TP3 : Steering Behaviors

Objectif du TP

L'objectif de ce TP est d'appliquer des comportements basiques à des personnages non joueurs afin de leur donner un comportement plus intelligent comme Pursue, Evade, Wander, Flock, Obstacle Avoidance ainsi que le fameux PathFinding. Il en existe d'autres comme Seek, Flee, Arrival ou Path Following.

Ajout de PNJ et notion de Tâche

1. Bien évidemment, il faut ajouter de nouveaux Personnages Non Joueurs. Pour cela, examinez le contenu du répertoire `models` et chargez le modèle du personnage Eve ainsi que *toutes* ses animations. Conseil : servez-vous des instructions utilisées pour charger Ralph et commentez momentanément l'ajout de ralph dans le graphe de scène.
2. Visionnez chacune des animations en exécutant le programme (une animation par exécution).
3. Afin d'observer le comportement des joueurs (Eve ou Ralph), il est plus adéquat de passer dans une vision divine. Modifiez votre programme pour avoir cette vision.
4. Avant de détailler la notion d'agents intelligents, nous devons étudier de plus près la notion de tâche ou `Task` dans Panda3D. Les tâches sont des méthodes qui sont appelées automatiquement par Panda3D lors du rafraichissement de la scène. Vous avez déjà utilisez une tâche sans vraiment le savoir : la gestion des boutons de la souris que vous utilisiez pour vous déplacer dans le monde 3D est une tâche définie par défaut dans `ShowBase`. Je vous rappelle que vous l'avez peut être désactivée avec l'instruction `base.disableMouse()`. Créons notre propre tâche en créant une nouvelle méthode `moveCameraTask` qui déplace automatiquement la caméra autour de Ralph en fonction du temps obtenu grâce à l'instruction `task.time` (cf. code ci-contre). Il faudra ajouter le module `Task` du package `direct.task`. Puis, ajoutez l'instruction `self.taskMgr.add( self.moveCameraTask, "moveCameraTask")` qui ajoute notre méthode au gestionnaire des tâches de Panda3D en lui donnant un nom pour pouvoir l'identifier plus tard si nécessaire. Pour bien comprendre ce que fait la tâche, activer le mode de débogage visuel (`base.oobeCull()`) et désactiver la souris. Maintenant que vous avez compris la notion de tâche, vous pouvez commentez l'instruction qui ajoute cette tâche au gestionnaire car elle ne nous intéresse pas ; attention, je n'ai pas forcément dit de supprimer la méthode `moveCameraTask`.

```
# Define a procedure to move the camera.
def moveCameraTask(self, task):
    angleDegrees = task.time * 6.0
    angleRadians = angleDegrees * (pi / 180.0)
    self.camera.setPos(20 * sin(angleRadians),
        -20.0 * cos(angleRadians), 3)
    self.camera.setHpr(angleDegrees, 0, 0)
    return Task.cont
```

Déplacement aléatoire (Wander)

5. Pour ajouter des comportements intelligents à nos PNJ, nous devons utiliser la librairie PandAI incluse dans Panda3D. Le contenu de ce TP est une adaptation/traduction du tutoriel contenu dans le manuel de Panda3D. Pour pouvoir utiliser cette librairie et associer le comportement le plus simple (déplacement aléatoire = wander) à Eve, il faut :
    a. Inclure la librairie avec l'instruction `from panda3d.ai import *` ;
    b. Ajoutons une méthode qui va définir les comportements intelligents de notre agent : `def setAI(self)` : et ajoutez l'appel à cette méthode dans le constructeur `self.setAI()`.
    c. Dans la méthode `setAI`, construisons un objet du type `AIWorld` qui va gérer tous les agents intelligents (`AICharacter`) de notre monde : `self.AIworld = AIWorld(render)` ;

d. Construisons un nouvel agent intelligent en lui donnant le nom `AIEveWander`, en l'associant à un personnage de notre monde, et en précisant son poids, sa vitesse et sa vitesse maximale : `self.AIeve = AICharacter("AIEveWander", self.eve, 100, 1, self.VitesseDeplacementRalphCourse);`

e. Ajoutons ce nouvel agent au gestionnaire des agents intelligents : `self.AIWorld.addAiChar(self.AIeve);`

f. Récupérons les comportements intelligents de ce nouvel agent : `self.AIbehaviors = self.AIeve.getAiBehaviors();`

g. Appliquons le déplacement aléatoire sur cet ensemble de comportements. Le comportement `wander` de Panda3D demande 4 paramètres : un angle maximum de changement de direction, un entier représentant le plan dans lequel l'agent va se déplacer (0 pour le plan XY, 1 - YZ, 2 - XZ, 3 - XYZ), le rayon du cercle dans lequel l'agent va se déplacer (le point de départ de l'agent correspondra au centre du cercle), et la priorité de ce comportement par rapport à d'autres comportement définis sur le même agent : `self.AIbehaviors.wander(5,0,self.LimiteTerrain / 2, 1)`.

