

Oh Mummy

par [Guillaume DuriAUD](#) ([Page personnelle](#))

Date de publication : 29/03/2007

Dernière mise à jour : 07/04/2007

- Développement de jeux en *Python*
- Découverte de la bibliothèque *Pygame*
- Développement du jeu **Oh Mummy**

- I - Prérequis
- II - Introduction
- III - Le jeu
- IV - Explication du code
 - IV-A - Importation et Initialisation
 - IV-B - Chargement des images
 - IV-C - Les différentes classes
 - IV-C-1 - Base commune des classes
 - IV-C-2 - Classe Gargou
 - IV-C-3 - Classe Life
 - IV-C-4 - Classe Score
 - IV-C-5 - Classe Mummy
 - IV-C-6 - Classe GuardianMummy
 - IV-C-7 - Classe Plate
 - IV-D - Fonction main()
- V - Conclusion
- VI - Téléchargement

I - Prérequis

Voici les langages et bibliothèques que j'ai utilisés pour développer Oh Mummy:

Système d'exploitation: Windows XP

Langage:  **Python 2.5**

Bibliothèque:  **Pygame 1.7.1**

II - Introduction

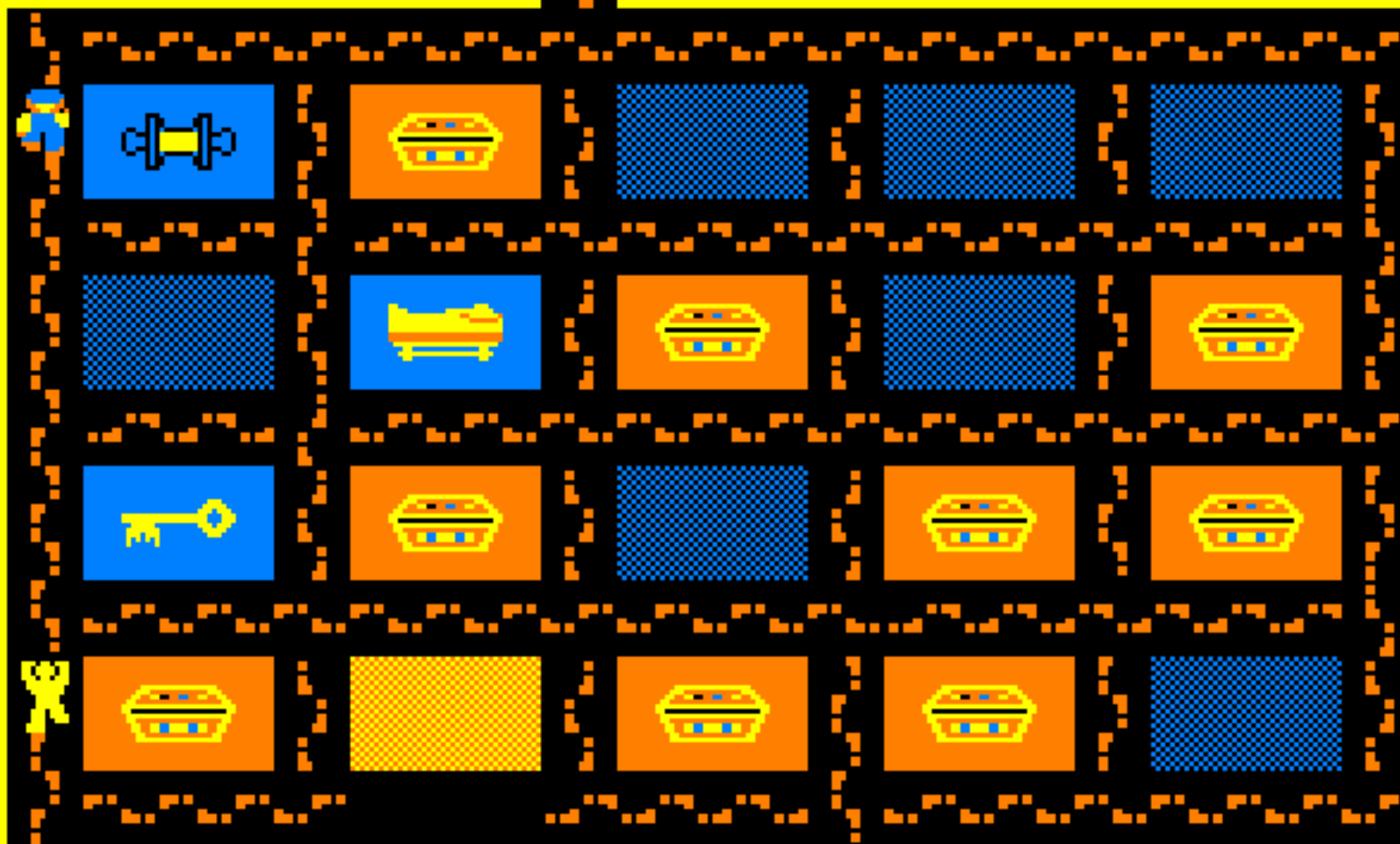
Mon objectif premier était de découvrir la bibliothèque **Pygame**. J'ai développé ce jeu sans aucune connaissance préalable de **Pygame**. Il m'a fallu une dizaine d'heures pour arriver à la version 1.0 qui ne doit plus à priori présenter de bogues. Vous pouvez donc utiliser ce document pour appréhender **Pygame**. De mon côté, je me suis avant tout servi du source **aliens.py** (disponible dans l'installation de **Pygame**) qui est très simple à comprendre pour apprendre les instructions de bases du module **Pygame**.

