionnaire repositionne le premier objet (Ralph) à la position précédent la collision. C'est exactement ce que nous souhaitons faire. Voila la démarche à suivre :

a. Dans ShowBase, il existe une variable correspondante à la détection de collisions dans le graphe de scène accessible via `base.cTrav`. Nous allons initialiser cette variable en créant un nouveau objet qui permet de parcourir le graphe de scène à la recherche de nœud de collisions `CollisionTraverser` avec l'instruction `base.cTrav = CollisionTraverser()`.

b. Nous allons créer le gestionnaire de collisions qui nous intéresse avec l'instruction `pusher=CollisionHandlerPusher()`.

c. Ensuite, nous allons ajouter ce gestionnaire à l'objet qui parcoure le graphe de scène avec l'instruction : `base.cTrav.addCollider(collRalphNodePath, pusher)`.

d. Enfin, il est nécessaire de configurer ce gestionnaire en précisant les solides de collision qui doivent être détectés par le gestionnaire avec l'instruction : `pusher.addCollider(collRalphNodePath, self.ralph, base.drive.node())`.

e. Maintenant, vous pouvez tester votre programme qui devrait empêcher Ralph de traverser les murs. Il est possible d'afficher en rouge les collisions détectées avec l'instruction : `base.cTrav.showCollisions(self.render)`.

Pour aller plus loin

31. Le changement brutal de direction, surtout dans la vision FPS, est très déroutant. Modifiez votre programme pour que les touches de direction droite et gauche ne modifient pas la position de Ralph mais simplement son orientation. Ensuite, faites en sorte que Ralph ne puisse pas reculer ; ainsi supprimez la gestion de la touche de direction vers le bas.

32. Le monde créé dans ce TP est un monde 2D. Regarder ce tutoriel pour créer un monde 3D à partir d'images. Toutefois, le déplacement de Ralph sera beaucoup plus compliqué puisqu'on devra gérer les collisions entre le personnage et le terrain comme c'est réalisé dans l'exemple de Panda3D : Uneven Terrain avec un gestionnaire de collisions spécifique `CollisionHandlerQueue` et un rayon qui traverse le personnage de la tête au pied. Ce gestionnaire permet de connaître la position du point de collision, ce qui permettra de positionner Ralph juste au dessus du terrain accidenté.

Sources pour ce TP

Panda3D : http://www.panda3d.org/
Librairie PandAI : http://www.etc.cmu.edu/projects/pandai/
Tutoriel d'un Shoot 'Em Up en Panda3D : http://www.mygamefast.com/volume1/issue1/?nopages=true