Compte rendu

Pacman en python

Explication du code

__Init__

Dans la fonction __init__ sont initialisés tous les paramètres de la partie.
Dans un premier, on ajoute un attribut contenant la liste des robots :

```
self.les_robots = []
self.numero_robot = 0
self.les_robots.append(wxRUR.Robot(self, col=1, row=1, orient='E', name="pacman", colour="light blue"))
self.les_robots.append(wxRUR.Robot(self, col=10, row=1, orient='W', name="pacman", colour="blue"))
self.les_robots.append(wxRUR.Robot(self, col=1, row=10, orient='E', name="pacman", colour="purple"))
self.les_robots.append(wxRUR.Robot(self, col=10, row=10, orient='W', name="pacman", colour="green"))
self.les_robots.append(wxRUR.Robot(self, col=1, row=6, orient='E', name="pacman", colour="grey"))
```

Les robots sont chacun initialisés à la position demandée et ajoutée à la liste des robots (attribut de la partie) . Pour créer un robot on utilise la librairie wxRUR et sa fonction Robot().

Ensuite, on peut créer un attribut pour le pacman, il sera initialisé avec la même fonction que les autres robots mais l'attribut colour doit être égal à pacman.
On ajoutera à ce pacman différent attributs pour stocké les cases parcourues et faire en sorte que son parcours soit affiché.

```
self.pacman = wxRUR.Robot(self, col=6, row=6, orient='E', name="pacman", colour="pacman")
self.pacman.cases_parcourues = []
self.pacman.cases_parcourues.append(self.pacman.getPos())
self.pacman.set_trace_style(style=-1,colour='black')
```

Ensuite, on demande alors au joueur de saisir la vitesse de la partie (rapidité des robots), et la durée de celle-ci .

```
self.vitesse = self.inputInt(text="Entrer la vitesse de la partie : ( Entier entre 1 et 9)", title = "Vitesse des robots")
while self.vitesse not in range(1,10):
    self.vitesse = self.inputInt(text = "Entrer la vitesse de la partie : ( Entier entre 1 et 9)", title = "Vitesse des robots")
self.vitesse = 1.0/vitesse
 self.duree = self.inputInt(text ="Entrer la duree limite de la partie : ( Entier entre 30 a 240)", title = "Duree de la partie")
while self.duree not in range(30,241):
    self.duree = self.inputInt(text = "Entrer la duree limite de la partie : ( Entier entre 30 a 240)", title = "Duree de la partie"
self.TempEcoule = 0
```

Un attribut TempsEcoule sera mis à zéro et permettra de connaître le temps écoulé depuis le début de la partie.

Maintenant, il faut pouvoir réagir aux clics sur les flèches du clavier. Pour cela, on utilise la fonction KeyEventHandler() avec en paramètre la fonction à appeler.
Pour faire bouger les robots (autonomes) on utilisera une alarme (un signal) qui en fonction de la vitesse saisie appel la fonction pour déplacer un robot. On fera de même avec la fonction score .

```
# Association de la méthode 'deplacer_pacman'
# à l'évènement "KeyEvent" (appui sur une touche)
self.KeyEventHandler(onKeyEvent=self.deplacer_pacman)
self.TimerEventHandler(onTimerEvent=self.deplacer_un_robot, TimerPeriod = self.vitesse)
self.TimerEventHandler(onTimerEvent=self.score, TimerPeriod= 0.1)
```

On lancera la partie avec la fonction MainLoop() et en paramètre la durée saisie.

```
# Boucle de gestion des evenements de l'application
self.MainLoop(   duration =  self.duree  )
```

Deplacer_pacman

Cette fonction a pour but de déplacer le pacman en fonction des clics de l'utilisateur.
Tout d'abord, il faut vérifié si le pacman a parcouru toutes les cases, si c'est le cas alors c'est une victoire. On l'affiche à l'utilisateur et on arrête le jeu.

```
if(len(self.pacman.cases_parcourues)==100):
    wxRUR.Application.ShowMessage(self, text="Gagne !!! Tu as toutes les cases", title="fin de la partie")
    wxRUR.Application.ExitMainLoop(self)
```