Ainsi, j'ai essayé d'être le plus efficace possible sans forcément obtenir un code très propre (j'ai utilisé de nombreuses variables globales utiles dans tout le programme pour pouvoir y accéder très simplement). De plus, j'ai utilisé, je pense, les instructions minimales fournies par **Pygame** sans chercher forcément à en utiliser d'autres plus performantes.

Au niveau du graphisme, j'ai repris exactement le graphisme que l'on pouvait voir sur les CPC de l'époque en ayant avant tout créé toutes les images qui auraient besoin d'être incorporées dans le programme. En revanche, je n'ai pas intégré le son.

SCORE 00095

MEN



III - Le jeu

Je pense que tout le monde connaît ce jeu et sait y jouer, ce n'est vraiment pas compliqué. je vous laisse avec l'explication originelle.

You have been appointed head of an archeological expedition, sponsored by the British Museum, and have been sent to Egypt to explore newly found pyramids.

Your party, initially, consists of five members. Your task is to enter the five levels of each pyramid, and recover from them five Royal Mummies and as much treasure as you can.

The partly excavated levels are in the form of a grid made up of twenty 'boxes'.

To uncover a 'box', move your team along the four sides of the box from each corner to the next.

Not all boxes need to be uncovered to enable you to go through the Exit and into the next level.

The Magic Scroll will allow you to be caught by a Guardian, without any harm to your team. The Scroll works only on the level on which found, it will only destroy one Guardian.

There are two ways to gain points, one is by uncovering the Royal Mummy the other, by uncovering Treasure.

When you have completed a pyramid, your success will be rewarded either by bonus points or the arrival of an extra member for your team.

The Guardians in the next pyramid, having been warned by those you have escaped from, will be more alert, so although the Guardians cannot follow you from one pyramid to the next, it will pay to be even more careful.

Each level has already been partly uncovered by local workers and it is up to your team to finish the dig.

Unfortunately, the workers digging aroused Guardians left behind by the ancient Egyptian Pharoahs to protect their royal tombs.

Each level has 2 Guardian Mummies, one lies hidden while the other goes in search of the intruders.

Each level contains, ten Treasure boxes, six empty boxes, and the rest hold a Royal Mummy, a Guardian Mummy a Key and a Scroll.

If you uncover the box holding the Guardian Mummy, it will dig it's way out and pursue you. Being caught by a Guardian Mummy kills one member of your team and the Mummy, unless that is, you have uncovered the Scroll.

When the boxes holding the Key and the Royal Mummy have been uncovered, you will be able to leave the level. Any remaining Guardians will be able to follow you onto the next level.

After completing all 5 levels of a pyramid you will, when you leave the fifth level, move to level 1, of the next pyramid.

IV - Explication du code

IV-A - Importation et Initialisation

Au niveau des importations, en général, on se contente de celles qui suivent:

```
import pygame
import pygame.image
from pygame.locals import *
pygame.init()
pygame.font.init()
```

L'écran du jeu est créé par la fonction **`pygame.display.set_mode((largeur, hauteur))`**. On peut aussi changer le titre de la fenêtre par l'instruction **`pygame.display.set_caption(MonTitre)`**

```
screen = pygame.display.set_mode((768, 544))
pygame.display.set_caption("Oh Mummy")
```

IV-B - Chargement des images

Pour charger une image, il suffit d'appeler la fonction **`pygame.image.load(imagepath)`**. J'ai décidé pour plus de clarté de charger toutes mes images dans un dictionnaire **`ImageOhMummy`**.

- **`ImageOhMummy['Gargou']`** contient les différentes positions que le héros peut prendre
- **`ImageOhMummy['Mummy']`** contient les différentes positions qu'une momie peut prendre
- **`ImageOhMummy['GuardianMummy']`** contient la séquence d'apparition de la momie cachée
- **`ImageOhMummy['Trace']`** contient les différentes traces que peut laisser le héros sur le sol
- **`ImageOhMummy['Box']`** contient les différentes découvertes que le héros peut faire
- **`ImageOhMummy['Chiffre']`** contient les chiffres pour afficher le score
- **`ImageOhMummy['ChiffreScroll']`** contient les chiffres pour afficher le score quand le héros possède le parchemin

```
ImageOhMummy = {}
ImageOhMummy['Gargou'] = {}
ImageOhMummy['Gargou']['Left'] = pygame.image.load("./Image/GargouLeft.png")
...
ImageOhMummy['Mummy'] = {}
ImageOhMummy['Mummy']['Left'] = pygame.image.load("./Image/MummyLeft.png")
...
ImageOhMummy['GuardianMummy'] = {}
for i in range(1, 17):
    ImageOhMummy['GuardianMummy'][i] = pygame.image.load("./Image/GuardianMummy"+str(i)+".png")
...
ImageOhMummy['Trace'] = {}
ImageOhMummy['Trace']['None'] = pygame.image.load("./Image/TraceNone.png")
ImageOhMummy['Trace']['LeftDown'] = pygame.image.load("./Image/TraceLeftDown.png")
...
ImageOhMummy['Box'] = {}
ImageOhMummy['Box']['RoyalMummy'] = pygame.image.load("./Image/RoyalMummy.png")
...
ImageOhMummy['Chiffre'] = {}
```

```
ImageOhMummy['ChiffreScroll'] = {}
ImageOhMummy['Score'] = pygame.image.load("./Image/Score.png")
ImageOhMummy['Men'] = pygame.image.load("./Image/Men.png")
ImageOhMummy['EndLife'] = pygame.image.load("./Image/EndLife.png")
ImageOhMummy['End200'] = pygame.image.load("./Image/End200.png")
ImageOhMummy['GameOver'] = pygame.image.load("./Image/GameOver.png")
for i in range(10):
    ImageOhMummy['Chiffre'][i] = pygame.image.load('./Image/Chiffre'+str(i)+".png")
    ImageOhMummy['ChiffreScroll'][i] = pygame.image.load('./Image/ChiffreScroll'+str(i)+".png")
```

