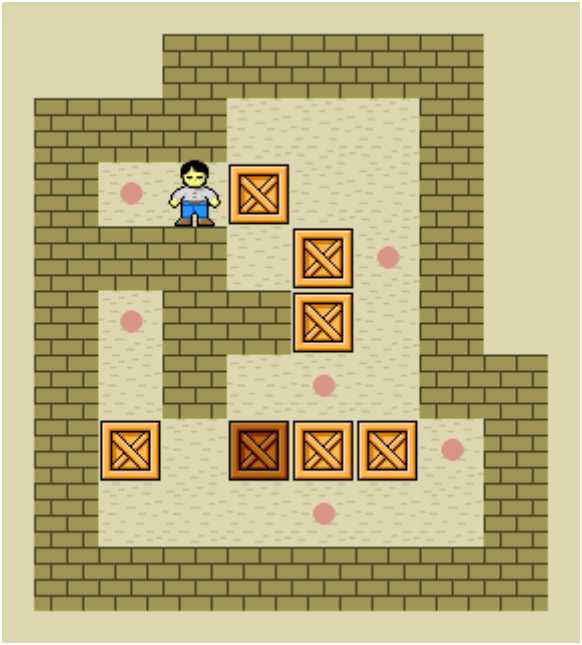
Sokoban 1

Le jeu



Gardien d'un entrepôt divisé en cases carrées, le joueur doit ranger des caisses sur des cases cibles. Il peut se déplacer dans les quatre directions, et pousser (mais pas tirer) une seule caisse à la fois. Une fois toutes les caisses rangées (c'est parfois un vrai casse-tête), le niveau est réussi et le joueur passe au niveau suivant, plus difficile en général. L'idéal est de réussir avec le moins de coups possibles (déplacements et poussées).

Le jeu que nous allons réaliser, est une traduction-adaptation du jeu Star Pusher de Al Sweigart. Ce jeu est très intéressant à plusieurs titres, car nous allons voir comment créer des niveaux de jeu, comment importer tout un groupe d'image et les gérer etc.

Les niveaux de jeu

Ils sont contenus dans des fichiers textes de ce type :

```
    #####
    #   #
    #$  #
  ###  $##
  #  $ $ #
### # ## #   ######
#   # ## #####  ..#
# $  $         ..#
##### ### #@##  ..#
    #     #########
    #######
```

: Mur.

$: Dans notre jeu ce sera la place d'une étoile au démarrage du jeu.

. : Une zone de rangement ou objectif. C'est à cette place que le personnage doit pousser une étoile.

* : Une zone de rangement (objectif) avec une étoile posée dessus

@ : La position de départ du personnage

+ : Une zone de rangement (objectif) avec le personnage posé dessus

Espace : Une zone à l'extérieur

Dans le fichier texte contenant la description des niveaux de jeu, les commentaires commencent par ;

La description d'un niveau se termine par une ligne vide.

Il nous faudra donc lire le fichier contenant les niveaux de jeu et traduire les symboles qu'il contient en type de tuile et position sur la scène.

Les symboles supplémentaires utilisés dans ce jeu, mais n'apparaissant pas dans un fichier de niveaux :

x : Un coin de mur

o : Un espace (planché) à l'intérieur

1 : Un rocher

2 : Un arbrisseau

3 : Un arbre moyen

4 : Un arbre ébourifé

Ces symboles seront rajoutés à la carte d'un niveau pour la compléter.

Les tuiles (tile en anglais)

Dans un jeu vidéo, les tuiles sont de petites images carrées, rectangulaires, voir hexagonales, disposées sur une grille. L'ensemble complet des tuiles disponibles pour une utilisation dans une zone de jeu est appelé un « jeu de tuiles ».

