

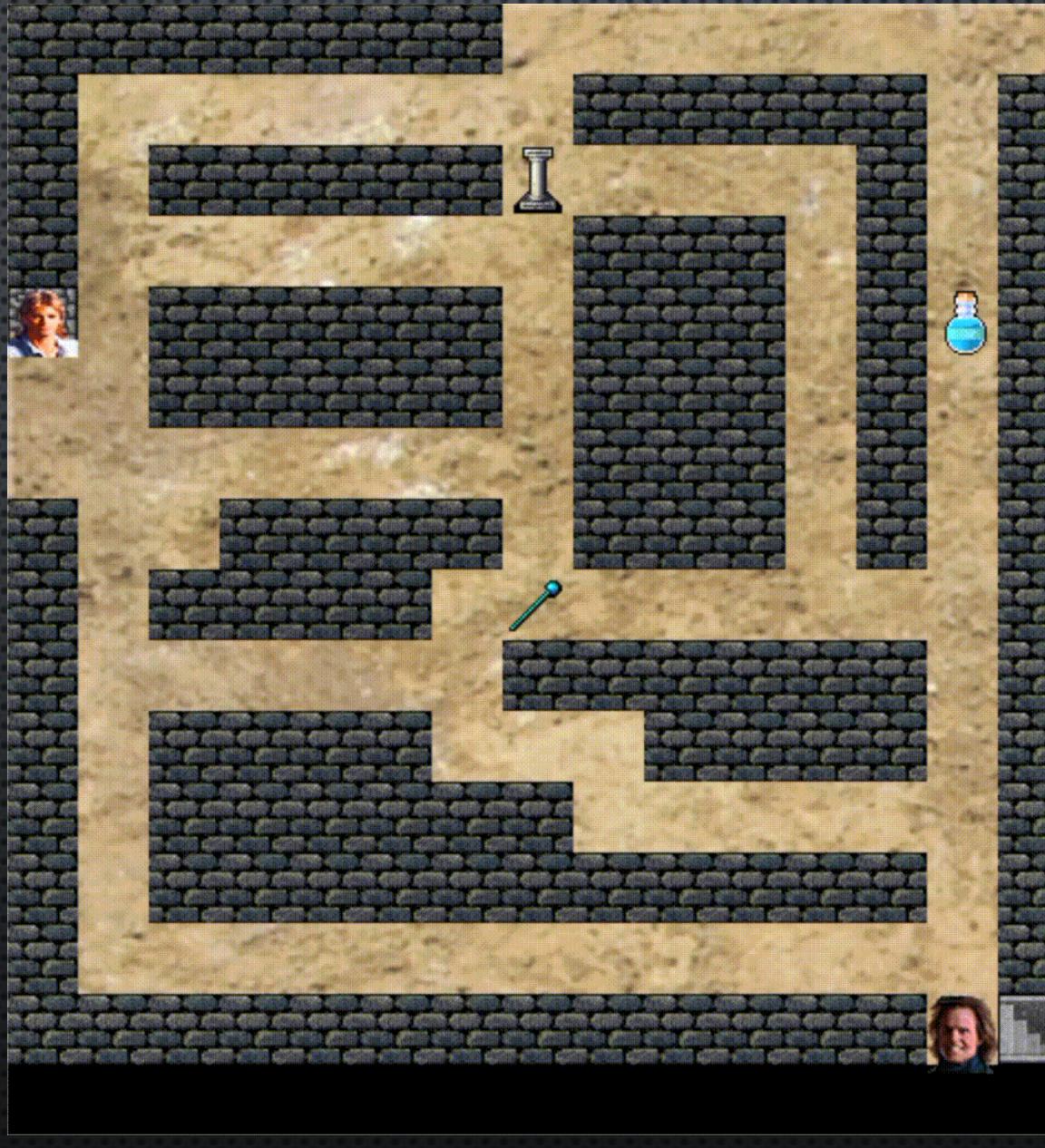
Labyrinthe - MacGyver

Le But

- Le but du jeu est de s'échapper du labyrinthe.
- Pour gagner l'utilisateur doit récupérer 3 objets et neutraliser le garde qui est situé devant la porte.