IV-C - Les différentes classes

IV-C-1 - Base commune des classes

Les différentes classes présentées à la suite ont toutes une base commune. Tout d'abord, elles dérivent toutes de la classe **pygame.sprite.Sprite**. De ce que j'en ai compris, cette classe permet de définir un objet visuable à l'écran. La dérivation se fait de la manière suivante:

pygame.sprite.Sprite.__init__(self, self.containers) où **self.containers** est un tuple contenant les *groupes de sprites* (dans notre cas des **RenderUpdates** (cf la fonction **main()**))

Dans la création de la classe, il y a ensuite 2 attributs à définir obligatoirement:

L'attribut **image** contient l'image de votre objet. Il peut par exemple correspondre à une image préalablement chargée ou bien à une surface vide (**pygame.Surface((largeur, hauteur))**) que l'on pourra remplir avec la méthode **fill(R, G, B)** ou **blit(img, (x, y))**.

L'attribut **rect** définit en général le rectangle englobant **image**. On se contente en général de le créer de la façon suivante: **self.rect = self.image.get_rect()**. On pourra ensuite positionner ce rectangle dans la scène en modifiant par exemple les attributs **top** et **left** de **rect**.

Enfin, pour détruire un objet dérivant de la classe **pygame.sprite.Sprite**, il suffira de lui appliquer la méthode **kill()**.

IV-C-2 - Classe Gargou



La classe **Gargou** permet de gérer le héros.

Les attributs **x** et **y** donne la position de héros dans le tableau représentant le plateau (cf la classe **Plate**). Le personnage prenant 4 cases de ce tableau, les composantes **x** et **y** correspondent à la position en haut à gauche du héros.

L'attribut **facing** donne le sens dans laquelle se déplace le héros (0: à droite, 1: en haut, 2: à gauche, 3: en bas). L'attribut **moving** gère les 2 positions possibles pour chaque sens de déplacement.

La fonction **move** gère le déplacement du personnage en fonction de la flèche sur laquelle vous avez appuyé. La fonction vérifie tout d'abord la validité du déplacement. Chaque fois que le personnage effectue un déplacement valide, il laisse derrière lui une trace de pas qui va permettre de dégager les boîtes. Cela sera géré par l'appel à

PLATE.Update(...). A chaque déplacement, en fonction de la valeur des attributs **facing** et **moving**, on met à jour l'attribut **image** par l'image correspondante.

Enfin, l'attribut **level** donne le niveau du héros et permet de définir l'algorithme de déplacement des momies. A chaque fois que le héros finit la série des 5 niveaux d'une pyramide, cet attribut augmente de 1.

Quand le héros a récupéré la clé et les restes de la momie, il peut changer de niveau en remontant à l'entrée initiale (condition à vérifier: **PLATE.Key and PLATE.RoyalMummy and self.x == 15 and self.y == 1**)

Une unique instance de cette classe est créée et est globale au jeu nommée **GARGOU**

```
class Gargou(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self, self.containers)
        self.image = ImageOhMummy['Gargou']['MoveDownRight']
        self.rect = self.image.get_rect()
        self.x = 15
        self.y = 1
        self.facing = 0
        self.moving = 0
        self.rect.left = 320
        self.rect.top = 94
        self.level = 1

    def move(self, direction):
        try: direct = direction.index(1)
        except: return

        self.facing = direct

        x = self.x; y = self.y
        if direct == 0: x = self.x + 1
        elif direct == 1: y = self.y - 1
        elif direct == 2: x = self.x - 1
        elif direct == 3: y = self.y + 1

        if PLATE.plate[y][x] == PLATE.plate[y][x+1] == PLATE.plate[y+1][x] == PLATE.plate[y+1][x+1]
== 1:
            self.x = x
            self.y = y
        else: return

        PLATE.Update(self.x, self.y, self.facing, self.moving)
        if PLATE.Key and PLATE.RoyalMummy and self.x == 15 and self.y == 1:
            PLATE.Level = max(1, (PLATE.Level + 1) % 6)
            PLATE.ChangeLevel = True
        self.moving = (self.moving + 1) % 2
        self.image = ImageOhMummy['Gargou']['Face'][2 * self.facing + self.moving]
        self.rect.left = 80 + 16 * self.x
        self.rect.top = 78 + 16 * self.y
```

IV-C-3 - Classe Life



La classe **Life** permet de gérer à la fois le nombre restant de vies du héros et son affichage à l'écran. La fonction **Update** permet de réafficher à l'écran les vies restantes du héros quand ce nombre change.