L'écran est représenté par une grille composée de nombreuses cases, sur lesquelles sont appliquées une image par case. C'est le « tile mapping »

Ici dans ce jeu de Mario, en dehors des personnages : Mario, ennemies, nous avons des tuiles uniques (points d'interrogation) et des tuiles composées comme le pot de fleurs, qui est plus gros qu'une case. Pour la machine, le pot de fleur ne sera pas une tuile unique, mais 8 tuiles infranchissables.

Les structures de données du jeu

L'état du jeu

Cet état, **jeuEtat**, est conservé dans un dictionnaire comportant 3 clés :

'**joueur**' : la valeur associée est un tuple de 2 entiers x et y, position actuelle du joueur.

'**compteur**' : nombre de pas déjà effectués par le joueur.

'**etoiles**' : liste contenant les coordonnées x et y de toutes les étoiles du niveau actuel.

Les niveaux

L'objet **niveau** est lui aussi un dictionnaire comportant 5 clés :

'**largeur**' : Entier donnant le nombre de tuiles en largeur sur la grille

'**hauteur**' : Entier donnant le nombre de tuiles en hauteur sur la grille

'**carte**' : la carte pour ce niveau de jeu, construite après lecture du fichier texte.

'**objectifs**' : liste des coordonnées x et y des espaces où il faut pousser les étoiles.

'**etatInitial**' : état du jeu au départ du niveau = **jeuEtat** après la lecture du fichier texte

Les cartes (map)

A la lecture du fichier contenant le descriptif des niveaux, deux listes sont créées.

Une première liste contient les lignes de texte d'un niveau : **lignesTexteCarte**. Chaque ligne représente une ligne sur la carte. La position de chaque caractère dans la ligne représente

une colonne sur la carte et le caratère lui-même, le type de tuile à mettre dans la case correspondante.

Une deuxième liste est créée en parcourant chaque ligne de la liste précédente afin de créer une **carte** en deux dimensions ([x][y]) avec comme valeur le caractère représentant le type de tuile.

Exemple : carte[1][3] = '@' indique que le personnage est au départ sur la ligne 3 colonne 1 de la carte.

Les tuiles, les personnages et leurs images

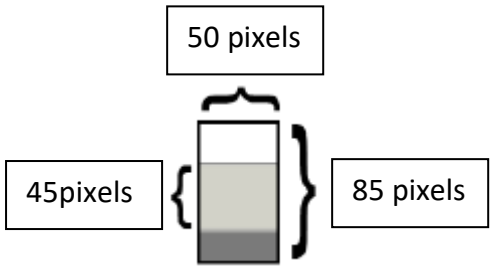
Toutes les images utilisables dans le jeu sont chargées en mémoire dans un dictionnaire, dont les clés seront utilisées pour associer un symbole du fichier texte, avec l'image correspondante.

Exemple :

```
DicoImage={'coin' : pygame.image.load('Wall Block Tall.png'),
           'mur' : pygame.image.load('Wood Block Tall.png'),
           'etoile' : pygame.image.load('Star.png'),
           Etc
          }
```

 Cette image est maintenant associée à la clé : mur

Les tuiles ont une largeur (**LARGEUR_TUILLE**) de 50 pixels, une hauteur (**HAUTEUR_TUILLE**) de 85 pixels. Dans ce jeu, certaines tuiles peuvent se chevaucher. Une troisième donnée est donc nécessaire pour indiquer la hauteur de la partie représentant le sol (**SOL_TUILE**): 45 pixels.



La partie sol (45 pixels) de la tuile sera toujours apparente, le reste peut être recouvert par une autre tuile. Certaines tuiles ont des zones transparentes.

Les tuiles extérieures peuvent avoir des décorations supplémentaires telles que des arbres, des rochers. Une variable **PCT_DECO** indique le pourcentage de tuiles extérieures qui de façon aléatoire auront ces décorations.

Liaison entre symbole du fichier texte et le dictionnaire des images.

On utilise la aussi un dictionnaire, dans lequel chaque clé est un symbole du fichier texte.