Les Fonctionnalités

- Il n'y a qu'un seul niveau. La structure (départ, emplacement des murs, arrivée), devra être enregistrée dans un fichier pour la modifier facilement au besoin.
- MacGyver sera contrôlé par les touches directionnelles du clavier.
- Les objets seront répartis aléatoirement et changeront d'emplacement si l'utilisateur ferme le jeu et le relance.
- La fenêtre du jeu sera un carré de 15 sprites de longueur.
- MacGyver devra se déplacer de case en case.
- Il récupérera un objet simplement en se déplaçant dessus.
- Le joueur gagne uniquement si MacGyver a bien récupéré tous les objets et trouvé la sortie du labyrinthe. S'il n'a pas tous les objets et qu'il se présente devant le garde, il meurt.



classes.py (les objets classes)

- **class Maze:**
 - Génère et affiche le labyrinthe
- **class Guard:**
 - Désigne l'emplacement du garde et l'affiche
- **class Player:**
 - Calcule l'emplacement et affiche MacGyver
- **class Item:**
 - Affiche les objets aléatoirement et compte les objets obtenus par l'utilisateur

Class Maze : Générer la structure

Fichier texte ‘n1’ :

1	xxxxxxxx00000000
2	x0000000xxxxx0x
3	x0xxxxx00000x0x
4	x0000000xxx0x0x
5	s0xxxxx0xxx0x0x
6	00xxxxx0xxx0x0x
7	00000000xxx0x0x
8	x00xxxx0xxx0x0x
9	x0xxxx00000000x
10	x00000xxxxxx0x
11	x0xxxx000xxxxx0x
12	x0xxxxxx000000x
13	x0xxxxxxxxxxxx0x
14	x000000000000x
15	xxxxxxxxxxxxxx0e

Variable self.structure :

Class Maze : Générer la structure

```
# Generate maze by reading the text file
def generate(self):
    with open(self.text, "r") as text:
        structure_list = []
        for line in text:
            each_line = []
            for letter in line:
                if letter != '\n':
                    each_line.append(letter)
            structure_list.append(each_line)
    self.structure = structure_list
```

1. Ouvrir le fichier texte.
2. parcourir chaque ligne et caractère
3. Ajouter chaque caractère dans une liste sauf '\n'.
4. Ajouter la liste qui est composée des caractères à 'structure_list'.
5. Répéter.
6. Passer la valeur de 'structure_list' à self.structure.

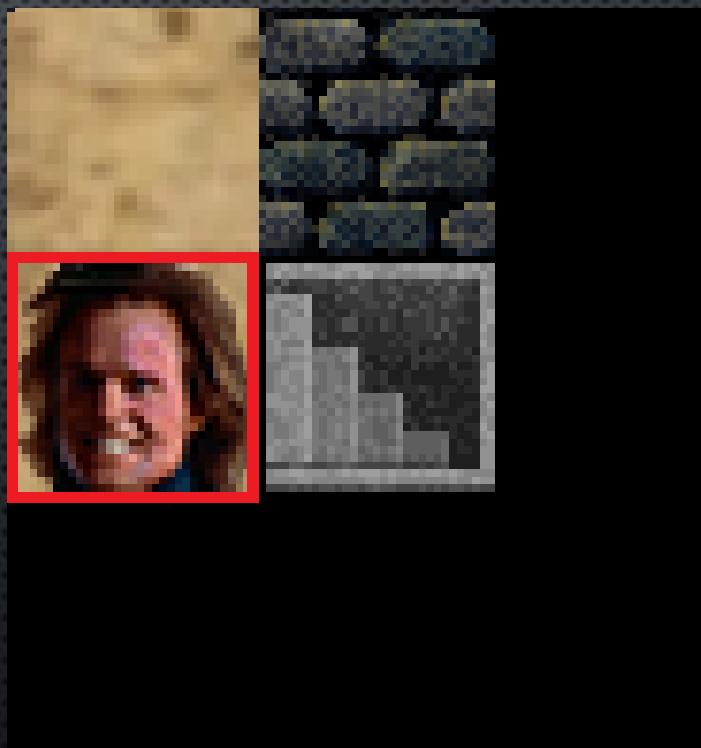
Class Maze : Afficher les sprites

```
row = 0
for line in self.structure:
    column = 0
    for letter in line:
        y = row * sprite_size
        x = column * sprite_size
        if letter == 'x':    # x = wall
            window.blit(wall, (x, y))
        elif letter == 's':  # s = starting position
            window.blit(depart, (x, y))
            self.starting_position = (x, y)
        elif letter == '0':  # 0 = corridor
            self.corridor.append((x, y))
        elif letter == 'e':
            window.blit(sortie, (x, y))
            self.exit_position = (x, y)
        column += 1
    row += 1
```