Une seule instance de cette classe sera créée en tant que variable globale et nommée **LIFE** pour une manipulation plus aisée.

```
class Life(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self, self.containers)
        self.life = 5 ## Le héros démarre avec 5 vies
        self.image = pygame.Surface((320, 32))
        self.rect = self.image.get_rect()
        self.rect.top = 78
        self.rect.left = 416

    def Update(self):
        self.image.fill((255, 255, 0))
        self.image.blit(ImageOhMummy['Men'], (0, 0))
        for i in range(0, self.life, 2):
            self.image.blit(ImageOhMummy['Gargou']['MoveRight'], (64 + 32* i, 0))
        for i in range(1, self.life, 2):
            self.image.blit(ImageOhMummy['Gargou']['Right'], (64 + 32* i, 0))
```

IV-C-4 - Classe Score

SCORE 00390

La classe **Score** permet de gérer le score du héros ainsi que son affichage à l'écran. Il y a de même une unique fonction **Update** appelée chaque fois que le score change. Si le héros a découvert le parchemin et qu'il ne l'a pas encore utilisé, le score est affiché en inversant les couleurs jaune et bleu.

Une seule instance de cette classe sera créée en tant que variable globale et nommée **SCORE** pour une manipulation plus aisée.

```
class Score(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self, self.containers)
        self.score = 0
        self.image = pygame.Surface((176, 16))
        self.image.fill((255, 255, 0))
        self.rect = self.image.get_rect()
        self.rect.top = 78
        self.rect.left = 96
        self.image.blit(ImageOhMummy['Score'], (0, 0))
        for i in range(5):
            self.image.blit(ImageOhMummy['Chiffre'][0], (96 + 16* i, 0))

    def Update(self):
        scorestr = str(self.score).zfill(5) ## Récupération du score sous forme d'une chaînes de
        caractères de longueur 5
        b = 0
        for i in scorestr:
            if PLATE.Scroll: self.image.blit(ImageOhMummy['ChiffreScroll'][int(i)], (96+16 * b, 0))
            else: self.image.blit(ImageOhMummy['Chiffre'][int(i)], (96+16 * b, 0))
            b += 1
```

IV-C-5 - Classe Mummy



La classe **Mummy** permet de gérer les momies (mouvement et affichage à l'écran). Toutes les momies du jeu sont contenues dans la liste globale **MUMMY**. Une instance de **Mummy** possède les mêmes attributs qu'une instance de **Gargou** à savoir **x**, **y**, **facing** et **moving** et ont la même signification.

L'attribut supplémentaire **movable** permet de gérer la vitesse de déplacement de la momie pour que celle-ci ne se déplace pas trop vite.

On distingue ici 3 fonctions **movevalable**, **move** et **Initialize**.

La fonction **Initialize** permet de repositionner chaque momie encore vivante au changement de niveau.

La fonction **movevalable** retourne toutes les cases sur laquelle une momie peut se déplacer à partir de sa position.

