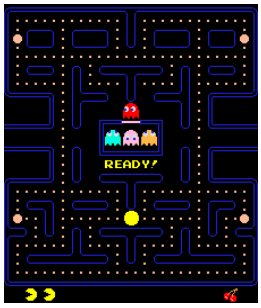
TP4 : Finite State Machines

Objectif du TP

La gestion de l'intelligence de PNJ peut s'avérer complexe au fur et à mesure que le nombre de PNJ augmente. En particulier, ce sont les différentes interactions entre les différents PNJ et les différentes actions possibles avec l'environnement qui complexifient cette gestion. Ainsi, pour la simplifier, les machines à états finis (Finite State Machines) ou automates finis ont été introduit. Cette notion est déjà implémentée dans Panda3D.

Conception de FSM

1. Depuis le deuxième TP, nous avons simulé le fonctionnement d'un tout petit automate. Identifiez ce qui correspondrait à cette pseudo-programmation. Dessinez (sur papier) l'automate correspondant.
2. Il est évident que pour des automates aussi simples, le programmeur ne va pas utiliser la grosse artillerie avec les FSM. Toutefois, vous connaissez sûrement le jeu du Pacman (cf. ci-contre pour un petit rappel). Dessinez l'automate correspondant à un fantôme ! Pour vous mettre sur la voie, on peut identifier au moins 5 états et 7 transitions ; ce qui commence sérieusement à compliquer le développement du jeu.



Déplacement Réaliste

3. Dans le TP2, nous avons souhaité avoir un déplacement plus fluide de Ralph. Pour cela, nous allons modifier notre programme de la façon suivante : nous allons créer une structure de données qui va conserver les événements sur les touches du clavier et nous allons définir une tâche (comme nous l'avions vu précédemment pour le déplacement de la caméra) qui s'occupera de déplacer Ralph en fonction des valeurs conservées dans la structure de données. Contrairement à précédemment, le joueur ne pourra utiliser que 3 touches de direction (gauche, droite, vers le haut pour aller en avant) ; il ne pourra pas reculer. De plus, nous ne gérerons que la marche pour l'instant et pas la course. Ainsi :
    a. Dans le constructeur, définissons une structure de données (appelée *dictionnaire*) qui associe une valeur à une chaine de caractères avec `self.keyMap = {"left":0, "right":0, "forward":0}` pour savoir si la touche a été appuyée ou pas. Le dictionnaire contient alors 3 clés nommées `left`, `right` et `forward` contenant toutes la valeur `0`.
    b. Nous allons définir une méthode `saveKey` qui modifie la valeur d'une clé du dictionnaire :

```
#Records the state of the arrow keys
def saveKey(self, key, value):
    self.keyMap[key] = value
```

    c. Nous allons modifier l'ancienne méthode `setKeys` pour ne gérer que les 3 touches et ne réagir qu'à l'appui ou au relâchement d'une touche. Chaque fois qu'une action sur une touche est effectuée, nous modifions le dictionnaire (cf. ci-contre).

```
def setKeys(self):
    #Allow to quit application with Escape key
    self.accept('escape', sys.exit)
    #Define what to do with arrow keys
    self.accept("arrow_left", self.saveKey, ["left",1])
    self.accept("arrow_right", self.saveKey, ["right",1])
    self.accept("arrow_up", self.saveKey, ["forward",1])
    self.accept("arrow_left-up", self.saveKey, ["left",0])
    self.accept("arrow_right-up", self.saveKey, ["right",0])
    self.accept("arrow_up-up", self.saveKey, ["forward",0])
```

    d. Dans le constructeur, nous allons ajouter au gestionnaire des tâches une méthode qui va s'occuper de modifier la position de Ralph à chaque fois qu'il y un nouvel affichage graphique en fonction des valeurs des touches stockées dans le dictionnaire avec l'instruction `self.taskMgr.add(self.moveRalph, "moveRalph")`.

e. Suite à l'instruction précédente, il est nécessaire d'ajouter une nouvelle méthode `moveRalph` qui va s'occuper de déplacer Ralph (cf. ci-contre). Du coup, vous pouvez commentez (ou supprimer cette fois-ci) la méthode `DeplaceRalph` car nous ne devrions plus en avoir besoin.

```
def moveRalph(self, task):
    # Get the time elapsed since last frame. We need this
    # for framerate-independent movement
    elapsed = globalClock.getDt()
    # If a move-key is pressed, move ralph in the specified direction.
    if (self.keyMap["left"]!=0):
        self.ralph.setH(self.ralph.getH() + elapsed*self.VitesseRotationRalph)
    if (self.keyMap["right"]!=0):
        self.ralph.setH(self.ralph.getH() - elapsed*self.VitesseRotationRalph)
    if (self.keyMap["forward"]!=0):
        self.ralph.setY(self.ralph, -(elapsed*self.VitesseDeplacementRalph))
    return Task.cont
```