La valeur associée à une clé de ce dictionaire, est la valeur associée à une clé du dictionnaire des images, donc la référence à l'image elle-même.

```
LiaisonTuile = {'x' : DicoImage['coin'],
                '#': DicoImage ['mur'],
                Etc.
                }
```

Liaison entre personnages et dictionnaire des images

Le jeu permet de changer le personnage affiché. Une liste de personnages fait le lien entre un personnage et son image.

ImagesJoueur =[DicoImage['princesse'], DictImage['garcon'], etc.]

Un index joueurActuel indique le numéro du joueur choisi par le joueur.

Liaison entre symbole du fichier texte et les éléments décoratifs extérieur (arbre, rocher…)

```
LiaisonExterieur = {'1' : DicoImage['rocher'],
                    '2': DicoImage ['petit arbre'],
                    Etc.
                   }
```

Lecture de fichier texte

Pour lire un fichier texte, il faut commencer par l'ouvrir en mode lecture.

fichier = open('monFichier.txt', 'r')

La fonction readlines() lit toutes les lignes de texte du fichier et les renvoie sous forme de liste.

contenu = fichier.readlines()

contenu est une liste de la forme :

[‘ma première ligne.\n’, ‘ma deuxième ligne.\n, ‘…\n’]

Une fois la lecture terminée, on ferme le fichier.

fichier.close()

```
def lectureFichierNiveaux(nomFichier):
    assert os.path.exists(nomFichier), 'Je ne trouve pas le fichier de niveaux: %s' % (nomFichier)
    fichierCarte = open(nomFichier, 'r')
    # Chaque niveau se termine par une ligne vide
    contenu = fichierCarte.readlignes() + ['\r\n']
    fichierCarte.close()
    #liste de tous les niveaux
    niveaux = []
    numNiveau = 0
    #liste contenant les lignes de texte d'un niveau
    lignesTexteCarte = []
    #la carte créée à partir des données de lignesTexteCarte
    carte = []
    for ligneNum in range(len(contenu)):
        #on traite chaque ligne du niveau
        ligne = contenu[ligneNum].rstrip('\r\n')
        if ';' in ligne:
            #une ligne démarrant avec ; est une ligne de commentaire
            #que l'on doit ignorer
            ligne = ligne[:ligne.find(';')]
        if ligne != '':
            # une ligne qui n'est pas vide, appartient à la carte de ce niveau.
            lignesTexteCarte.append(ligne)
```

```
elif ligne == '' and len(lignesTexteCarte) > 0:
    # Une ligne vide, indique la fin de la carte d'un niveau.
    #conversion du texte contenu dans lignesTexteCarte
    #en une carte de niveau
    # Trouver la ligne la plus longue de la carte.
    largeurMax = -1
    for i in range(len(lignesTexteCarte)):
        if len(lignesTexteCarte[i]) > largeurMax:
            largeurMax = len(lignesTexteCarte[i])
    # Ajouter des espaces aux lignes les plus courtes de façon à
    #ce que la carte soit rectangulaire.
    for i in range(len(lignesTexteCarte)):
        lignesTexteCarte[i] += ' ' * (largeurMax - len(lignesTexteCarte[i]))
    # Convertir lignesTexteCarte en une carte.
    for x in range(len(lignesTexteCarte[0])):
        carte.append([])
    for y in range(len(lignesTexteCarte)):
        for x in range(largeurMax):
            carte[x].append(lignesTexteCarte[y][x])
    #boucle dans la carte pour trouver les caractères @, . et $
    #indiquant l'état de départ du jeu
    #position de départ du joueur
    startx = None
    starty = None
    #liste de chaque coordonnées x, y des objectifs
    objectifs = []
    #liste de chaque coordonnées x, y des étoiles
    etoiles = []
```

```
        for x in range(largeurMax):
            for y in range(len(carte[x])):
                if carte[x][y] in ('@', '+'):
                    #'@' position départ joueur
                    #'+' un objectif avec le joueur posé dessus au départ
                    startx = x
                    starty = y
                if carte[x][y] in ('.', '+', '*'):
                    # '.' objectif
                    # '*' un objectif avec une étoile posée
                    # '+' un objectif avec le joueur posé dessus au départ
                    objectifs.append((x, y))
                if carte[x][y] in ('$', '*'):
                    # '$' étoile
                    # '*' un objectif avec une étoile posée
                    etoiles.append((x, y))

        # Tests élémentaires :
        assert startx != None and starty != None, 'Dans le niveau %s (autour de la ligne %s) dans %s il manque un "@" ou un "+" pour marquer le point de départ du joueur.' % (numNiveau+1, ligneNum, nomFichier)
        assert len(objectifs) > 0, 'Dans le niveau %s (autour de la ligne %s) dans %s il doit y avoir au moins un objectif.' % (numNiveau+1, ligneNum, nomFichier)
        assert len(etoiles) >= len(objectifs), 'Dans le niveau %s (autour de la ligne %s) dans %s il y a %s objectifs et seulement %s étoiles. Ce niveau ne peut pas être résolu' % (numNiveau+1, ligneNum, nomFichier, len(objectifs), len(etoiles))

        # Etat du jeu au départ du niveau
        jeuEtat = {'joueur': (startx, starty),
                   'compteur': 0,
                   'etoiles': etoiles}
```

```
            niveau = {'largeur': largeurMax,
                      'hauteur': len(carte),
                      'carte': carte,
                      'objectifs': objectifs,
                      'etatInitial': jeuEtat}
            niveaux.append(niveau)
            # Remise à 0 des variables pour lire la carte suivante
            lignesTexteCarte = []
            carte = []
            jeuEtat = {}
            numNiveau += 1
    return niveaux
```