La fonction **move** gère le déplacement de la momie. Ce déplacement va dépendre du niveau atteint par le héros. Les momies seront ainsi de plus en plus "intelligentes" et chercheront à force à se déplacer en direction du héros.

```
class Mummy(pygame.sprite.Sprite):
    def __init__(self, x, y):
        pygame.sprite.Sprite.__init__(self, self.containers)
        self.image = ImageOhMummy['Mummy']['MoveRight']
        self.rect = self.image.get_rect()
        self.x = x
        self.y = y
        self.facing = 0
        self.moving = 0
        self.rect.left = 80 + x * 16
        self.rect.top = 78 + y * 16
        self.movable = 0

    def movevalable(self):
        res = []
        if PLATE.plate[self.y][self.x+2] == 1 and PLATE.plate[self.y+1][self.x+2] == 1:
            res.append(0)
        if PLATE.plate[self.y-1][self.x] == 1 and PLATE.plate[self.y-1][self.x+1] == 1:
            res.append(1)
        if PLATE.plate[self.y][self.x-1] == 1 and PLATE.plate[self.y+1][self.x-1] == 1:
            res.append(2)
        if PLATE.plate[self.y+2][self.x] == 1 and PLATE.plate[self.y+2][self.x+1] == 1:
            res.append(3)
        return res

    def move(self):
        self.movable = (self.movable + 1) % 3
        if self.movable == 0:
            if GARGOU.level == 1: direct = random.randint(0, 3)
            elif GARGOU.level == 2: direct = random.choice([0, 1, 2, 3, self.facing, self.facing])
            elif GARGOU.level == 3: direct = random.choice(self.movevalable())
            elif GARGOU.level == 4:
                m = self.movevalable()
                m.extend([self.facing, self.facing, self.facing])
                direct = random.choice(m)
            elif GARGOU.level == 5:
                m = []
                if self.x == GARGOU.x:
                    if self.y - GARGOU.y < 0: direct = 3
                    elif self.y - GARGOU.y > 0: direct = 1
                elif self.y == GARGOU.y:
```

```

        if self.x - GARGOU.x < 0: direct = 0
        elif self.x - GARGOU.x > 0: direct = 2
    else:
        m = self.movevalable()
        m.extend([self.facing, self.facing, self.facing])
        direct = random.choice(m)
    elif GARGOU.level >= 6:
        m = []
        if self.x - GARGOU.x < 0: m.append(0)
        elif self.x - GARGOU.x > 0: m.append(2)
        if self.y - GARGOU.y < 0: m.append(3)
        elif self.y - GARGOU.y > 0: m.append(1)
        direct = random.choice(m)

    self.facing = direct

    x = self.x; y = self.y
    if direct == 0: x = self.x + 1
    elif direct == 1: y = self.y - 1
    elif direct == 2: x = self.x - 1
    elif direct == 3: y = self.y + 1

    if PLATE.plate[y][x] == PLATE.plate[y][x+1] == PLATE.plate[y+1][x] ==
    PLATE.plate[y+1][x+1] == 1:
        for mum in MUMMY:
            if self != mum and abs(x-mum.x) <= 1 and abs(y -mum.y) <= 1:
                return
        else:
            self.x = x
            self.y = y
    else: return
    self.moving = (self.moving + 1) % 2
    self.image = ImageOhMummy['Mummy']['Face'][2 * self.facing + self.moving]
    self.rect.left = 80 + 16 * self.x
    self.rect.top = 78 + 16 * self.y

def Initialize(self, x, y):
    self.x = x
    self.y = y
    self.facing = 0
    self.moving = 0
    self.rect.left = 80 + x * 16
    self.rect.top = 78 + y * 16
    self.movable = 0
    
```

IV-C-6 - Classe GuardianMummy



La classe **GuardianMummy** permet de gérer la naissance de la momie cachée dans une des boîtes. Sa position dans sa boîte va dépendre de la position du héros au moment où celui-ci a dégagé entièrement la boîte de telle sorte que cette position soit la plus proche de la position du dégagement. Là encore, il n'y a qu'une fonction utile, la fonction **Update**. Celle-ci va gérer l'apparition régulière de la momie.

Il y a 16 vues avant que la momie soit entièrement apparue. Une fois ce nombre atteint, on rajoute donc une momie dans la liste globale **MUMMY**. la case de naissance de la momie devenant une case sur laquelle le héros et les autres momies peuvent se déplacer, on met à jour le tableau **PLATE.plate**. Enfin, on vide la liste globale **GUARDIANMUMMY** (qui ne peut contenir au maximum qu'un élément) et on tue l'instance de la classe **GuardianMummy**.

```
class GuardianMummy(pygame.sprite.Sprite):
    def __init__(self, x, y):
        ## x et y corresponde au numéro de la boîte sachant que 0<=x<=4 et 0<=y<=3
        pygame.sprite.Sprite.__init__(self, self.containers)
        self.view = 0
        self.image = pygame.Surface((0,0)) ## On part d'une image vide
        self.rect = self.image.get_rect()
        ## Définition de la position de naissance de la Momie
        if GARGOU.x < 3 + 7 * x: self.x = 3 + 7 * x
        elif GARGOU.x > 6 + 7 * x: self.x = 6 + 7 * x
        else: self.x = GARGOU.x
        if GARGOU.y < 5 + 5 * y: self.y = 5 + 5 * y
        elif GARGOU.y > 6 + 5 * y: self.y = 6 + 5 * y
        else: self.y = GARGOU.y
        self.movable = 0

    def Update(self):
        self.movable = (self.movable + 1 ) % 6
        if self.movable == 0:
            self.view += 1
            self.image = ImageOhMummy['GuardianMummy'][self.view]
            self.rect = self.image.get_rect()
            self.rect.top = 110 + 16 * self.y - 2 * self.view
            self.rect.left = 80 + 16 * self.x

            if self.view == 16:
                MUMMY.append(Mummy(self.x, self.y))
                PLATE.plate[self.y][self.x] = 1
                PLATE.plate[self.y+1][self.x] = 1
                PLATE.plate[self.y][self.x+1] = 1
                PLATE.plate[self.y+1][self.x+1] = 1
                PLATE.image.blit(ImageOhMummy['Trace']['None'], (16 * self.x - 16, 16 * self.y -
16))
                PLATE.image.blit(ImageOhMummy['Trace']['None'], (16 * (self.x+1) - 16, 16 * self.y
- 16))
                PLATE.image.blit(ImageOhMummy['Trace']['None'], (16 * self.x - 16, 16 * (self.y+1)
- 16))
                PLATE.image.blit(ImageOhMummy['Trace']['None'], (16 * (self.x+1) - 16, 16 *
(self.y+1) - 16))
                GUARDIANMUMMY.pop()
                self.kill()
```

IV-C-7 - Classe Plate

0,7	0,8	0,9	0,10	0,11	0,12	0,13	0,14	0,15	0,16	0,17	0,18	0,19	0,20	0,21	0,22	0,23	0,24	0,25	0,26	0,27	0,28	0,29	0,30	0,31	0,32
1,7	1,8	1,9	1,10	1,11	1,12	1,13	1,14	1,15	1,16	1,17	1,18	1,19	1,20	1,21	1,22	1,23	1,24	1,25							
2,7	2,8	2,9	2,10	2,11	2,12	2,13	2,14	2,15	2,16	2,17	2,18	2,19													
5,7																									
6,7																									
7,7																									
10,7																									
11,7																									
12,7																									
20,7			20,10																						
21,7			21,10																						
22,7			22,10																						

La classe **Plate** gère le plateau de jeu.

