JANCARD Arnaud
WATTRE Anthony

Jeu d’échec

Pourquoi on a choisis le sujet du jeu d’échec ?

Tous d’abord notre projet est un jeu qui se nomme les échecs. Ce projet a été amusant à réaliser au cours de l'année. Nous avons opté pour ce sujet car nous sommes tous les deux intéressés par les échecs et nous y jouons régulièrement. C'est un jeu qui demande beaucoup de réflexion et qui est très polyvalent, car il offre de nombreuses combinaisons possibles à apprendre et à réaliser pendant la partie. Pour résumer les règles les échecs sont un jeu pour deux joueurs. Chaque joueur contrôle seize pièces : un roi, une dame, deux tours, deux fous, deux cavaliers et huit pions. Le but est de mettre en échec et mat le roi adverse, en le menaçant sans possibilité de fuite. Le jeu se termine par un échec et mat, une partie nulle (par exemple en cas de pat où aucun joueur ne peut jouer légalement), ou par abandon. Nous avons beaucoup joué tous les deux à un jeu qui se nomme chess.com. En effet ce jeu multijoueur nous a intéressé et nous as poussé a faire ce projet. Malgré la complexité du projet et les nombreuses fonctions qu'il a fallut pour le réaliser on a réussi a faire une interface d'un échiquier avec des pièces a leur endroit respectif ainsi que leur déplacement.

Qui a fais quoi ?

Arnaud:

Sur le jeu d'échecs, j'ai contribué en créant la classe Piece, où j'ai défini les caractéristiques et comportements communs à toutes les pièces du jeu, telles que le type de pièce (roi, dame, tour, fou, cavalier, pion) et la couleur (blanc ou noir). Ensuite, j'ai également participé à la conception de la classe Couleur pour gérer les attributs liés à la couleur des pièces, ce qui est essentiel pour différencier les pièces des deux joueurs. Par la suite, j'ai pris en charge une partie significative du développement de la classe Jeu. J'ai commencé par implémenter la structure de base, qui reprenait les principes définis dans la classe Piece .

Anthony:

Sur le jeu d'échecs, j'ai contribué dans le développement de la classe Nouveau_Plateau. Cette classe est essentielle pour gérer le plateau de jeu et garantir le bon déroulement des parties. J'ai conçu la structure de la classe Nouveau_Plateau, en déterminant les attributs et les méthodes nécessaires pour représenter le plateau de jeu d'échecs. J'ai fais la méthode `creer_Plateau` pour initialiser le plateau de jeu avec les pièces d'échecs correctement positionnées au début de la partie. J'ai aussi pris en charge la méthode `deplacer`, qui permet le déplacement des pièces sur le plateau. J'ai implémenté la méthode `dessiner_plateau` pour dessiner le plateau de jeu et les cases sur l'interface utilisateur à l'aide de Pygame. J'ai défini les couleurs et les dimensions des cases pour créer une représentation visuelle du plateau. J'ai également implémenté des méthodes pour obtenir des informations sur les pièces sur le plateau, telles que la méthode `obtenir_piece`, qui renvoie la pièce située à une position donnée sur le plateau.Et j’ai aussi développé l’autre partie de la class jeu où select est appelée lorsque le joueur sélectionne une case sur le plateau, elle gère la logique de sélection des pièces et des déplacements valides, dessiner_deplacements_valides dessine

visuellement les déplacements valides pour la pièce sélectionnée. Ensuite la fonction position prend les coordonnées d'un clic de souris et retourne la ligne et la colonne correspondantes sur le plateau de jeu. Enfin, la condition if __name__ == "__main__" vérifie si le fichier est exécuté directement comme un script Python, et appelle la fonction principal.

Partie réseaux:

Nous avons une feuille PHP et HTML qui est une page web qui affiche les informations d'un joueur ainsi que les détails de ses parties.

Lien vers une feuille de style CSS : `<link href="css/playerStyle.css" rel="stylesheet" type="text/css">` : Cette ligne lie la page HTML à une feuille de style CSS externe, permettant de styliser la mise en page et l'apparence des éléments HTML.

Connexion à la base de données MySQL :
```php
$user = 'root';
$password = '';
$connection = new PDO('mysql:host=localhost;dbname=AppJeu', $user, $password);
```
Ces lignes établissent une connexion à une base de données MySQL locale appelée "AppJeu" en utilisant le nom d'utilisateur "root" et un mot de passe vide.