Modifier la carte pour y ajouter de nouveau éléments

Actuellement, la carte obtenue par lecture du fichier texte, ne fait pas la distinction entre tuiles au sol à l'extérieur des murs et tuiles au sol à l'intérieur des murs. Toutes ces tuiles sont pour l'instant symbolisées par un espace sur la carte. La fonction modifierCarte, est chargée de symboliser les tuiles à l'intérieur des murs par 'o'

 Une tuile extérieure symbolisée par un espace

 Une tuile d'intérieur symbolisée par 'o'

De la même façon, les murs sont tous symbolisés par le caratère '#'. Rien ne distingue l'emplacement des coins de mur. La fonction modifierCarte, est chargée de symboliser les coins de mur par le symbole 'x'.

 Une tuile mur symbolisée par '#'

 Une tuile coin symbolisée par ‘x’

Enfin, le jeu prévoit d’afficher à l’extérieur des murs des décorations supplémentaires, choisies de façon aléatoire. Les tuiles extérieures choisies pour être décorées voient le caractère espace remplacé par un code de décoration : chiffre de 1 à 4

code 1 code 2 code 3 code 4

```
def modifierCarte(carte, startxy):
    startx, starty = startxy
    # Copie la carte de départ, pour ne pas la modifier
    carteCopy = copy.deepcopy(carte)
    # Supprime tous les symboles qui ne correspondent pas à des murs
    for x in range(len(carteCopy)):
        for y in range(len(carteCopy[0])):
            if carteCopy[x][y] in ('$', '.', '@', '+', '*'):
                carteCopy[x][y] = ' '
    # remplissage par diffusion : remplace les caractères représentant
    # des tuiles externes, par des caratères de tuiles internes pour toutes
    #les tuiles situées entre les murs.
    remplissageDiffusion(carteCopy, startx, starty, ' ', 'o')
    # Conversion des murs adjacents en coins.
    for x in range(len(carteCopy)):
        for y in range(len(carteCopy[0])):
            if carteCopy[x][y] == '#':
                if (estMur(carteCopy, x, y-1) and estMur(carteCopy, x+1, y)) or \
                   (estMur(carteCopy, x+1, y) and estMur(carteCopy, x, y+1)) or \
                   (estMur(carteCopy, x, y+1) and estMur(carteCopy, x-1, y)) or \
                   (estMur(carteCopy, x-1, y) and estMur(carteCopy, x, y-1)):
```

```
                carteCopy[x][y] = 'x'
            #décoration des tuiles externes : 20% des tuiles seulement
            elif carteCopy[x][y] == ' ' and random.randint(0, 99) < PCT_DECO:
                carteCopy[x][y] = random.choice(list(LiaisonExterieur .keys()))

    return carteCopy


def estMur(carte, x, y):