h. Comme l'agent à pour objectif de se déplacer dans notre jeu, il faut lui appliquer une animation comme la marche par exemple : `self.eve.loop("walk");`

i. C'est maintenant que la notion de tâche (cf. exercice 4) intervient. En effet, pour donner un comportement intelligent, il faut continuellement modifier la position et/ou l'orientation de notre personnage. Pour cela, il faut donc ajouter une nouvelle tâche, et donc une nouvelle méthode, qui va au moins appeler la méthode `update()` du monde intelligent que nous avons créé précédemment. On ajoute donc l'instruction `taskMgr.add(self.AIUpdate, "AIUpdate")` dans la méthode `setAI` et la méthode ci-contre.

```
#to update the AIWorld
def AIUpdate(self,task):
    self.AIworld.update()
    return Task.cont
```

j. Exécuter votre programme pour observer le résultat. Vous pouvez modifier temporairement la masse d'Ève afin de mieux observer son déplacement et apprécier son comportement intelligent. La vitesse dépend de sa masse. Que remarquez-vous ?

6. Sur cette page de forum, vous pouvez lire les commentaires des auteurs sur la notion d'*Area of Effect* qui explique le problème rencontré précédemment. Ainsi, pour résoudre ce problème, inspirez-vous des questions 29 et 30 du TP2 que nous avons résolu pour Ralph.

PURSUE ET EVADE

7. Nous souhaitons tester le comportement `pursue`. L'idée est qu'Ève poursuive Ralph en sachant qu'il peut se déplacer plus vite puisqu'il court. Toutefois, initialement, nous devons placer Ève à un autre endroit que Ralph sinon nous ne pourrons pas observer le comportement. Pour cela, importons d'abord le package `random` avec l'instruction `import random` qui gère les nombres aléatoires. Ensuite, dans la méthode `loadModels`, définissons la position initiale de Ève en utilisant la méthode `setPos()` et en générant un nombre aléatoire entre a et b avec l'instruction `random.randint(a,b)`.

8. Ensuite, commentez la création du comportement `Wander` et inspirez-vous en pour créer le comportement `Pursue`. Il faudra pour cela donner 2 paramètres : quel objet l'agent doit poursuivre (ce sera Ralph) et la priorité de ce comportement qui est à `1` par défaut. Exécutez votre programme pour admirer ce comportement.

9. Revenons sur cette notion de priorité. Vous pouvez combiner les comportements intelligents d'Ève, par exemple, en définissant une priorité de `0.5` à chaque comportement. Pour cela, dé-commentez le

comportement `wander` et combinez le avec `pursue`. Testez votre programme et appréciez … Qu'en déduisez-vous ?

10. Pour mieux apprécier la combinaison de comportement, on se propose de combiner `wander` avec le comportement `evade`. Le scénario consiste à ce que ça soit Ralph le chasseur et Ève l'objectif. Programmez ce cas de figure en utilisant le comportement evade qui est l'inverse de `pursue` ?

Limite(s) de Pursue

11. Afin que notre jeu soit un peu plus crédible, il est nécessaire d'ajouter des obstacles dans notre monde. Pour cela, charger et afficher le modèle `Table`. Faites en sorte qu'il ne soit pas positionné en 0,0,0 et augmentez sa taille de 2 en largeur uniquement.

12. Si vous testez le programme précédent, vous vous rendrez compte que Ralph ou Ève peuvent traverser la table. Il faut donc ajouter une boite de collision dans le graphe de scène avec les dimensions de la table. Pour connaitre ces dimensions, il suffit d'utiliser la méthode `getTightBounds()` sur la table ; l'instruction `minpoint,maxpoint=self.table.getTightBounds()` nous permet de récupérer les extrémités de la table utiles pour la définition de la boite de collision. Inspirez vous du TP2 pour que les joueurs ne puissent plus traverser la table. Faites en sorte qu'Ève poursuive Ralph à nouveau (vous pouvez laisser la combinaison avec `wander`). Afin de bien se rendre compte de la limite du comportement , pendant l'exécution du jeu, placez la table entre Ralph et Ève comme on peut le voir sur la figure ci-contre. Que constatez-vous ?