Récupération des données du joueur :
```php
$playerId = $_GET['id'];
$query = $connection->prepare('SELECT * FROM player WHERE id="' . $playerId . '"');
$query->execute();
$player = $query->fetch();
```
Ces lignes récupèrent les informations du joueur spécifié par son identifiant (passé via l'URL) à partir de la table "player" dans la base de données.

Récupération des parties du joueur :
```php
$parties = $connection->query('SELECT * FROM party WHERE player1_id="' . $playerId. '"OR player2_id ="'. $playerId . '"');
```
Cette requête SQL récupère toutes les parties dans lesquelles le joueur est impliqué, soit en tant que joueur 1 (player1_id), soit en tant que joueur 2 (player2_id).

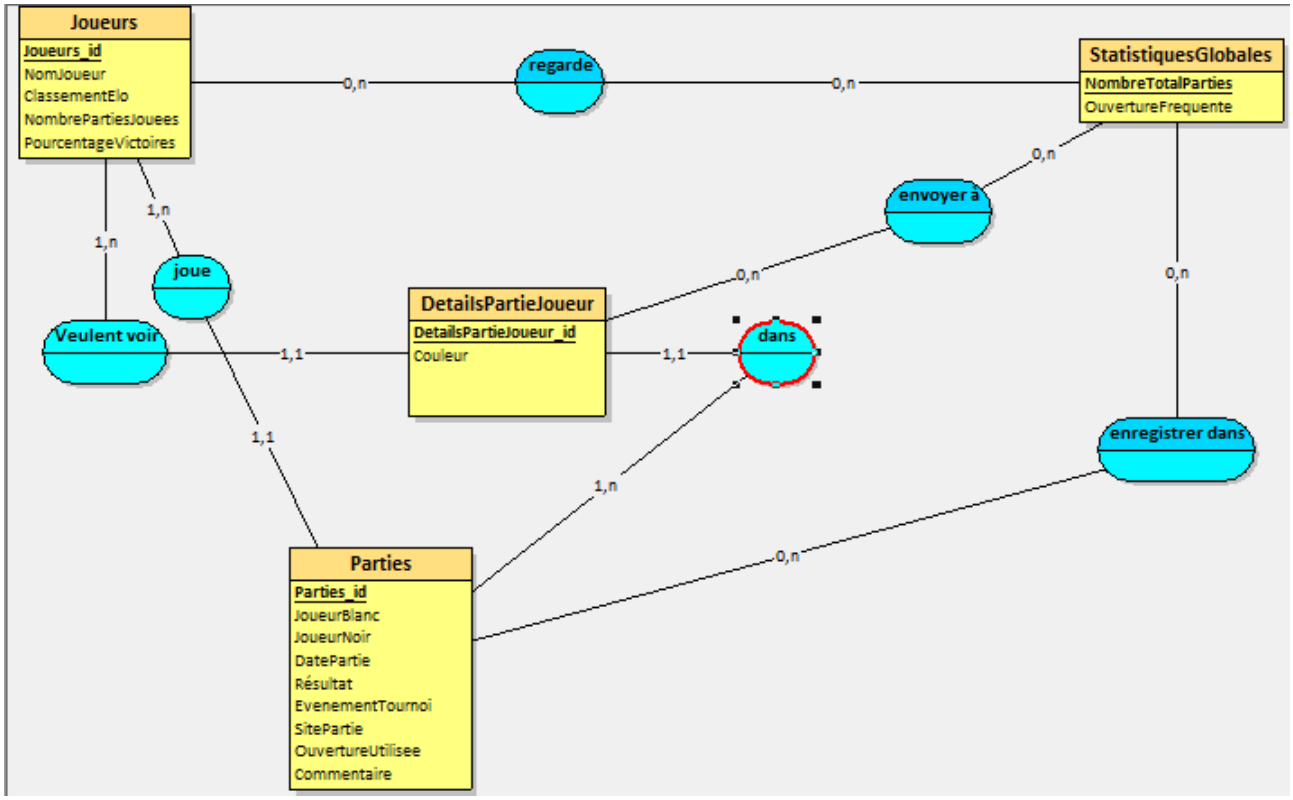
Fonction de conversion du temps :
```php
function convert_temps($temps){
    // Logique de conversion du temps en heures, minutes et secondes
}
```
Cette fonction prend un temps en secondes en entrée et le convertit en heures, minutes et secondes selon la logique définie à l'intérieur de la fonction.