    """Renvoie True si à la position (x,y) sur la carte
    on a un mur. Sinon retourne False."""
    if x < 0 or x >= len(carte) or y < 0 or y >= len(carte[x]):
        return False # x et y ne sont pas sur la carte
    elif carte[x][y] in ('#', 'x'):
        return True # en x, y il y a bien un mur
    return False
```

Remplissage par diffusion

Les algorithmes de remplissage par diffusion sont utilisés dans tous les logiciels permettant de colorier une surface à l'aide d'un outil « pot de peinture ». 

L'algorithme de remplissage par diffusion prend trois paramètres pour une image donnée : la position du pixel de départ (appelé aussi germe), la couleur ciblée (colcible) et la couleur de remplacement (colrep). L'algorithme recense tous les pixels de l'image qui sont connectés au germe par un chemin de la couleur ciblée et substitue à cette dernière la couleur de remplacement

remplissage(pixel, colcible, colrep).

début

Soit P une liste vide

si couleur(pixel) ≠ colcible **alors** sortir **finsi**

Mettre le pixel dans la liste P

Tant que P non vide

faire

Sortir un pixel n de P

couleur(n) ← colrep

si couleur(pixel n au nord) = colcible **alors** Mettre le pixel n au nord dans P **finsi**

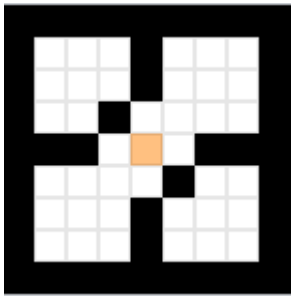
si couleur(pixel n au sud) = colcible **alors** Mettre le pixel n au sud dans P **finsi**

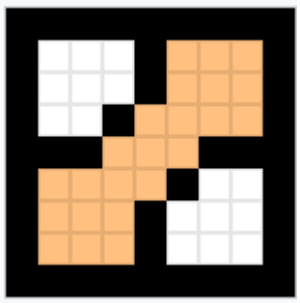
si couleur(pixel n à l'est) = colcible **alors** Mettre le pixel n à l'est dans P **finsi**

si couleur(pixel n à l'ouest)= colcible **alors** Mettre le pixel n à l'ouest dans P **finsi**

fintantque

fin

Au début :  en cours d'algorithme 

A la fin : 

```
def remplissageDiffusion(carte,x,y,ancienCaractere, nouveauCaractere):
    """ Remplace à l'intérieur des murs, les caractères ' ' par un caractère 'o'
    """
    espaceATester=[]
    if carte[x][y]== ancienCaractere:
        espaceATester.append((x,y))
```

```
    while espaceATester !=[]:
        x,y = espaceATester.pop()
        carte[x][y]=nouveauCaractere
        #test case à droite (Est)
        if x < len(carte)-1 and carte[x+1][y]== ancienCaractere:
            espaceATester.append((x+1,y))
        #test case à gauche (Nord)
        if x > 0 and carte[x-1][y]== ancienCaractere:
            espaceATester.append((x-1,y))
        #test case en dessous (Sud)
        if y < len(carte[x])-1 and carte[x][y+1]== ancienCaractere:
            espaceATester.append((x,y+1))
        #test case au dessus (Nord)
        if y > 0 and carte[x][y-1]== ancienCaractere:
            espaceATester.append((x,y-1))
```