L'attribut **plate** est un tableau à 2 dimensions représentant les cases possibles de déplacement, 1 étant une case accessible et 0 une non accessible (bord ou une partie d'une boîte). Comme on le voit sur la figure ci-dessus, les cases jaunes représentent une partie d'un mur, les cases blanches une partie d'une boîte et les cases noires représentent le chemin sur lequel peut mouvoir le héros.

L'attribut **trace** est un tableau à 2 dimensions donnant les cases sur lesquelles le héros s'est déjà déplacé et permettant de savoir si une boîte a été entièrement dégagée. Il a donc les mêmes dimensions que l'attribut **plate**.
L'attribut **box** est un tableau à 2 dimensions donnant l'état des boîtes à découvrir (au départ toutes sont à **NonTested**).

L'attribut **boxchoice** est un tableau à 1 dimension de même taille que **box** qui contient toutes les découvertes à faire, chaque découverte étant aléatoirement reliée à une case de **box**.

L'attribut **Key** détermine si la clé a été découverte.

L'attribut **Scroll** détermine si le héros possède le parchemin.

L'attribut **RoyalMummy** détermine si les restes de la momie ont été découvertes.

L'attribut **Level** détermine le niveau dans la pyramide dans lequel évolue le héros (1 <= Level <= 5)

L'attribut **ChangeLevel** indique si le héros doit changer de niveau dans la pyramide.

Une case du plateau de jeu représente une surface de 16x16 à l'écran. Ainsi les personnages prennent 2x2 cases sur le jeu. Une boîte contient 5x3 cases. On se retrouve ainsi avec un total de 39x26 cases sachant que la bordure du tableau **plate** représentent le mur extérieur (d'épaisseur une case). Il y a ainsi tout une gymnastique pour se réperer à l'intérieur de ce tableau.

Chaque fois qu'on initialise le plateau (c'est à dire qu'on change de niveau), une momie supplémentaire apparaît (**MUMMY.append(Mummy(1, 23))**) que l'on place en bas à gauche de l'écran, les momies survivantes sont repositionnées à côté.

La fonction **Update** (appelée lors du mouvement du héros) permet de mettre à jour et d'afficher le tableau des traces de pas puis appelle la fonction **UpdateBox** pour vérifier si une boîte a été ou non entièrement dégagée. Celle-ci appelle elle-même la fonction **UpdateClose** en spécifiant la boîte test. Si une boîte est dégagée, on effectue le résultat de la boîte (augmentation du score, apparition d'une momie, découverte d'un objet, ...).

