Brice Aubineau

B.Eng.1

Documentation technique

Sommaire

Justification du choix du langage et de la librairie graphique.

Pour réaliser ce projet, j'ai décidé d'utiliser Python avec l'interface graphique Pygame, car python est un langage que je connais mieux que le C et Pygame est une librairie graphique avec laquelle je suis plus à l'aise qu'avec Tkinter, notamment grâce aux différents projets que j'ai pu effectuer auparavant.

Description des structures de données.

Dans ce projet il y a deux structures de données principales :

-la classe Pion perme

```
class Pion(pygame.sprite.Sprite):
    def __init__(self, image, joueur, rect, x, y):
        super().__init__()
        self.image = image
        self.joueur = joueur
        self.rect = rect
        self.x = x
        self.y = y
```

-le plateau de jeu contenant tous les pions

```
plateau = [[Pion(pionNoir, 1, pionNoir.get_rect(), 0, 0), Pion(pionNoir, 1, pionNoir.get_rect(), 0, 1),
            Pion(pionNoir, 1, pionNoir.get_rect(), 0, 2), Pion(pionNoir, 1, pionNoir.get_rect(), 0, 3),
            Pion(pionNoir, 1, pionNoir.get_rect(), 0, 4)],
           [Pion(pionNoir, 1, pionNoir.get_rect(), 1, 0), Pion(pionNoir, 1, pionNoir.get_rect(), 1, 1),
            Pion(pionNoir, 1, pionNoir.get_rect(), 1, 2), Pion(pionNoir, 1, pionNoir.get_rect(), 1, 3),
            Pion(pionNoir, 1, pionNoir.get_rect(), 1, 4)],
           [Pion(pionNoir, 1, pionNoir.get_rect(), 2, 0), Pion(pionNoir, 1, pionNoir.get_rect(), 2, 1),
            Pion(pionNeutre, 0, pionNeutre.get_rect(), 2, 2), Pion(pionBlanc, 2, pionBlanc.get_rect(), 2, 3),
            Pion(pionBlanc, 2, pionBlanc.get_rect(), 2, 4)],
           [Pion(pionBlanc, 2, pionBlanc.get_rect(), 3, 0), Pion(pionBlanc, 2, pionBlanc.get_rect(), 3, 1),
            Pion(pionBlanc, 2, pionBlanc.get_rect(), 3, 2), Pion(pionBlanc, 2, pionBlanc.get_rect(), 3, 3),
            Pion(pionBlanc, 2, pionBlanc.get_rect(), 3, 4)],
           [Pion(pionBlanc, 2, pionBlanc.get_rect(), 4, 0), Pion(pionBlanc, 2, pionBlanc.get_rect(), 4, 1),
            Pion(pionBlanc, 2, pionBlanc.get_rect(), 4, 2), Pion(pionBlanc, 2, pionBlanc.get_rect(), 4, 3),
            Pion(pionBlanc, 2, pionBlanc.get_rect(), 4, 4)]]
```

Présentation des algorithmes de gestion de la grille.

Pour ce qui est des algorithmes de gestion de la grille je n'en ai pas à proprement parlé car j'ai initialisé mon plateau avec les valeurs de bases des pions et ensuite les changements de pion se font dans ma fonction de mouvement. Jai malgré tout implémenté une fonction permettant un bon affichage du plateau de jeu ainsi que des pions.

```
def drawBoard():
    screen.blit(background, background.get_rect())
    for i in range(len(plateau)):
        for j in range(len(plateau[i])):
            plateau[i][j].rect.topleft = ((165 * (j + 0.15)), (155 * (i + 0.15)))
            if plateau[i][j].image != pionNeutre:
                screen.blit(plateau[i][j].image, plateau[i][j].rect)
    pygame.display.update()
```

Présentation des algorithmes implémentant les déplacements et les captures.

