Documentation technique :

Alquerque

Sommaire[1]

[1] Ctrl + clic droit pour se diriger vers les chapitres que vous souhaitez

Choix du langage et librairie graphique :

Pour ce projet j'ai décidé d'utiliser le langage python. Il est très utilisé pour faire des petits jeux indépendants. De plus j'ai une plus grande expérience en python qu'en C que j'ai appris à coder qu'en première année de bachelor.

Donc pour conclure sur le langage de programmation ce choix fut rapide pour le python car j'avais beaucoup plus d'expériences.

Librairie graphique :

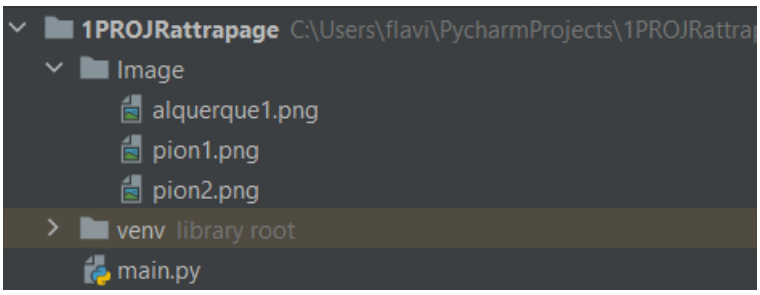
J'ai choisi pygame pour la librairie graphique :

- Premièrement car il y a une plus grande facilité dans le graphisme car nous avons juste besoin d'un plateau en png pour qu'on est déjà le plateau et qu'on puisse poser les pions

- Il y a une plus grande facilité à coder que le Tkinter

- Et enfin il rend plus facilement les images cliquables c'est-à-dire qu'on peut les sélectionner facilement

Description des structures de données :



Les dossiers du jeu s'organisent comme ceci donc avec main.py qui possède le code en entier tandis que nous avons un dossier image qui contient le plateau du jeu mais aussi les pions.

```python
class Player(pygame.sprite.Sprite):
    def __init__(self, player):
        super().__init__()
        self.player = player
        self.image = None
        if player == 2:
            self.image = pygame.transform.scale(pygame.image.load("Image/pion1.png"), (60, 60))
        if player == 1:
            self.image = pygame.transform.scale(pygame.image.load("Image/pion2.png"), (60, 60))
        elif player == 0:
            self.image = pygame.transform.scale(pygame.image.load("Image/pion2.png"), (60, 60))

    def set(self, player):
        self.player = player
        if player == 2:
            self.image = pygame.transform.scale(pygame.image.load("Image/pion1.png"), (60, 60))
        if player == 1:
            self.image = pygame.transform.scale(pygame.image.load("Image/pion2.png"), (60, 60))
        elif player == 0:
            self.image = pygame.transform.scale(pygame.image.load("Image/pion2.png"), (60, 60))


tableau = [[Player(2), Player(2), Player(2), Player(2), Player(2)],
           [Player(2), Player(2), Player(2), Player(2), Player(2)],
           [Player(2), Player(2), Player(0), Player(1), Player(1)],
           [Player(1), Player(1), Player(1), Player(1), Player(1)],
           [Player(1), Player(1), Player(1), Player(1), Player(1)]]
```

Donc notre « class » s'appelle Player, donc tous d'abord j'initialise les fonctions pour qu'elles soient liées.

Le « set(self, player) : » servira pour le mouvement des pions tandis que le « self.image » va servir à définir les images dans le tableau juste en bas du screen avec des nombres.

Le pion1 représente les pions noirs et les pion2 représentent les pions blancs et pour ce qui est du « player == 0 » c'est un pion blanc pour le centre mais invisible pour nous lors de la création du plateau mais il sert à avoir quelque chose de cliquable sur le plateau même quand il n'y a pas de pion.

Présentation des algorithmes de gestion de la grille :

```
tableau = [[Player(2), Player(2), Player(2), Player(2), Player(2)],
           [Player(2), Player(2), Player(2), Player(2), Player(2)],
           [Player(2), Player(2), Player(0), Player(1), Player(1)],
           [Player(1), Player(1), Player(1), Player(1), Player(1)],
           [Player(1), Player(1), Player(1), Player(1), Player(1)]]


def InitBoard():
    board = pygame.image.load("Image/alquerque1.png")
    board_rect = board.get_rect(center=(300, 298.5))
    game.blit(board, board_rect)

    for i in range(len(tableau)):
        for j in range(len(tableau[i])):
            if tableau[i][j].image is not None:
                tableau[i][j].rect = tableau[i][j].image.get_rect(center=(122 * (j + 0.46), 122 * (i + 0.47)))
                if tableau[i][j].player != 0:
                    game.blit(tableau[i][j].image, tableau[i][j].rect)
    pygame.display.flip()
```

On reprend le tableau d'au-dessus pour définir les pions et comment ils devront se placer.