Affichage des informations du joueur et des détails des parties :
Les informations du joueur (ID, prénom, nom, téléphone, email, etc.) ainsi que les détails des parties (numéro, adversaire, gagnant/perdant, niveau, durée, date, etc.) sont affichées dans des balises HTML. Les informations sont récupérées à partir des données récupérées de la base de données MySQL.

Requêtes pour récupérer les statistiques des parties et les informations de temps :
Il y a plusieurs requêtes similaires utilisées pour récupérer les statistiques des parties du joueur, telles que le nombre de parties gagnées, perdues, en cours, en recherche d'adversaire, ainsi que le temps global de jeu et le temps moyen par partie. Ces données sont ensuite affichées dans la page HTML après avoir été traitées par la fonction de conversion du temps.

Fermeture de la connexion à la base de données :
```php
$connection = null;
```
Cette ligne ferme la connexion à la base de données une fois que toutes les opérations de récupération des données sont terminées.

SQL :



Description de la class Pièce et Jeu D'Arnaud:

```
class Piece:
    def __init__(self, Carré, couleur, type_piece, ligne, colonne):
        self.Carré = Carré
        self.couleur = couleur
        self.ligne = ligne
        self.colonne = colonne
        self.type_piece = type_piece
        self.x = colonne * Carré
        self.y = ligne * Carré
        self.coups_possibles = []

    def deplacer_piece(self, ligne, colonne):
        self.ligne = ligne
        self.colonne = colonne
        self.x = colonne * self.Carré
        self.y = ligne * self.Carré

    def vider_coups_possibles(self):
        self.coups_possibles.clear()
```

La classe Piece est définie pour représenter une pièce individuelle dans un jeu d'échecs.

1. Init
   - Lorsqu'une nouvelle instance de Piece est créée, des attributs tels que Carré, couleur, type_piece, ligne, colonne sont définis.
   - self.Carré représente la taille d'un carré sur le plateau de jeu.
   - self.couleur représente la couleur de la pièce (blanc ou noir).
   - self.type_piece indique le type de la pièce (roi, dame, tour, fou, cavalier, pion).
   - self.ligne et self.colonne définissent la position initiale de la pièce sur le plateau.
   - self.x et self.y calculent les coordonnées graphiques de la pièce sur l'écran en fonction de sa position sur le plateau.
   - self.coups_possibles est une liste initialement vide pour stocker les déplacements possibles de la pièce.

Méthode deplacer_piece:

- La méthode deplacer_piece permet de déplacer la pièce vers une nouvelle position spécifiée par ligne et colonne.
- Elle met à jour les attributs self.ligne et self.colonne avec les nouvelles valeurs.
- Ensuite, les coordonnées graphiques self.x et self.y sont recalculées en fonction de la nouvelle position sur le plateau.

Méthode vider_coups_possibles:

- La méthode vider_coups_possibles est utilisée pour réinitialiser la liste self.coups_possibles en la vidant, c'est-à-dire en supprimant tous ses éléments.

Class pion :
1ere partie :

```
class Pion(Piece):
```

```
    def __init__(self, Carré, couleur, type_piece, ligne, colonne):
        super().__init__(Carré, couleur, type_piece, ligne, colonne)
        self.premier_deplacement = True
```

super().__init__ :
Dans le constructeur de la classe Pion, en appelant super().init(), on initialise les attributs de base de la classe parente Piece avant d'ajouter les caractéristiques spécifiques au pion.

Carré: représente la taille d'un carré sur le plateau de jeu.