4. Maintenant que Ralph ne tourne pas brusquement, nous pouvons avoir une vision de type FPS en positionnant la camera au niveau des yeux de Ralph, un peu derrière lui et orienté un peu vers le bas (cf. question 25 du TP2). Il faudra également positionner la camera relativement à Ralph. Ceci nous permettra d'avoir un déplacement plus réaliste que dans les précédents TPs.

Picture In Picture

5. Comme nous ne voyons pas derrière nous, nous allons incruster en haut à droite de l'écran une image qui correspond à la vision arrière de Ralph. Pour cela, créez une nouvelle méthode nommée `pip` comme ci-contre.

```
def pip(self):
    self.backCam = base.makeCamera(base.win, displayRegion = (0.89, 0.99, 0.89, 0.99))
    self.backCam.reparentTo(self.ralph)
    self.backCam.setZ(5)
    self.backCam.node().setActive(False)
```

6. La dernière ligne de la fonction `pip` désactive l'incrustation car nous souhaitons afficher le PIP seulement lors de l'appui sur la touche 'p'. Modifiez la méthode `setKeys` pour gérer la touche p. Il sera nécessaire de créer une autre méthode que l'on nommera `activatePip` qui aura pour objectif d'activer ou de désactiver l'incrustation. Pour connaitre l'état (activé ou non) de la caméra, on pourra utiliser la méthode `self.backCam.node().isActive()`.

7. Dans ce qui suit, nous n'avons pas besoin d'Ève. Le plus simple est de ne pas l'afficher. Pour cela, commentez la ligne qui ajoute ève au graphe de scène.

Notre Pacman revisité : RalphMan

8. L'idée est de commencer à programmer l'automate d'un fantôme du classique jeu Pacman. Ainsi, nous allons transformer la méthode `setFlocking()`.

   a. Créez un nouveau fichier texte `GhostFSM.py`.

   b. Ajoutez le module FSM de Panda3D avec `from direct.fsm.FSM import FSM`.

   c. Créez un nouveau type de variable objet (une classe) qui hérite de toutes les fonctionnalités du type de données FSM définie dans Panda3D avec `class GhostFSM(FSM):`.

   d. Ajoutons le constructeur de ce nouveau type d'objet en nous inspirant du code du TP2 pour le type `MyFirstGame` qui héritait de `ShowBase`. Nous allons définir une méthode `setNodePath()` qui stockera la géométrie du fantôme (`nodepath`) passé en paramètre ainsi que le graphe de scène (`render`). Une autre méthode `getNodePath()` renvoie la géométrie du fantôme qui ne sera plus enregistrée dans le programme principal.

```
class GhostFSM(FSM):
    def __init__(self): #optional because FSM already defines __init__
        #if you do write your own, you *must* call the base __init__!
        FSM.__init__(self, 'GhostFSM')

    def setNodePath(self,nodepath,render):
        #define the geometry
        self.ghost=nodepath
        self.render=render
        self.ghost.setScale(0.01)
        self.ghost.reparentTo(render)

    def getNodePath(self):
        return self.ghost
```

e. De cette façon, la méthode `setFlocking()` devra être modifiée pour prendre en compte ce nouveau type d'objet. En effet, il faut créer un objet de type `GhostFSM`, créer la géométrie du fantôme et passer en paramètre le `render` de `ShowBase`. Ensuite, il faut positionner le fantôme aléatoirement en récupérant sa géométrie depuis l'objet créé précédemment.

```
def setFlockingFSM(self):
    self.flockObject = Flock(1, 270,10,2,4,1)
    self.AIworld.addFlock(self.flockObject)
    self.ghosts=[]

    for i in range (self.NumberOfGhosts):
        ghost = GhostFSM()
        ghost.setNodePath(self.loader.loadModel("models/ghost"),self.render)
        ghost.getNodePath().setPos(random.randint(-self.LimiteTerrain,self.LimiteTerrain),
                                   random.randint(-self.LimiteTerrain,self.LimiteTerrain),0)
```

f. Nous faisons de même en créant la méthode `setAI(self, AIworld)` dans `GhostFSM` qui stocke le monde des agents intelligents (`AIworld`), définit un nouvel agent (le fantôme) et le stocke. Une autre méthode `getAI(self)` retourne l'agent intelligent. Inspirez vous du contenu de la méthode `setFlocking` pour programmer ces deux fonctions (copier-coller intelligent).