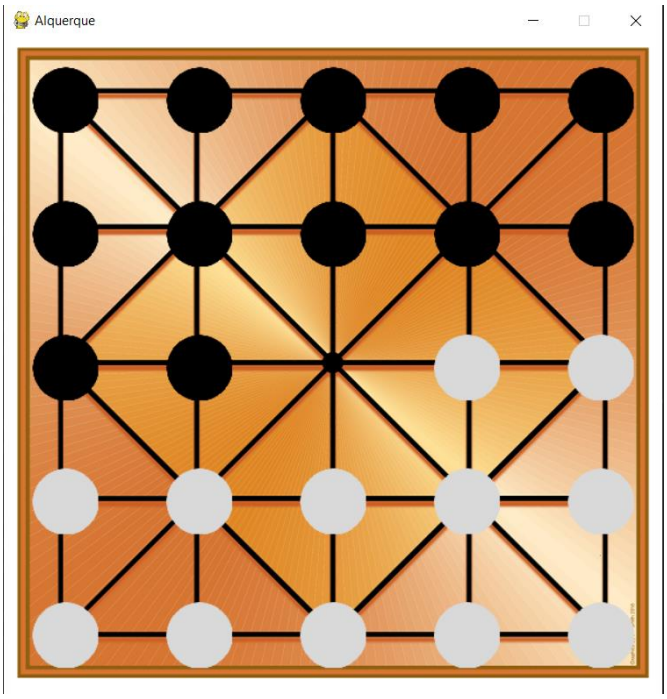
La fonction InitBoard commence par définir le plateau avec une image, on va aussi définir le centre pour que les pions se placent déjà au centre du plateau mais ils ne sont toujours par répartie correctement sur les cases du plateau.

La seconde partie on va parcourir notre tableau grâce à deux boucles et que s'il n'est pas None c'est-à-dire qu'il contient par exemple des images alors le programme continue. (Pour le « .image » il faut revenir sur la Class c'est ici qu'on définit).

Après ça on reprends notre plateau et on va définir l'espace de chaque pion pour qu'il s'écarte en hauteur et largeur afin d'avoir le bon espace entre chaque pion et enfin pour les axes i et j on les additionnes avec des petits nombre pour déplacer tous le plateau afin qu'il soit aligner avec toutes les cases et les pions.

Le dernier « if » sert à définir l'image en quelque sorte invisible pour la case vide car sur le plateau on ne verra pas le pion mais pour l'ordinateur le pion existe bien et nous aideras dans le mouvement.

On obtient alors ceci :




```
def pion():
    for a in range(len(tableau)):
        for b in range(len(tableau[a])):
            if tableau[a][b].image is not None and (
                    tableau[a][b].rect.collidepoint(event.pos[0], event.pos[1])) is True:
                return True
    return False


def move(event: MOUSEMOTION):
    for y in range(len(tableau)):
        for x in range(len(tableau[y])):
            if tableau[y][x].rect.collidepoint(event.pos[0], event.pos[1]) is True:
                return y, x
    return None
```

C’est deux dernières fonctions vont m’aider pour le mouvement mais aussi à me repérer dans la grille.

La fonction pion va renvoyer « True » si je clique sur une image sinon elle renvoie false, ça me permet de savoir si les images sont cliquables sur le plateau.

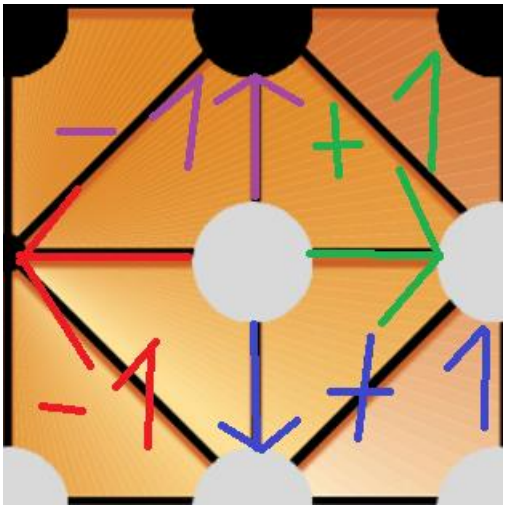
La fonction move sert à définir les coordonnées sur le tableau afin de me repérer où je suis mais aussi de m’aider à déplacer plus tard les pions vers une

autre case. Attention les coordonnées sont inversées c'est-à-dire qu'on aura (y, x).