Sinon, il faut récupérer le clic de l'utilisateur avec la fonction GetKeyCode(), regarder si il correspond à une direction. Si c'est le cas alors on regarde si il n'est pas confronté à un mur avec la fonction is_clear(). Aucun mur, alors on tourne le pacman avec turn() puis on le fait avancer avec move().

```
else:
    directionsPossible = {315:'N',314:'W',317:'S',316:'E'}
    direction = KeyEvent.GetKeyCode()
    if(directionsPossible.has_key(direction)):
        direction = directionsPossible[direction]
        if(self.pacman.is_clear(direction)):
            self.pacman.turn(direction)
            self.pacman.move()
```

Maintenant le pacman est sur une nouvelle case, on peut donc ajouter cette case à celles parcourus mais seulement si elle n'y ait pas encore

```
if(self.pacman.cases_parcourues.count(self.pacman.getPos())==0):
    self.pacman.cases_parcourues.append(self.pacman.getPos())
```

Ensuite on vérifie qu'il n'entre pas en collision avec un robot, pour cela on va récupérer la position de chaque robot et la comparer à celle du pacman si ce sont les mêmes alors c'est perdu.

```
for robot in self.les_robots:
    if robot.getPos() == self.pacman.getPos() :
        if len(self.pacman.cases_parcourues)!=100 :
            wxRUR.Application.ShowMessage(self, text="Perdu !!! Tu a rencontre un robot", title="fin de la partie")
            wxRUR.Application.ExitMainLoop(self)
```

deplacer_un_robot

Cette fonction a pour but de déplacer le robot dans la direction qui l'approchera le plus du pacman Elle fera d'abord appelle à le fonction direction_poursuite() (décrite ci dessous) pour connaître la direction à suivre, le robot se tourne et avance comme le pacman. Si le robot se trouve sur la même case que le pacman alors c'est perdu.

```
def deplacer_un_robot(self,TimerEvent):
    direction = self.direction_poursuite(self.les_robots[self.numero_robot])
    self.les_robots[self.numero_robot].turn(direction)
    self.les_robots[self.numero_robot].move()
    if self.pacman.getPos() == self.les_robots[self.numero_robot].getPos() :
        wxRUR.Application.ShowMessage(self, text="Game over : Les robots ont eu raison de vous !!!", title="fin de la partie")
        wxRUR.Application.ExitMainLoop(self)
    self.numero_robot = (self.numero_robot+1)%len(self.les_robots)
```

direction_poursuite

Cette fonction a pour but de connaître la direction qui rapprochera le plus le robot du pacman, on test toutes les directions, sauf celle ou se trouve un mur, avec la méthode distance_ au_pacman et on applique une simple recherche du maximum.

```
def direction_poursuite(self,robot):
    distanceMax = 0
    bestDirection = None
    for direction in self.deplacement.keys() :
        if robot.is_clear(direction) :
            distanceTest = self.distance_au_pacman(robot, direction)
            if distanceTest < distanceMax or bestDirection == None:
                distanceMax = distanceTest
                bestDirection = direction
    return bestDirection
```

distance_au_pacman

Cette fonction a pour but de retourner la distance entre le robot et le pacman dans une direction donnée. C'est une distance à vol d'oiseau.

```
def distance_au_pacman(self,robot,direction):
    xp,yp = self.pacman.getPos()
    xr,yr = robot.getPos()
    coordonnex,coordonney = self.deplacement[direction]
    newxr,newyr = xr+coordonnex, yr+coordonney
    distance = math.sqrt((xp-newxr)*(xp-newxr)+(yp-newyr)*(yp-newyr))
    return distance
```

Score

Cette fonction a pour but d’afficher et mettre à jour le score de la partie et de vérifier si le temps limité est atteint.

```
score(self,TimerEvent):
temps = self.duree - self.TempEcoule
self.TempEcoule += 1
score = score = 3*self.vitesse*len(self.pacman.cases_parcourues)
self.SetStatusText("Temps restant : " + str(temps-1) + " Nb cases parcourues : " + str(len(self.pacman.cases_parcourues) ) + " score : " + str(score) )
if(temps == 1):
    wxRUR.Application.ShowMessage(self, text="Game over : Le temps a eu raison de vous", title="fin de la partie")
    wxRUR.Application.ExitMainLoop(self)
```