- couleur: La couleur du pion.

type_piece: Le type de la pièce, qui pourrait être "pion" dans ce cas.
ligne et colonne: Les coordonnées de la position initiale du pion sur le plateau.
premier_deplacement: Cet attribut est initialisé à True indique si le pion a déjà effectué son premier déplacement. Pour déterminer si le pion peut avancer de deux cases lors de son premier mouvement.
2eme partie :

```
 def obtenir_coups_possibles(self, ligne, colonne, Plateau):
        self.vider_coups_possibles()
        if self.couleur == Blanc:
            direction = -1
        else :
            direction = 1
        lignes = [ligne + direction]
        if self.premier_deplacement:
```

lignes.append(ligne + 2 * direction) si c'est le premier déplacement d'un pion alors celui pourras se déplacer de deux case au lieu d'une

```
  def obtenir_coups_possibles(self, ligne, colonne, Plateau):
        self.vider_coups_possibles() :
```

Cette fonction dans la classe Pion réinitialise simplement la liste des coups possibles du pion à chaque appel, afin de préparer le terrain pour le calcul de nouveaux coups possibles en fonction de sa position actuelle sur le plateau de jeu.

```
if self.couleur == Blanc:
            direction = -1
        else :
            direction = 1 :
```

suivant la couleur du joueur, définie la direction dans laquelle iront les pions

lignes = [ligne + direction] :

elle ajuste la position verticale ou horizontale d'une pièce sur un l’échiquier .

```
f self.premier_deplacement:
        lignes.append(ligne + 2 * direction) :
```

si c'est le premier déplacement d'un pion alors celui pourras se déplacer de deux case au lieu d'une 3eme partie :

```
  for l in lignes:
        if 0 <= l < len(Plateau) and Plateau[l][colonne] == 0:
            self.coups_possibles.append((l, colonne))
```

Dans cette boucle, chaque valeur de la liste "lignes" est vérifiée pour s'assurer qu'elle est dans les limites du plateau de jeu (représenté par "Plateau") et que la case correspondante sur le plateau est vide (représentée par la valeur 0). Si ces conditions sont remplies, la position est ajoutée à la liste "self.coups_possibles" des coups possibles pour le pion.
Le .append = ajoute une nouvelle position valide à la liste des coups possibles du pion.

La lettre l représente les lignes et len(plateau) la longueur de la liste plateau

donc si la case correspondante et vide, la position est ajoutée à la liste des mouvements possibles du pion.

4 eme partie :

```
 for compensation in (-1, 1):
        colonne_suivante = colonne + compensation
        if 0 <= colonne_suivante < len(Plateau[0]):
            piece = Plateau[ligne + direction][colonne_suivante]
            if piece and piece.couleur != self.couleur:
                self.coups_possibles.append((ligne + direction, colonne_suivante))

      return self.coups_possibles
```

for compensation in (-1, 1): les valeurs -1 et 1 dans cette boucle correspondent aux directions dans lesquelles le pion peut se déplacer horizontalement sur l'échiquier.
Lorsque la valeur de compensation est -1, cela signifie que le pion peut se déplacer d'une case vers la gauche sur l'échiquier (reculer pour les Blancs, avancer pour les Noirs).
Lorsque la valeur de compensation est 1, cela signifie que le pion peut se déplacer d'une case vers la droite sur l'échiquier (avancer pour les Blancs, reculer pour les Noirs). Tous dépend de la couleur que l’on joue et de la facon de se placé .

colonne_suivante = colonne + compensation : Cela signifie que pour chaque direction, la nouvelle colonne est calculée en ajoutant la valeur de compensation à la colonne actuelle du pion.

colonne_suivante < len(Plateau[0]) :On vérifie si la nouvelle colonne calculée est à l'intérieur des limites horizontales du plateau de jeu.

piece = Plateau[ligne + direction][colonne_suivante] : On récupère la pièce sur la case adjacente à la position actuelle du pion.

if piece and piece.couleur != self.couleur: Si une pièce existe sur cette case et si elle est d'une couleur différente de celle du pion actuel.

self.coups_possibles.append((ligne + direction, colonne_suivante)) : Cette ligne ajoute la position de la case adjacente où se trouve une pièce adverse à la liste des coups possibles du pion, si cette condition est vérifiée.
return self.coups_possibles : La méthode retourne la liste mise à jour des coups possibles du pion.