1. Parcourir chaque liste dans `self.structure`
2. Parcourir les strings dans la liste
3. Calculer l'emplacement en pixel
4. Afficher diffèrent spirites par rapport aux Strings
5. Garder la position de départ, arrivée et couloir
6. Incrémenter la colonne et répéter
7. Incrémenter la ligne et répéter

class Guard:

Cette classe désigne l'emplacement du garde en cherchant les cases libres autour de la sortie.

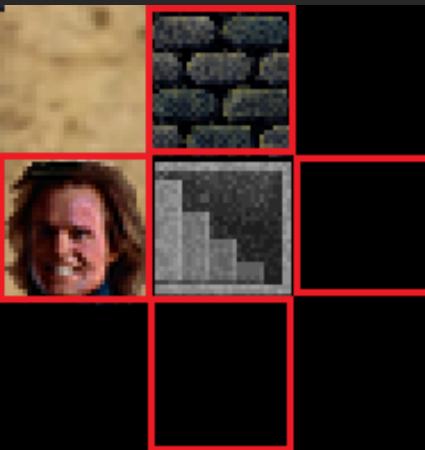


class Gard:

1. Trouver les case autour de la sortie

```
def guardPosition(self, window):  
  
    possible_position = {}  
  
    row = int(self.maze.exit_position[1] / sprite_size)  
    column = int(self.maze.exit_position[0] / sprite_size)  
  
    x_axis = self.maze.exit_position[0]  
    y_axis = self.maze.exit_position[1]  
  
    up = row - 1  
    down = row + 1  
    left = column - 1  
    right = column + 1
```

- 1) Trouver des indexes de la sortie.
- 2) Calculer des indexes autour de la sortie.



```
class Gard:
```

2. Eliminer les cases inexistantes

```
up = row - 1  
down = row + 1  
left = column - 1  
right = column + 1  
  
# Find coordinates of blocks around the exit and add them to possible_position  
try:  
    possible_position['up'] = self.maze.structure[up][column]  
except IndexError:  
    pass  
try:  
    possible_position['down'] = self.maze.structure[down][column]  
except IndexError:  
    pass
```



- 1) Ajouter les cases existantes dans une dictionnaire
- 2) Si la case n'existe pas, ignorer l'erreur.

```
class Gard:
```

3. Eliminer les mur

```
# Checking for blocks which has value of '0'  
check_position = {k: v for k, v in possible_position.items() if v == '0'}
```

4. Choisir une case

```
# Selects a random block by passing a key from the check_position  
gardien_case = random.choice(list(check_position.keys()))
```

```
# Calculate postion of the guard to be displayed on the screen  
if gardien_case == 'up':  
    y_axis -= sprite_size  
    self.gardien_position = (x_axis, y_axis)
```

```
self.maze.corridor.remove(self.gardien_position)
```

```
class Player:
```

Affiche MacGyver et calculer sa nouvelle position
contrôlée par l'utilisateur.

class Player :

1. Afficher MacGyver sur l'emplacement de départ

```
class Player:

    def __init__(self, maze):
        # position of player in terms of coordinates and pixels
        self.maze = maze
        self.player_position = self.maze.starting_position
```

class Player :