Au niveau des algorithmes permettant les déplacements ainsi que les captures de pions il y a 3 algorithmes principaux et 2 algorithmes annexes :

Les deux premiers sont les algorithmes "possible()" et "possibleCapture()" permettant de vérifier si un déplacement ou une capture d'un pion est possible.

```
def possible(pion1, pion2):
    if pion2 in getVoisinsList(pion1) and pion2.joueur == 0:
        return True
    return False


def possibleCapture(pion1, pion2):
    if ((pion1.x + pion2.x) % 2 == 0) and ((pion1.y + pion2.y) % 2 == 0):
        pionMid = plateau[math.floor((pion1.x + pion2.x) / 2)][math.floor((pion1.y + pion2.y) / 2)]
        pionMid.x = math.floor((pion1.x + pion2.x) / 2)
        pionMid.y = math.floor((pion1.y + pion2.y) / 2)
        if pion2.joueur == 0 and pionMid.joueur != pion1.joueur and pionMid.joueur != 0 and pionMid in getVoisinsList(
                pion1) and pionMid in getVoisinsList(pion2) and (
                (pion1.x < pionMid.x < pion2.x or pion1.x > pionMid.x > pion2.x) or (
                pion1.y < pionMid.y < pion2.y or pion1.y > pionMid.y > pion2.y)):
            return True
    return False
```

Le troisième algorithme, le plus important, pour permettre le mouvement d'un pion est la fonction "play()" se servant des deux fonctions précédentes pour vérifier si un déplacement ou une capture est possible et auquel cas l'effectuer.

```
def play(pion1, pion2):
    global player
    if len(otherCapture(player)) >= 1 and possible(pion1, pion2):
        pion1.joueur = 0
        pion1.image = pionNeutre
        player = next(players)
        return True
    elif possible(pion1, pion2):
        pion2.joueur = pion1.joueur
        pion2.image = pion1.image
        pion1.joueur = 0
        pion1.image = pionNeutre
        player = next(players)
        return True
    elif possibleCapture(pion1, pion2):
        plateau[math.floor((pion1.x + pion2.x) / 2)][math.floor((pion1.y + pion2.y) / 2)].joueur = 0
        plateau[math.floor((pion1.x + pion2.x) / 2)][math.floor((pion1.y + pion2.y) / 2)].image = pionNeutre
        pion2.joueur = pion1.joueur
        pion2.image = pion1.image
        pion1.joueur = 0
        pion1.image = pionNeutre
        if len(otherCapture(player)) >= 1:
            if (pion1, pion2) not in otherCapture(player):
                return True
        else:
            player = next(players)
            return True
    else:
        return False
```

On arrive enfin aux deux fonctions annexes :

La première est la fonction "getVoisinsList()" permettant de récupérer la liste des voisins pour un pion en prenant en compte les diagonales ou non en fonctions de la position du pion sur le plateau.

```
def getVoisinsList(pion):
    listeVoisins = []
    x = pion.x
    y = pion.y
    for i in range(-1, 2):
        for j in range(-1, 2):
            if (x + y) % 2 == 0:
                if 0 <= x + i <= len(plateau) - 1 and 0 <= y + j <= len(plateau) - 1 and (i != 0 or j != 0):
                    listeVoisins.append(plateau[x + i][y + j])
            else:
                if 0 <= x + i <= len(plateau) - 1 and 0 <= y + j <= len(plateau) - 1 and ((x + i == x) != (y + j == y)):
                    listeVoisins.append(plateau[x + i][y + j])
    return listeVoisins
```

La seconde est la fonction "otherCapture()" permettant de savoir si un joueur peut effectuer une autre capture sur le plateau et donc permettre un enchainement de capture.

```
def otherCapture(joueur):
    listeCapture = []
    for i in range(len(plateau)):
        for j in range(len(plateau[i])):
            if plateau[i][j].joueur == joueur:
                for k in getVoisinsList(plateau[i][j]):
                    for l in getVoisinsList(k):
                        if possibleCapture(plateau[i][j], l) and (
                                plateau[i][j], l) not in listeCapture:
                            listeCapture.append((plateau[i][j], l))
    return listeCapture
```

Difficultés rencontrées.

Tout au long de ce projet j'ai rencontré des difficultés. Les principales furent

- la capture car il fallait vérifier que toutes les conditions étaient bien réunies pour autoriser une capture de pion
- l'enchainement de capture car il fallait vérifier dans tous le tableau si un joueur pouvait capturer un pion
- le choix de la capture menant à l'élimination du plus de pions adverses, sur ce point j'ai effectué plusieurs tentatives infructueuses, c'est pour ça que cette fonctionnalité n'apparait pas dans le projet