Dessiner la scène

Il s'agit ici de créer une surface à afficher avec pygame, sur laquelle tous les caractères de la carte du niveau, sont remplacés par l'image correspondante.

```
def dessinerScene(carte, jeuEtat, objectifs):
    """Dessine la carte sur une surface (Scene) en y incluant le joueur et les étoiles.
    Ici on fait donc le lien entre les caractères enregistrés sur une carte
    et les images qui leur correspondent.
    """
    # surfaceCarte est un objet Surface sur lequel les tuiles sont dessinées.
    surfaceCarteLargeur = len(carte) * LARGEUR_TUILE
    surfaceCarteHauteur = (len(carte[0]) - 1) * SOL_TUILE + HAUTEUR_TUILE
    surfaceCarte = pygame.Surface((surfaceCarteLargeur, surfaceCarteHauteur))
surfaceCarte.fill(BLEUCLAIR)
```

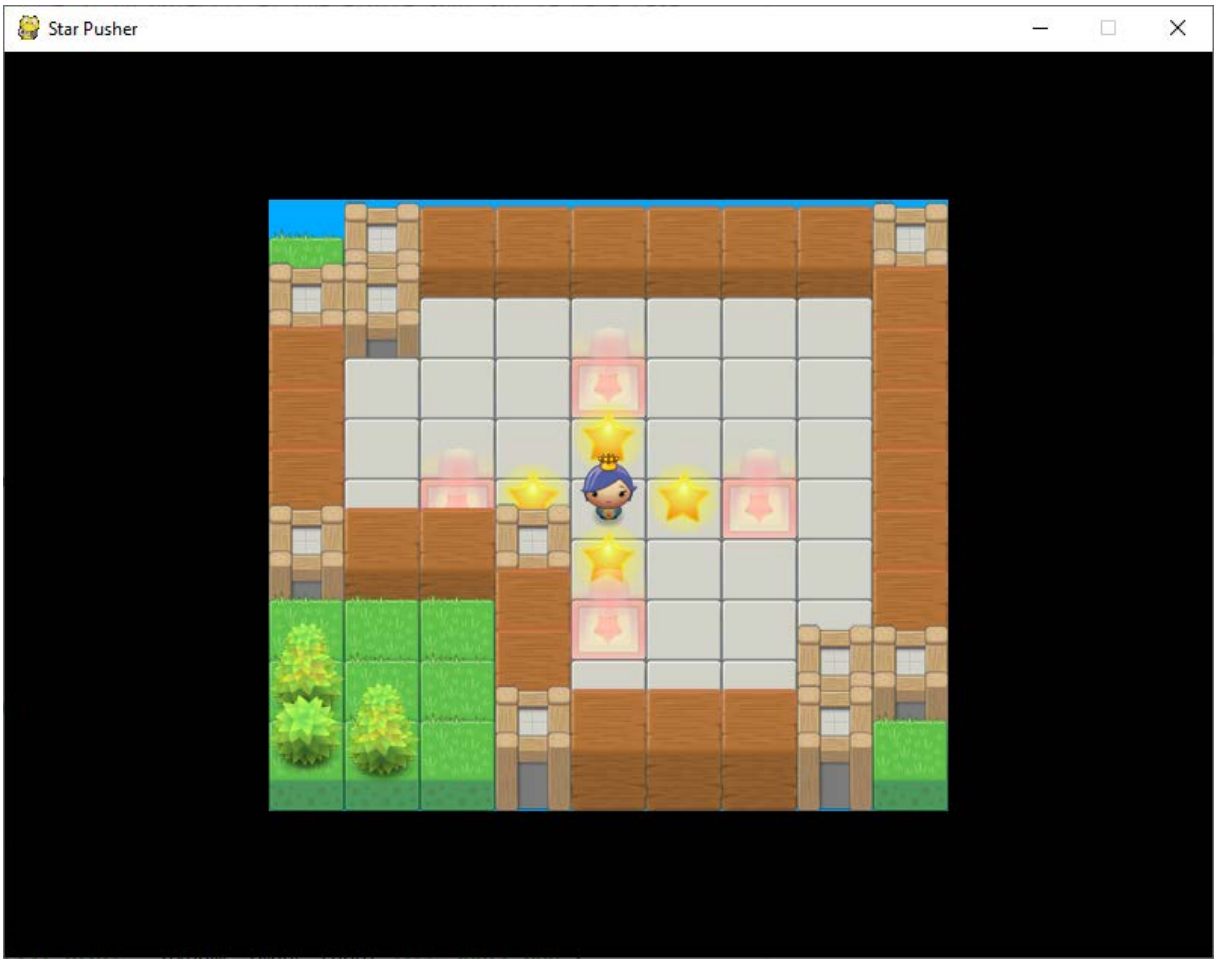
```
    # Dessin des tuiles
    for x in range(len(carte)):
        for y in range(len(carte[x])):
            spaceRect = pygame.Rect((x * LARGEUR_TUILE, y * SOL_TUILE,
LARGEUR_TUILE, HAUTEUR_TUILE))
            # On dessine en premier les tuiles de base : sol/mur
            if carte[x][y] in LiaisonTuile:
                tuileBase = LiaisonTuile[carte[x][y]]
            elif carte[x][y] in LiaisonExterieur:
                tuileBase = LiaisonTuile[' ']
            surfaceCarte.blit(tuileBase, spaceRect)
           #on rajoute les décorations supplémentaires extérieures
            if carte[x][y] in LiaisonExterieur:
                surfaceCarte.blit(LiaisonExterieur[carte[x][y]], spaceRect)
            #dessin des étoiles
            elif (x, y) in jeuEtat['etoiles']:
                if (x, y) in objectifs:
                    # Si un objectif et une étoile sont sur la même case
