Documentation technique

- Pour l'affichage du fond « la machine à sous » nous avons utilisé une planche d'image :

Que nous avons découpée avec une boucle FOR et mise dans une liste :

```
# Decoupage image pour arriere plan
for i in range(16):
    tile = machine_a_sous_tile.subsurface(pygame.Rect(i*c, 0, c, c))  # Découpage de la ligne d'indice 0
    tiles.append(tile)  # Ajout de chaque image dans la liste tiles
```

Puis on a créé une autre liste avec les indices correspondants aux images de la planche découpée :

```
# Liste d'affichage des tuiles du background de la machine à sous
tile_map = [[0, 0, 0, 0, 0, 0, 9, 8, 8, 8, 8, 8, 7, 0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 2, 14, 14, 14, 14, 14, 6, 0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 3, 4, 4, 4, 4, 5, 0, 0, 0, 0, 0, 0],
            [0, 0, 0, 9, 8, 8, 8, 12, 8, 8, 8, 12, 8, 8, 8, 7, 0, 0, 0],
            [0, 0, 0, 2, 1, 1, 1, 13, 1, 1, 1, 13, 1, 1, 1, 6, 0, 0, 0],
            [0, 0, 0, 2, 1, 1, 1, 10, 1, 1, 1, 10, 1, 1, 1, 6, 0, 0, 0],
            [0, 0, 0, 2, 1, 1, 1, 15, 1, 1, 1, 15, 1, 1, 1, 6, 0, 0, 0],
            [0, 0, 0, 2, 1, 1, 1, 13, 1, 1, 1, 13, 1, 1, 1, 6, 0, 0, 0],
            [0, 0, 0, 3, 4, 4, 4, 11, 4, 4, 4, 11, 4, 4, 4, 5, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
```

Et enfin affiché cette liste avec une double boucle FOR :

```
# Double boucle pour afficher le contenu du tableau
for colonne in range(len(tile_map[0])):
    for ligne in range(len(tile_map)):
        # Affiche du tableau sur la fenêtre
        screen.blit(tiles[tile_map[ligne][colonne]], dest: (c*colonne, c*ligne))
```

- La gestion des boutons se fait avec des classes qui s'occupent d'initialiser, d'afficher et de vérifier les collisions des boutons

```
class Button:
    def __init__(self, x, y, width, height, text, action=None, color=None, color_border=None,
                 border_ext=None, border_int=None, color_font=None, taille=None):
        # Crée un rectangle pour le bouton avec les coordonnées et les dimensions fournies
        self.rect = pygame.Rect(x, y, width, height)
        self.text = text  # Stockage de texte à afficher sur le bouton
        self.font = pygame.font.Font( name: None, size=taille)  # Initialisation de la police pour le texte du bouton
        self.action = action  # Stockage de l'action à exécuter lorsque le bouton est cliqué
        self.color = color  # Stockage de la couleur de l'intérieur du bouton
        self.color_border = color_border  # Stockage de la couleur de la bordure du bouton
        self.border_ext = border_ext  # Stockage de la taille du coin de la bordure extérieur du bouton
        self.border_int = border_int  # Stockage de la taille du coin de la bordure intérieur du bouton
        self.color_font = color_font  # Stockage de la couleur du texte du bouton

    16 usages
    def draw(self, screen, color=None, color_border=None, border_ext=None, border_int=None, color_font=None):
        # Dessine le fond du bouton avec une couleur verte (GREEN)
        pygame.draw.rect(screen, self.color, self.rect, border_radius=self.border_ext)
        # Dessine une bordure plus claire autour du bouton
        pygame.draw.rect(screen, self.color_border, self.rect.inflate(-6, -6), border_radius=self.border_int)
        # Rend le texte du bouton avec la police spécifiée et la couleur DARK_GREEN
        text_surface = self.font.render(self.text, antialias: True, self.color_font)
        text_rect = text_surface.get_rect(center=self.rect.center)  # Centre le texte au milieu du bouton
        screen.blit(text_surface, text_rect)  # Affiche le texte sur l'écran

    16 usages
    def handle_event(self, event):
        if event.type == pygame.MOUSEBUTTONDOWN:  # Vérifie si un événement de clic de souris a eu lieu
            # Vérifie si la position du clic de souris est à l'intérieur des limites du bouton
            if self.rect.collidepoint(event.pos):
                if self.action:  # Vérifie si une action est associée au bouton
                    self.action()  # Exécute l'action associée au bouton
```