2. déplacements de MacGyver

```
def move(self, direction):
    # Calculate opstion of player in coordiates and pixels
    row = int(self.player_position[1] / sprite_size)
    column = int(self.player_position[0] / sprite_size)
    x_axis = self.player_position[0]
    y_axis = self.player_position[1]

    # Directin choosen by player
    if direction == 'up':
        # cannot go over the boarder of screen
        if y_axis > 0:
            # cannot go through the wall
            if self.maze.structure[row - 1][column] != "x":
                # calculate new postion
                y_axis -= sprite_size
                # passing new postion as a tuple to a variable
                self.player_position = (x_axis, y_axis)
```

- 1) Argument ‘direction’ prendre une valeur de string si l’utilisateur appuie sur les touches directionnelles du clavier
- 2) Trouver l’index de la case actuelle de MacGyver
- 3) Limiter le déplacement uniquement au couloir.
- 4) Si les conditions sont remplies, calculer la nouvelle position.

class Item:

Affiche les objets non obtenus par MacGyver et
compte les objets obtenus.

class Item:

1. Choisir les coordonnées aléatoirement

```
# Randomly reate postion of items
class Item:

    def __init__(self, maze,):
        self.maze = maze
        self.score = 0

        item1 = pygame.image.load(arrow).convert_alpha()
        item2 = pygame.image.load(tube).convert_alpha()
        item3 = pygame.image.load(ether).convert_alpha()

        self.item_name = [item1, item2, item3]
        self.item_position = []
        self.items_picked_up = []

    def itemPosition(self):
        for i in range(len(self.item_name)):
            item = random.choice(self.maze.corridor)
            self.item_position.append(item)
            self.maze.corridor.remove(item)
```

- 1) Créer une liste qui contient les images des objets.
- 2) Parcourir la longueur de liste
- 3) Sélectionner des coordonnées aléatoirement de ‘self.maze.corridor’
- 4) Ajouter les coordonnées à ‘self.item_position’
- 5) Supprimer les coordonnées de ‘self.maze.corridor’
- 6) Répéter

Class Item:

2. Afficher les objets

```
def displayItems(self, player, window):  
  
    self.player = player  
    self.window = window  
  
    item_obtained = False  
    for i in range(len(self.item_position)):  
        # Matching item name and position from two lists and display  
        self.window.blit(self.item_name[i], self.item_position[i])  
        # Setting name and position of item obtained to a variable  
        if self.player.player_position == self.item_position[i]:  
            del_item_position = self.item_position[i]  
            del_item_name = self.item_name[i]  
            # Passes item picked up to a new list in order  
            self.items_picked_up.append(del_item_name)  
            # tracks the score  
            self.score += 1  
            item_obtained = True  
  
    # Removes item obtained from the lists  
    if item_obtained:  
        self.item_position.remove(del_item_position)  
        self.item_name.remove(del_item_name)
```