Donc après avoir parlé de l'implémentation de la grille je peux m'occuper des fonctions qui serviront à déplacer mais aussi à capturer les pions adverses.

Présentation des algorithmes implémentant les déplacements et les captures :

Pour faire le déplacement je suis parti d'un principe simple avec les axes que j'ai pu créer lors de l'implémentation de la grille de jeux :



Nos axes sont inversés donc on part du haut et on ajoute 1 pour aller vers le bas, maintenant que la méthode est expliquée on va passer à l'implémentation du code :

```
def case(player: tuple):
    voisin = []
    try:
        if player[0] - 1 >= 0 and tableau[player[0] - 1][player[1]]:
            voisin.append((player[0] - 1, player[1]))
        if player[0] + 1 >= 0 and tableau[player[0] + 1][player[1]]:
            voisin.append((player[0] + 1, player[1]))
        if player[1] - 1 >= 0 and tableau[player[0]][player[1] - 1]:
            voisin.append((player[0], player[1] - 1))
        if player[1] + 1 >= 0 and tableau[player[0]][player[1] + 1]:
            voisin.append((player[0], player[1] + 1))
        if (player[0] % 2 != 0 and player[1] % 2 != 0) or (player[0] % 2 == 0 and player[1] % 2 == 0):
            if player[0] - 1 >= 0 and player[1] - 1 >= 0 and tableau[player[0] - 1][player[1] - 1]:
                voisin.append((player[0] - 1, player[1] - 1))
            if player[0] + 1 >= 0 and player[1] + 1 >= 0 and tableau[player[0] + 1][player[1] + 1]:
                voisin.append((player[0] + 1, player[1] + 1))
            if player[0] + 1 >= 0 and player[1] - 1 >= 0 and tableau[player[0] + 1][player[1] - 1]:
                voisin.append((player[0] + 1, player[1] - 1))
            if player[0] - 1 >= 0 and player[1] + 1 >= 0 and tableau[player[0] - 1][player[1] + 1]:
                voisin.append((player[0] - 1, player[1] + 1))
    except IndexError:
        pass
    return voisin
```

La fonction « case » sert de base principale pour les déplacements de pions. Le principe est très simple, j'utilise une liste « voisin » où j'ajouter les modifications de coordonnées si on veut aller en haut, en bas etc… Pour les diagonales je me base sur le même principe mais cette fois il faut faire attention à que les pions ne puissent pas aller vers des cases qui sont diagonales à eux sans un trait pour les relier, c'est pour ça qu'il y a une condition qui est définis grâce à nos coordonnées dans la fonction vue plus haut.

```
def movepos(pion1: tuple, pion2: tuple):
    if pion2 in case(pion1) and tableau[pion2[0]][pion2[1]].player == 0 and tableau[pion1[0]][
        pion1[1]].player == turnplayer:
        return True
    else:
        return False
```

La fonction « movepos » sert quant à elle a afficher les changements sur le plateau donc à l'actualiser afin de permettre au joueur de voir son plateau en direct. Le fonctionnement est très simple, c'est qu'à chaque fois que c'est le tour d'un joueur il va alors que réinitialiser le plateau et afficher le changement.

Capture :

```
def capture(pion1: tuple, pion2: tuple):
    for items in case(pion1):
        if tableau[pion1[0]][pion1[1]].player == turnplayer and tableau[pion2[0]][
            pion2[1]].player == 0 and items in case(pion2) and tableau[items[0]][
            items[1]].player == pionadverse() and (
                (((pion1[0] + pion2[0]) / 2, (pion1[1] + pion2[1]) / 2) == items) or (
                pion1[0] == pion2[0] == items[0] or pion1[1] == pion2[1] == items[1])):
            return items
    return None
```