Chaque bouton est modifiable car dans l'initialisation on entre des variables à compléter lors de la création du bouton :

```
# Bouton pour ouvrir le menu1
button2 = Button((SCREEN_WIDTH - 150) // 2 + 340, (SCREEN_HEIGHT - 60) // 2 - 275,  width: 180,  height: 60,
                 text: "MINI-JEUX", action=menu1, color=GREEN, color_border=LIGHT_GREEN, border_ext=28,
                 border_int=20, color_font=DARK_GREEN, taille=36)
```

- La vérification des gains se fait avec une liste qui vérifie les valeurs des rouleaux de la machine à sous en cours :

```
# Liste de verification des gain
lst_verif = [[7, 7, 7], [1, 1, 1], [9, 9, 9], [8, 8, 8], [4, 4, 4], [13, 13, 13], [12, 12, 12], [11, 11, 11],
             [5, 5, 5], [4, 4, 4], [2, 2, 2], [3, 3, 3], [10, 10, 10], [5, 7, 7], [1, 1, 5], [8, 9, 9], [5, 8, 8],
             [5, 4, 6], [5, 13, 13], [5, 12, 12], [5, 11, 11], [4, 4, 5], [2, 2, 5], [3, 3, 5],
             [5, 10, 10], [5, 5, 7], [1, 5, 5], [5, 5, 9], [5, 5, 8], [5, 5, 4], [5, 5, 13], [5, 5, 12],
             [5, 5, 11], [4, 5, 5], [2, 5, 5], [3, 5, 5], [5, 5, 10], [8, 6, 8], [8, 6, 9], [6, 8, 8],
             [8, 8, 9], [6, 9, 9], [8, 9, 9], [6, 8, 9], [5, 6, 8], [5, 6, 9], [5, 8, 9], [11, 11, 12],
             [11, 11, 13], [11, 12, 12], [12, 12, 13], [11, 13, 13], [12, 13, 13], [11, 12, 13], [5, 11, 12],
             [5, 11, 13], [5, 12, 13]]
```

Puis on récupère l'indice de la combinaison actuelle si elle est dans la liste et on récupère la valeur du gain, dans une autre liste qui comporte tous les gains, avec cet indice :

```
# Liste des gains de la machine à sous
recompense = [1000, 500, 250, 250, 250, 200, 200, 200, 150, 100, 50, 50, 50, 1000, 500, 250, 250, 250, 200,
              200, 200, 100, 50, 50, 50, 1000, 500, 250, 250, 250, 200, 200, 200, 100, 50, 50, 50, 200,
              200, 200, 200, 200, 200, 200, 150, 150, 150, 150, 150, 150, 150, 150, 150, 150, 100, 100, 100]
```

Une sauvegarde du nombre de pièce s'effectue à chaque lancé et à chaque fin de mini-jeu, la sauvegarde se fait avec un fichier que l'on ouvre en mode w (écriture) mais qui enlève également le contenue puis on ajoute au fichier le contenu de la variable des pièces :

```
def save_file():
    file1 = open("file/coins", 'w')  # ouverture du fichier des coins en mode écriture
    file1.write(str(coins))  # Mise à jour du fichier de coins
    file1.close()  # Fermeture du fichier de coins
```

A chaque ouverture du programme le fichier et ouvert en mode r (lecture) afin de récupérer le contenu pour le texte qui affiche le nombre de pièces actuel :

```
file = open("file/coins", "r")  # Ouverture du fichier du stock des coins en mode lecture
coins = int(file.read())  # Variable des coins mise à la valeur du fichier de stock des coins
file.close()  # Fermeture du fichier du stock des coins
```