14. Pour tenter de résoudre le problème de la question précédente, il existe le comportement Obstacle Avoidance qui rendra un peu plus d'intelligence à Ève. Pour ce comportement, il est nécessaire d'ajouter au comportement quel(s) obstacle(s) il doit éviter. Cela se fait avec la méthode `self.AIworld.addObstacle(self.table)`. Testez ce nouveau comportement avec plusieurs positions de Ralph et d'Ève pendant le jeu. Que constatez-vous ?

Pathfinding

15. Commentez les comportements utilisés précédemment (Wander, Pursue, Evade et Obstacle Avoidance).

16. Comme vous l'aurez compris, le meilleur moyen pour qu'Ève poursuive Ralph est d'utiliser le fameux Pathfinding. Ce comportement est un peu plus compliqué que les autres car il nécessite un maillage 2D sur lequel les PNJs peuvent se déplacer et sur lequel ils peuvent calculer le plus court chemin. Ce maillage est appelé `Navigation Mesh` dans Panda3D. La création de ce maillage nécessite l'utilisation d'un moteur 3D type 3DSMax, Blender ou Maya et qui doit ensuite être converti en fichier `Comma Separated Values` (CSV). Comme nous n'avons pas le temps de s'y attarder (lire les tutoriaux disponibles dans PandAI), nous allons utiliser un maillage déjà tout fait, obtenu dans les multiples exemples de Panda3D. Ainsi, pour utiliser ce comportement, il est nécessaire de :

    a. Charger le maillage 2D avec l'instruction `self.AIbehaviors.initPathFind ("models/navmesh.csv")`.

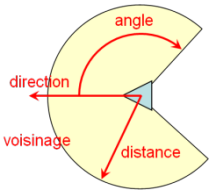
    b. Préciser à l'algorithme du plus court chemin qu'il y a un objet qu'Ève ne peut pas traverser avec l'instruction `self.AIbehaviors.addStaticObstacle(self.table)`. Comme la table n'est pas sensée bouger, elle est considérée comme statique. Si elle pouvait bouger dans le temps, il aurait fallu utiliser la méthode `addDynamicObstacle.(self.table)`.

    c. Demander à Ève de suivre le plus court chemin avec la méthode `self.AIbehaviors.pathFindTo(self.ralph)`.

d. Testez votre programme. Il est possible qu'il y ait encore des comportements bizarres dû surement à une incompatibilité du maillage et de notre monde.

FLOCKING (OU INTELLIGENCE COLLECTIVE)

17. Le flocking est un comportement qui autorise un PNJ à se déplacer de manière réaliste dans un groupe d'agents. Il est d'abord caractérisé par un nombre qui identifie de manière unique un groupe d'agents. Il est possible qu'il y ait plusieurs groupes différents avec des flockings différents. Il est également constitué d'un angle de vision (ou cône de vision) et la longueur du cône de vision représenté par un rayon qui correspond au voisinage perceptible par le PNJ. Enfin, il y a les valeurs de séparation (garder une certaine distance avec son voisin), de cohésion (calculer la position moyenne du groupe et y aller) et d'alignement (calculer la direction moyenne du groupe et tendre vers cette direction) qui caractérise le comportement collectif lui-même. Voici un tableau que les auteurs considèrent comme un bon point de départ pour tester le flocking :



Type	Angle de vision	Rayon de vision	Séparation	Cohésion	Alignement
Normal	270	10	2	4	1
Loose	180	10	2	4	5
Tight	45	5	2	4	5

Vous l'aurez compris : pour créer un flocking, il faut définir plusieurs agents pour apprécier leur comportement intelligent. Pour cela :

a. Définissons une nouvelle méthode appelée `setFlocking`.

b. Construisons un flocking avec l'instruction `self.flockObject = Flock(1, 270,10,2,4,1)`. Vous pouvez modifier les valeurs comme vous voulez selon les explications données précédemment. Ensuite, cet objet de flocking doit être ajouté au monde intelligent avec `self.AIworld.addFlock (self.flockObject)`.

c. Comme nous devons construire plusieurs agents du même type, nous allons utiliser la notion d'ensemble en construisant une liste d'agents vide (des fantômes dans notre cas) avec les instructions : `self.ghosts=[]` et `self.ghostsAI=[]` pour stocker la géométrie et l'intelligence de chaque agent.

d. Comme les prochaines instructions seront communes à tous les fantômes, nous exécutons les mêmes instructions autant de fois qu'il y a de fantômes souhaités avec une boucle : `for i in range (NumberOfGhosts):`.