De même que la plupart des autres classes, une seule instance globale est créée et nommée **PLATE**, ce qui permet d'y avoir accès facilement dans les autres classes.

```
class Plate(pygame.sprite.Sprite):
    def __init__(self, level):
        pygame.sprite.Sprite.__init__(self, self.containers)
        self.Initialize(level)

    def Initialize(self, level):
        self.plate = [39 * [0] for i in range(26)]
        self.trace = [39 * [0] for i in range(26)]
        self.box = [5 * ['NonTested'] for i in range(4)]
        l = str(level)
        self.boxchoice = ['Treasure', 'Treasure', 'Treasure', 'Treasure', 'Treasure', 'Treasure',
'Treasure', 'Treasure', 'Treasure', 'Treasure',
                        'GuardianMummy'+l, 'Key', 'RoyalMummy', 'Scroll', 'Empty'+l, 'Empty'+l,
'Empty'+l, 'Empty'+l, 'Empty'+l, 'Empty'+l]
        random.shuffle(self.boxchoice)

        self.Key = False
        self.Scroll = False
        self.RoyalMummy = False
        self.Level = level
        self.ChangeLevel = False

        for i in range(3, 24, 5):
            for j in range(1, 38):
                self.plate[i][j] = 1
                self.plate[i+1][j] = 1
        for j in range(1, 37, 7):
            for i in range(3, 25):
                self.plate[i][j] = 1
                self.plate[i][j+1] = 1
        self.plate[1][15] = 1
        self.plate[1][16] = 1
        self.plate[2][15] = 1
        self.plate[2][16] = 1

        self.image = pygame.Surface((592, 384))
        self.image.fill((255, 255, 0))
        self.rect = self.image.get_rect()
        self.rect.left = 96
        self.rect.top = 94
```

```

    ## Affichage du plateau
    for i in range(1, 25):
        for j in range(1, 38):
            if self.plate[i][j] == 1:
                self.image.blit(ImageOhMummy['Trace']['None'], (16 * j - 16, 16 * i - 16))

    for i in range(4):
        for j in range(5):
            self.image.blit(ImageOhMummy['Box']['NonTested'+1], (32 + 16 * 7 * j, 64 + 16 * 5 *
i))
    LIFE.Update()
    MUMMY.append(Mummy(1, 23))
    i = 1
    for mum in MUMMY:
        mum.Initialize(1 + 2 * i, 23)
        i += 1

    def Update(self, x, y, facing, moving):
        ## Mise à jour des traces
        if facing == 0:
            self.trace[y][x-1] = 1
            self.trace[y+1][x-1] = 1
            if moving == 0: self.image.blit(ImageOhMummy['Trace']['DownRight'], (16 * x - 32, 16 *
y - 16))
        else: self.image.blit(ImageOhMummy['Trace']['UpRight'], (16 * x - 32, 16 * y - 16))
        elif facing == 1:
            self.trace[y+2][x] = 1
            self.trace[y+2][x+1] = 1
            if moving == 0: self.image.blit(ImageOhMummy['Trace']['LeftUp'], (16 * x - 16, 16 * y +
16))
        else: self.image.blit(ImageOhMummy['Trace']['RightUp'], (16 * x - 16, 16 * y + 16))
        elif facing == 2:
            self.trace[y][x+2] = 1
            self.trace[y+1][x+2] = 1
            if moving == 0: self.image.blit(ImageOhMummy['Trace']['DownLeft'], (16 * x + 16, 16 *
y - 16))
        else: self.image.blit(ImageOhMummy['Trace']['UpLeft'], (16 * x + 16, 16 * y - 16))
        elif facing == 3:
            self.trace[y-1][x] = 1
            self.trace[y-1][x+1] = 1
            if moving == 0: self.image.blit(ImageOhMummy['Trace']['LeftDown'], (16 * x - 16, 16 * y
- 32))
        else: self.image.blit(ImageOhMummy['Trace']['RightDown'], (16 * x - 16, 16 * y - 32))

        ## Mise à jour des cases
        if y>2: self.UpdateBox(x, y)

    def UpdateBox(self, x, y):
        xx1 = min(4, max(0, (x - 3) // 7))
        yy1 = min(3, max(0, (y - 5) // 5))
        xx2 = min(4, (x - 1) // 7)
        yy2 = min(3, (y - 3) // 5)
        if self.box[yy1][xx1] == 'NonTested': self.UpdateClose(xx1, yy1)
        if self.box[yy1][xx2] == 'NonTested': self.UpdateClose(xx2, yy1)
        if self.box[yy2][xx1] == 'NonTested': self.UpdateClose(xx1, yy2)
        if self.box[yy2][xx2] == 'NonTested': self.UpdateClose(xx2, yy2)

    def UpdateClose(self, x, y):
        for i in range(3 + 5 * y, 10 + 5 * y):
            for j in range(1 + 7 * x, 10 + 7 * x):
                if self.plate[i][j] == 1 and self.trace[i][j] == 0: return False
        self.box[y][x] = self.boxchoice[5 * y + x]
        self.image.blit(ImageOhMummy['Box'][self.box[y][x]], (32 + 112 * x, 64 + 80 * y))
        if self.box[y][x] == 'Treasure': SCORE.score += 5
        elif self.box[y][x] == "RoyalMummy":
            SCORE.score +=50
            self.RoyalMummy = True

```

```
elif self.box[y][x] == "Scroll":
    self.Scroll = True
    SCORE.Update()
elif self.box[y][x] == "Key":
    self.Key = True
elif self.box[y][x] == "GuardianMummy"+str(self.Level):
    GUARDIANMUMMY.append(GuardianMummy(x, y))
SCORE.Update()
```

IV-D - Fonction main()

La fonction **main** va nous permettre d'initialiser toutes nos variables et de gérer les évènements clavier et le rafraîchissement de l'écran.

clock = pygame.time.Clock() permet de créer une horloge qui permettra ensuite de fixer le nombre d'images à la seconde que générera le jeu. Ce nombre est alors défini par la fonction **clock.tick(fps)** où **fps** est le nombre d'images par seconde.

On définit ensuite les différents *groupes de sprite* (**all**, **plate**, **endgame**) de type ici **pygame.sprite.RenderUpdates** qui géreront l'affichage à l'écran, lesquels sont attribués aux différentes classes (**Plate.containers = plate**, **Score.containers = all**, ...). Enfin nos différentes variables sont créées et définies de manière globale. Puis on affiche un arrière plan sur l'écran de jeu (**screen.blit(background, (0, 0))**) et on force le rafraîchissement entier de la fenêtre de jeu (**pygame.display.flip()**)

```
def main():
    clock = pygame.time.Clock()

    background = pygame.Surface(SCREENRECT.size)
    background = background.convert()
    background.fill((255,255,0))

    all = pygame.sprite.RenderUpdates()
    plate = pygame.sprite.RenderUpdates()
    endgame = pygame.sprite.RenderUpdates()
    Plate.containers = plate
    Score.containers = all
    Gargou.containers = all
    Life.containers = all
    Mummy.containers = all
    GuardianMummy.containers = all
    global PLATE
    global SCORE
    global GARGOU
    global LIFE
    global MUMMY
    global GUARDIANMUMMY

    SCORE = Score()
    LIFE = Life()
    MUMMY = []
    GUARDIANMUMMY = []
    GARGOU = Gargou()
    GARGOU.level = 1
    PLATE = Plate(1)

    screen.blit(background, (0, 0))
    pygame.display.flip()
```

On définit enfin une boucle infinie dans laquelle sera contrôlé le jeu.