g. Modifiez également la méthode `setFlocking()` pour prendre en compte ces modifications dans `GhostFSM` sans toucher à la gestion des collisions.

h. Dorénavant, il reste à définir les différents états de l'automate `GhostFSM`. En réalité, ils sont définis implicitement en créant des méthodes nommées `enter` et/ou `exit` suivi du nom de l'état avec pour convention une majuscule à la première lettre de l'état. Par exemple, si on considère l'état correspondant au déplacement aléatoire (`RandomMove`), la méthode `enterRandomMove(self)` définit dans `GhostFSM` contiendra les instructions qui définissent l'entrée de l'automate dans cet état, et la méthode `exitRandomMove(self)` les instructions permettant de revenir exactement au même comportement avant l'entrée dans l'état. Pour l'entrée dans l'état `RandomMove`, il faut avant tout récupérer les comportements de cet agent avec `AIGhostBehav=self.ghostAI.getAiBehaviors()`. En étudiant la documentation de la classe `AIBehaviors`, on constate qu'il est possible de connaître le statut d'un comportement (méthode `behaviorStatus` qui renvoi `disabled`, `active` ou `paused`), de stopper un comportement (méthode `pauseAi('nom_du_comportement')`) ou de le relancer (méthode `resumeAi('nom_du_comportement')`). Ainsi, quand on entre dans l'état `RandomMove`, il faut vérifier qu'il n'a pas déjà été défini (`AIGhostBehav.behaviorStatus('wander')=='disabled'`); et si ce n'est pas le cas, il faut le définir sinon le relancer avec `resumeAi('wander')`.

i. Dans la méthode `exitRandomMove(self)`, il faut également récupérer les comportements du fantôme avec `self.ghostAI.getAiBehaviors()`. Ensuite, il faut vérifier que le comportement `wander` est bien actif avec `AIGhostBehav.behaviorStatus('wander')=='active'` ; si c'est bien le cas, il faut mettre en pause ce comportement avec `AIGhostBehav.pauseAi('wander')`.

j. Inspirez vous des questions 8.h et 8.i pour définir les méthodes `enter` et `exit` des comportements `pursue` et `evade`. À la différence de `wander`, il est nécessaire de préciser un `NodePath` qu'il faut fuir ou suivre. En réalité, il est possible d'ajouter des paramètres aux méthodes `enter` et `exit` de la façon suivante : `def enterEvade(self, nodepath):`. Par conséquent, si on souhaite que les fantômes poursuivent Ralph, il faut préciser dans la méthode `setFlocking()` que le fantôme se trouve dans cet état avec `ghost.request('Pursue',self.ralph)`.

k. Testez votre programme.

l. À l'aide de la touche `'g'`, nous souhaitons simuler le fait que Ralph a mangé une super pac-gomme qui le rend invincible et qui lui permet de manger les fantômes ou que la super pac-gomme a terminé son effet. Supposons que la méthode activée par la touche s'appelle `pacGomme(self)` ; il est

possible de connaitre l'état courant de l'automate avec `ghost.state`. Ainsi, si les fantômes sont en train de poursuivre Ralph alors ils doivent le fuir et inversement. Programmez la méthode `pacGomme`.

UN PEU DE SONS

9. Afin de vous montrer que la prise en compte d'automate peut être très utile, nous souhaitons ajouter un son à chaque fois qu'un fantôme change d'état. Rien de plus simple : dans le constructeur de GhostFSM, ajoutons le chargement d'un son bien précis avec l'instruction : `self.mysound = base.loader.loadSfx("sounds/pacman_eat_gum.wav")`.

10. Ensuite, il existe une méthode intitulée `defaultFilter` qui est appelée à chaque fois qu'il y a un changement d'état valide. Dans cette méthode, nous pouvons jouer le son chargé précédemment dans le constructeur (cf. ci-contre).

```
def defaultFilter(self, request, args):
    self.mysound.play()
    return request
```

Pour aller plus loin

11. Gérer les collisions pour que Ralph puisse manger un monstre : ajout un état (mort) à partir duquel certaines transitions ne sont pas possibles …

Sources pour ce TP

Panda3D : http://www.panda3d.org/
Librairie PandAI : http://www.etc.cmu.edu/projects/pandai/