e. Dans la boucle, charger le modèle `ghost.egg` dans une variable temporaire appelée `ghost` (différent de `self.ghosts` créée précédemment). La taille du fantôme doit être égale à `0.01`. Positionnez le fantôme aléatoirement sur l'aire de jeu (cf. question 7 avec le positionnement d'Ève). Ajouter le fantôme au graphe de scène et ajoutez le fantôme `ghost` à l'ensemble des fantômes `ghosts` avec l'instruction `self.ghosts.append(ghost)`.

f. Toujours dans la boucle, créez l'agent intelligent `ghostAI`. Ajouter le au monde IA. Maintenant, il faut préciser que cet agent fait parti du flocking créé en 17.b avec `self.flockObject. addAiChar(ghostAI)`. On ajoute cet agent intelligent `ghostAI` à l'ensemble des agents intelligents `self.ghostsAI`.

g. Comme précisé dans le manuel, le flocking est un comportement collectif qui doit être combiné avec un comportement propre à chaque agent. Ainsi, nous décidons que les fantômes poursuivent Ralph. Par conséquent, toujours dans la boucle, récupérez les comportements de l'agent `ghostAI` et affectez lui la poursuite vers Ralph.

h. Vous pouvez maintenant exécutez votre programme pour visualiser ce magnifique comportement collectif ! Testez les différents flocking de bases proposés par les auteurs, puis les vôtres.

i. Les fantômes peuvent malheureusement traverser les objets. Pour cela, nous devons leur ajouter des solides de collision. Le problème est que la méthode `setCollision()` a besoin d'être appelée après la méthode `setFlocking()` pour définir les collisions sur les fantômes mais que `setFlocking()` définit les fantômes, utilisés dans `setCollision()`.Le plus simple est de préciser que la variable `pusher`, utile pour la gestion des collisions, soit également accessible dans la méthode `setFlocking()` lors de la construction des fantômes. Pour cela, remplacer `pusher` par `self.pusher` dans la méthode `setCollisions()`. Ensuite, dans la méthode `setFlocking()`, ajouter la construction d'un nœud de collision pour chaque fantôme comme nous l'avons déjà fait pour Ralph et pour Ève dans la question 29 ou 30 du TP2.

j. Ajoutez le comportement obstacle avoidance sur les ghosts pour éviter qu'ils traversent les objets de la scène.

18. Nous souhaitons ajouter plusieurs tables dans la scène afin de donner une impression de labyrinthe et afin de mieux tester les différents comportements. Pour cela, nous allons modifier la méthode `loadModels` de la façon ci-contre. Tout d'abord, on crée une liste vide nommée `tables`. On répète 20 fois la création d'une table qu'on ajoute au graphe de scène. Cette fois-ci, on utilise un nombre aléatoire pour savoir comment on oriente la table : verticale ou horizontale. Bien évidemment, le reste du code devra être modifié car le programme ne connait plus table mais tables. En particulier, dans la méthode `setCollisions()`, il faudra ajouter chaque table. Pour parcourir l'ensemble des tables, il suffit d'utiliser l'instruction `for table in tables:` et utiliser la variable `table` plutôt que `self.table` dans la boucle.

```
self.tables=[]
for t in range(20):
    table = self.loader.loadModel("models/table")
    table.setPos(random.randint(-self.LimiteTerrain,self.LimiteTerrain),
                 random.randint(-self.LimiteTerrain,self.LimiteTerrain),0)
    table.setScale(2,1,1)
    if random.random() > 0.5:
        table.setH(90)
    table.reparentTo(self.render)
    self.tables.append(table)
```

Pour aller plus loin

19. En vous inspirant des exemples fournis par Panda3D dans le Pathfinding par exemple, rendez la gestion des touches clavier plus fluide qu'elle ne l'est.

20. Créez un terrain 3D plutôt que 2D et faites évoluer tous les personnages sur ce terrain accidenté.

21. Imaginez ajouter des petites et des grandes tables (même modèle géométrique mais de hauteurs différentes) dans l'aire de jeu. Ève possède une animation de saut. L'idée est d'autoriser Ève à sauter sur les petites boites mais pas sur les grandes pendant sa poursuite vers Ralph.

Sources pour ce TP

Panda3D : http://www.panda3d.org/

Librairie PandAI : http://www.etc.cmu.edu/projects/pandai/

Modèles 3D gratuit (Objets, Animaux, Humains …) : http://alice.org/pandagallery/

"Steering Behaviors for Autonomous Characters", Craig Reynolds, Game Developers Conference 1999, http://www.red3d.com/cwr/steer/