Cette boucle commence par une condition qui teste les événements reçus par **Pygame** (`pygame.event.get()`). Si le joueur a appuyé sur la touche *Escape* (événement de type **KEYDOWN**) ou bien qu'un événement de type **QUIT** a été généré, on quitte le jeu.

On effectue alors les actions des différents personnages:

- On récupère les touches sur lesquelles le joueur a appuyé en ne considérant que les flèches et on effectue les déplacements du héros (`GARGOU.move(direction)`).
- Si une momie est en train de naître, `guardian.Update()` met à jour son évolution.
- Pour chaque momie du jeu, on effectue un déplacement (`mum.move()`). Puis on vérifie qu'une fois déplacée, elle n'entre pas en collision avec le héros. Si le héros perd sa dernière vie, on affiche l'image GameOver (`screen.blit(ImageOhMummy['GameOver'], (208, 238))`) et on attend que le joueur quitte le jeu ou recommence une partie en appuyant sur la touche 'c' ou 'C'.

Si le joueur a fini un tableau (condition `PLATE.ChangeLevel` à vérifier), on réinitialise le tout.

Enfin, à la fin de la boucle, on réaffiche en dessinant nos groupes de sprite sur la scène (`pl = plate.draw(screen)` et `dirty = all.draw(screen)`) et en effectuant le rafraîchissement à l'écran (`pygame.display.update(pl)` et `pygame.display.update(dirty)`), le tout en forçant le nombre de frames par seconde (`clock.tick(12)`)

```
goon = True
while goon:
    for event in pygame.event.get():
        if event.type == QUIT or (event.type == KEYDOWN and event.key == K_ESCAPE): return

    keystate = pygame.key.get_pressed()
    direction = [keystate[K_RIGHT], keystate[K_UP], keystate[K_LEFT], keystate[K_DOWN]]

    GARGOU.move(direction)
    for guardian in GUARDIANMUMMY: guardian.Update()
    for mum in MUMMY:
        mum.move()
        if abs(mum.x - GARGOU.x) <= 1 and abs(mum.y - GARGOU.y) <= 1:
            if not PLATE.Scroll:
                LIFE.life -= 1
                LIFE.Update()
                if LIFE.life == 0:
                    goon = False
                    screen.blit(ImageOhMummy['GameOver'], (208, 238))
                    while True:
                        for event in pygame.event.get():
                            if event.type == QUIT or (event.type == KEYDOWN and event.key ==
K_ESCAPE): return
                        keystate = pygame.key.get_pressed()
                        if keystate[ord('c')] or keystate[ord('C')]:
                            return main()
                        pygame.display.flip()
                        clock.tick(10)
            else:
                PLATE.Scroll = False
```

```
        SCORE.Update()
        mum.kill()
        MUMMY.remove(mum)

    if PLATE.ChangeLevel:
        if PLATE.Level == 1:
            if random.randint(0, 1) == 0 and LIFE.life < 9:
                background.blit(ImageOhMummy['EndLife'], (0, 0))
                LIFE.life += 1
            else:
                background.blit(ImageOhMummy['End200'], (0, 0))
                SCORE.score += 200

        screen.blit(background, (0, 0))
        for mum in MUMMY: mum.kill()
        MUMMY = []
        GARGOU.level += 1

    while True:
        for event in pygame.event.get():
            if event.type == QUIT or (event.type == KEYDOWN and event.key == K_ESCAPE):

return

        keystate = pygame.key.get_pressed()
        if keystate[ord('c')] or keystate[ord('C')]: break
        pygame.display.flip()
        clock.tick(10)

    for mum in GUARDIANMUMMY: mum.kill()
    GUARDIANMUMMY = []
    background.fill((255,255,0))
    screen.blit(background, (0, 0))
    pygame.display.flip()

    PLATE.Initialize(PLATE.Level)
    SCORE.Update()

#draw the scene
pl = plate.draw(screen)
dirty = all.draw(screen)

pygame.display.update(pl)
pygame.display.update(dirty)

clock.tick(12)
pygame.time.wait(1000)
```

V - Conclusion

Je pense que vous n'aurez aucun mal à comprendre ce source d'un peu plus de 500 lignes. D'autres développements pourraient s'ajouter pour rester encore plus fidèle à la version CPC (par exemple, insertion des pages de présentation du jeu, ajout des sons et musiques, ...).

Si vous voulez aller plus loin avec **Pygame**, je vous conseille alors de parcourir directement la documentation disponible sur le site officiel.

VI - Téléchargement

Version	Date	Taille	Mode FTP	Mode HTTP de secours
1.0.0.0.1	2007.04.05	33.0 ko	OhMummy.zip	OhMummy.zip