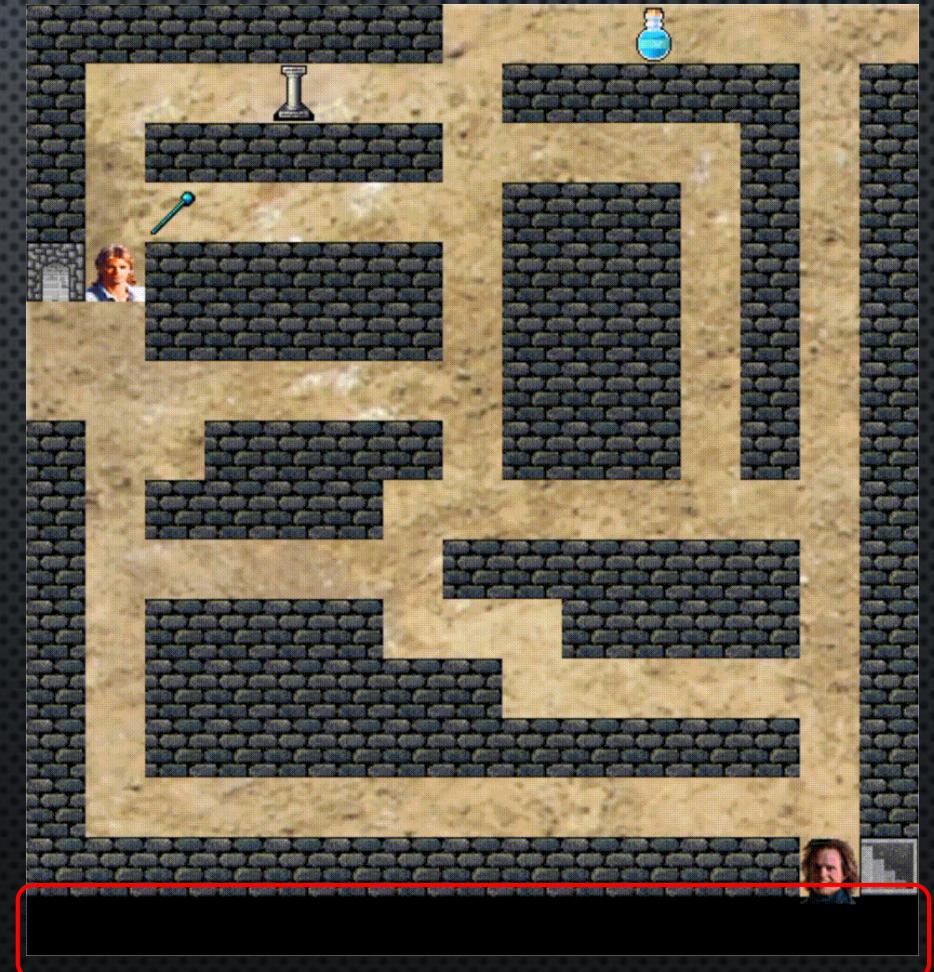
- 1) Obtenir la longueur de la liste ‘self.item_position’
- 2) Associer une image et une position puis afficher l'image.
- 3) Si le joueur arrive à la position de l'image.
 - a. Copier les valeurs
 - b. Ajouter les objet obtenu dans la liste ‘self.items_picked_up’
 - c. Compter le point
 - d. item_obtained = True
- 4) Répéter
- 5) Si item_obtained == True, supprimer l'objet et la position de sa liste.

class Item:

3. Afficher les objets obtenus sur le panneau d'affichage dans l'ordre d'obtention

```
# Display Items obtained on score board
for i in range(len(self.items_picked_up)):
    self.window.blit(self.items_picked_up[i], (sprite_size * i, screen_size))
# Displays a finish.png on score board
if self.score == 3:
    end = pygame.image.load(finish).convert()
    self.window.blit(end, (sprite_size * 3, screen_size))
```

1. Parcourir la longueur de la liste qui contient les images des objets obtenus
2. Les afficher sur le panneau d'affichage
3. Si 'self.score == 3, afficher l'instruction.



labyrinthe.py

- crée la fenêtre
- charge et affiche les images
- crée et gère les instances
- définit les conditions pour le jeu
 - Fermeture de la fenêtre
 - Contrôler MacGyver
 - Décider si MacGyver a gagné ou perdu

constantes.py

```
# constante variables
SPRITE_SIZE = 32
TILE = 15
SCREEN_SIZE = int(TILE * SPRITE_SIZE)
START = 'images/depart.png'
FOND = 'images/fond.jpg'
MUR = 'images/mur.png'
ICON = 'images/dk_bas.png'
MACGYVER = 'images/macgyver.png'
ARROW = 'images/arrow.png'
ETHER = 'images/ether.png'
TUBE = 'images/tube.png'
EXIT = 'images/exit.png'
GARDIEN = 'images/gardien.png'
```

- Contient toutes les constantes.

Pour améliorer

- Ajouter du son
- Plusieurs niveaux
- L'écran d'accueil
- Un garde mobile