La fonction « Capture » va servir à savoir si oui ou non il est possible de capturer un pion sur le plateau, si oui alors la fonction va retourner item donc permettre de capturer un pion et sinon elle retourne None car il n'y a aucune possibilité.

```
def capturer(pion1: tuple, pion2: tuple, pion3: tuple):
    tableau[pion1[0]][pion1[1]].set(0)
    tableau[pion2[0]][pion2[1]].set(turnplayer)
    tableau[pion3[0]][pion3[1]].set(0)
```

La fonction « capturer » va servir initialiser le plateau quand un pion est capturé, donc avoir une interface graphique qui s'actualise.

```
def secondcapture():
    for i in range(len(tableau)):
        for j in range(len(tableau[i])):
            for k in case((i, j)):
                for l in case((k[0], k[1])):
                    if tableau[i][j].player == turnplayer and l not in case((i, j)) and capture((i, j), l) is not None:
                        return True
    return False
```

La fonction « secondcapture » sert principalement à bloquer ou plutôt supprimer un pion si jamais on pouvait capturer un autre pion ce qui a pour but d'obliger le joueur à capturer le pion. Elle avait pour second but aussi de faire l'enchaînement de capture mais je n'ai pas réussi à le faire.

```
def pionadverse():
    if turnplayer == 1:
        return 2
    else:
        return 1
```

Pour l'alternance des joueurs.

```
def endgame():
    for i in range(len(tableau)):
        for j in range(len(tableau[i])):
            if tableau[i][j].player == turnplayer:
                return False
    return True
```

Pour relancer une partie de jeux quand il n'y aura plus de pions noirs ou blanc sur le plateau.

Tous nos fonctions vont se reliés dans une seule et même boucle while qui aidera au déroulement du jeux dans toutes sa partie et ainsi nous aider à comprendre s'il y a un bug dans les fonctions.

```
continuer = True
case_nei, second_nei = None, None
turnplayer = 1
while continuer:
    InitBoard()
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            continuer = False
        elif event.type == MOUSEBUTTONDOWN:
            click = move(event)
            if click is None:
                break
            if case_nei is None:
                case_nei = click
            elif case_nei is not None and click != case_nei and second_nei is None:
                second_nei = click
            print(click)
            if case_nei is not None and second_nei is not None:
                mp = movepos(case_nei, second_nei)
                if mp is True and secondcapture() is True:
                    tableau[case_nei[0]][case_nei[1]].set(0)
                    tableau[second_nei[0]][second_nei[1]].set(0)
                elif mp is True and secondcapture() is False:
                    tableau[second_nei[0]][second_nei[1]].set(turnplayer)
                    tableau[case_nei[0]][case_nei[1]].set(0)
                if mp is False and capture(case_nei, second_nei) is not None:
                    capturer(case_nei, second_nei, capture(case_nei, second_nei))
```

```
                turnplayer = pionadverse()

                case_nei, second_nei = None, None

            if endgame() is True:
                tableau = [[Player(2), Player(2), Player(2), Player(2), Player(2)],
                           [Player(2), Player(2), Player(2), Player(2), Player(2)],
                           [Player(2), Player(2), Player(0), Player(1), Player(1)],
                           [Player(1), Player(1), Player(1), Player(1), Player(1)],
                           [Player(1), Player(1), Player(1), Player(1), Player(1)]]
                turnplayer = 1
pygame.quit()
```

La boucle while va nous servir à faire appel à toutes les fonctions vues plus haut pour fonctionner le jeu, l'avantage c'est que cela nous permet de faire nous-même le déroulement du jeu.

Difficultés rencontrées :

Plusieurs difficultés ont eux lieu pendant la programmation du projet.

Le premier point est celui du déplacement vers une case vide, les faire bouger étaient simples mais la difficulté était d'interdire certaines diagonales à des pions. C'est pour cela que j'ai utilisé une condition afin d'empêcher ça mais pour ce faire j'ai dû m'aider en créant mes axes afin de savoir comment interdire leur déplacement.

Le second point fut l'enchaînement de capture que je n'ai pas réussi dans ce projet. Au départ je voulais faire un break pour dire que s'il y a une autre

possibilité de capturer alors il reste sur le même joueur mais le problème était que le break arrêter le code et donc laisser au deux joueurs deux tours au lieu d'enchaîner tour par tour et l'autre bug et que si on cliquait deux fois d'affilés sur un pion alors on pouvait le bouger n'importe où. Voilà pourquoi il n'y a pas de seconde capture.