                    #on dessine l'objectif en premier.
                    surfaceCarte.blit(DicoImage['objectif couvert'], spaceRect)
                # Ensuite on dessine l'étoile.
                surfaceCarte.blit(DicoImage['etoile'], spaceRect)
            #dessine les objectifs, sans étoiles dessus
            elif (x, y) in objectifs:
                surfaceCarte.blit(DicoImage['objectif decouvert'], spaceRect)

            # Dessine le joueur
            if (x, y) == jeuEtat['joueur']:
                #la valeur contenue dans joueurActuel correspond à une clé
                #du dictionnaire ImagesJoueur et indique quelle image représente
```

```
            #actuellement le joueur
            surfaceCarte.blit(ImagesJoueur[joueurActuel], spaceRect)
    return surfaceCarte
```

Le programme de test

Sokoban1.py implémente toutes les fonctions décites ci-dessus, et réalise un premier test, consistant à afficher un niveau de jeu à partir du fichier texte contenant :

```
; Starting demo level:
 ########
##      #
#   .   #
#   $   #
# .$@$. #
####$   #
   #.   #
   #   ##
   #####
```



Deux lancements du jeu, ne donnera pas les mêmes décorations à l'extérieur.

On remarquera le remplacement des tuiles « mur » par des tuiles « coin » à tous les emplacements où deux murs se rejoignent.

 un mur  un coin

L'intérieur est tapissé de tuile d'intérieur et l'extérieur de tuile d'extérieur.

 extérieur  intérieur

Les caractères '.' du fichier texte, sont remplacés par une tuile « objectif decouvert »

Les caractères '$' du fichier texte, sont remplacés par une tuile « etoile »

Le caractère '@' Du fichier texte, est remplacé par une tuile « joueur » : ici 'princesse'