- Les menus sont des fonctions qui affichent une image sur la machine à sous dès le bouton cliqué avec une variable booléenne :

```
# Fonction ouverture de la notice
2 usages
def notice():
    global image_visible, text_visible, but_visible, run, text2_visible, but_activer
    image_visible = True  # L'image de la notice s'affiche
    text_visible = False  # Le texte disparait
    text2_visible = False  # Le texte des coins disparait
    but_visible = True  # Le bouton de retour à la machine à sous apparait
    run = False  # Désactivation des touches du clavier
    but_activer = False  # Désactivation des boutons de la machine à sous
```

```
# Fonction ouverture du menu 1
2 usages
def menu1():
    global image_visible1, text_visible, but_visible1, run, but_activer  # D
    image_visible1 = True  # L'image du menu1 s'affiche
    text_visible = False  # Le texte disparait
    but_visible1 = True  # Le bouton de retour à la machine à sous apparait
    run = False  # Désactivation des touches du clavier
    but_activer = False  # Désactivation des boutons de la machine à sous
```

Sur ces menus apparaissent des boutons comme un bouton retour à la machine à sous :

Ils font disparaitre le menu qui est en cours avec des fonctions retour :

```
# Fonction de retour à la machine à sous et ferme la notice
2 usages
def back_notice():
    global image_visible, text_visible, but_visible, run, text2_visible, but_activer
    image_visible = False  # L'image de la notice s'enlève
    text_visible = True  # Le texte réapparait
    text2_visible = True  # Le texte des coins réapparait
    but_visible = False  # Le bouton de retour à la machine à sous disparait
    run = True  # Reactivation des touches du clavier
    but_activer = True  # Reactivation des boutons de la machine à sous
```

```
# Fonction retour à la machine à sous et fermeture du menu 1
2 usages
def back_menu1():
    global image_visible1, text_visible, but_visible1, run, but_activer  # Déf
    image_visible1 = False  # L'image du menu1 s'enlève
    text_visible = True  # Le text réapparait
    but_visible1 = False  # Le bouton de retour à la machine à sous disparait
    run = True  # Reactivation des touches du clavier
    but_activer = True  # Reactivation des boutons de la machine à sous
```

- Gestion des collisions :

La gestion des collisions des boutons se fait avec la fonction handle_event() dans la classe Button, cette fonction vérifie si la souris a cliqué dans les limites du bouton :

```
def handle_event(self, event):
    # Vérifie si un événement de clic de souris a eu lieu
    if event.type == pygame.MOUSEBUTTONDOWN:
        # Vérifie si la position du clic de souris est à l'intérieur des limites du bouton
        if self.rect.collidepoint(event.pos):
            # Vérifie si une action est associée au bouton
            if self.action:
                # Exécute l'action associée au bouton
                self.action()
```

Si le bouton est cliqué la fonction appelle la fonction associée, définie dans l'initialisation du bouton au début de la classe.

La gestion des collisions dans les mini-jeux est divisé en 2 catégories la première celle où on vérifie si le joueur a sa position en x dans un intervalle précis :

```
if x_perso3 >= 1125:  # Si la position en X du personnage est égale à celle de l'arrivée
    ecran_gain = True  # Affichage de l'écran de gain
    resultat = "GAGNÉ"  # Le resultat affiche gagné
```

La deuxième où on crée un rectangle sur les personnages et objets et on vérifie si les deux rectangles se touchent avec la fonction colliderect() de pygame :

```
# Création du rectangle de collision ennemie (haie)
rectangle_ennemi = pygame.Rect(x_collision_obj, y_obj, largeur_ennemi, hauteur_ennemi)
pygame.draw.rect(screen, BLACK, rectangle_joueur)  # Affichage du rectangle joueur
pygame.draw.rect(screen, BLACK, rectangle_ennemi)  # Affichage du rectangle ennemi
pygame.time.Clock().tick(50)  # Programme mis à 50 fps (image par seconde)
if rectangle_joueur.colliderect(rectangle_ennemi):  # Si rectangle joueur touche rectangle ennemi
    resultat_coins = compte_pts  # Récuperation des gains
    coins = coins + compte_pts  # Ajout des gains aux coins
```