Description des class Nouveau_Plateau et Jeu d'Anthony :

```
class Nouveau_Plateau:
    def __init__(self, Largeur, Hauteur, Lignes, Colonnes, Case, Victoire):
        self.Largeur = Largeur
        self.Hauteur = Hauteur
        self.Case = Case
        self.Plateau_Jeu = self.Largeur // 2
        self.Victoire = Victoire
        self.Lignes = Lignes
        self.Colonnes = Colonnes
        self.Plateau = []
        self.creer_Plateau()
```

Initialisation (`__init__`):

- Lorsqu'une nouvelle instance de `Nouveau_Plateau` est créée, les attributs tels que `Largeur`, `Hauteur`, `Lignes`, `Colonnes`, `Case`, `Victoire` sont définis.
- `self.Largeur` et `self.Hauteur` représentent respectivement la largeur et la hauteur du plateau.
- `self.Lignes` et `self.Colonnes` représentent respectivement le nombre de lignes et de colonnes du plateau.
- `self.Case` représente la taille d'une case sur le plateau.
- `self.Plateau_Jeu` est calculé comme la moitié de la largeur du plateau, potentiellement utilisé pour des calculs spécifiques.
- `self.Victoire` est un attribut qui semble être lié aux conditions de victoire du jeu.
- `self.Plateau` est une liste vide qui sera utilisée pour stocker les informations sur les cases du plateau.
- La méthode `creer_Plateau()` est appelée à la fin de l'initialisation pour initialiser le plateau de jeu.

```
def creer_Plateau(self):
    for ligne in range(self.Lignes):
        self.Plateau.append([0 for _ in range(self.Colonnes)])
        for colonne in range(self.Colonnes):
            if ligne == 1 or ligne == 6:
                couleur = Pion_Noir if ligne == 1 else Pion_Blanc
                couleur_piece = Noir if ligne == 1 else Blanc
                nom_piece = "Pion"
```

```
            self.Plateau[ligne][colonne] = couleur(self.Case, couleur_piece, nom_piece, ligne,
colonne)
        elif ligne == 0 or ligne == 7:
            if colonne == 0 or colonne == 7:
                classe_piece = Tour
            elif colonne == 1 or colonne == 6:
                classe_piece = Cavalier
            elif colonne == 2 or colonne == 5:
                classe_piece = Fou
            elif colonne == 3:
                classe_piece = Reine
            elif colonne == 4:
                classe_piece = Roi
            couleur = Tour_Noir if ligne == 0 else Tour_Blanc
            couleur_piece = Noir if ligne == 0 else Blanc
            nom_piece = classe_piece.__name__
            self.Plateau[ligne][colonne] = classe_piece(self.Case, couleur_piece, nom_piece, ligne,
colonne)
```

Elle itère à travers chaque ligne et colonne du plateau à l'aide de deux boucles `for`. Pour chaque case du plateau, elle examine la position de la ligne et de la colonne pour déterminer quelle pièce doit être placée à cet emplacement. Si la ligne est 1 ou 6, cela signifie qu'il faut placer des pions (noirs ou blancs) sur cette rangée. Si la ligne est 0 ou 7, cela signifie qu'il faut placer les autres pièces (tour, cavalier, fou, reine, roi) sur cette rangée. Les types de pièces à placer dépendent de la colonne dans laquelle la pièce doit être placée. Par exemple, si la colonne est 0 ou 7, il faut placer une tour. Si la colonne est 1 ou 6, il faut placer un cavalier, et ainsi de suite. En fonction de la position de la pièce sur le plateau, la méthode crée une instance de la classe correspondante (Pion, Tour, Cavalier, Fou, Reine ou Roi) avec les attributs appropriés comme la taille de la case, la couleur de la pièce, le nom de la pièce et les coordonnées de la pièce sur le plateau, et l'assigne à la position correspondante dans la liste `Plateau`.

```
def obtenir_piece(self, ligne, colonne):
    return self.Plateau[ligne][colonne]
```

La méthode `obtenir_piece(self, ligne, colonne)` permet d'obtenir la pièce située à une position spécifique sur le plateau en fournissant les coordonnées de la ligne et de la colonne. Elle renvoie la pièce présente à cet emplacement.

```
def deplacer(self, piece, ligne, colonne):

    self.Plateau[piece.ligne][piece.colonne], self.Plateau[ligne][colonne] = \

        self.Plateau[ligne][colonne], self.Plateau[piece.ligne][piece.colonne]

    piece.deplacer_piece(ligne, colonne)
```

La méthode `deplacer(self, piece, ligne, colonne)` est utilisée pour déplacer une pièce à une nouvelle position spécifiée par les coordonnées de ligne et de colonne. Elle effectue le déplacement de la pièce en mettant à jour les informations de la liste `Plateau`. Elle utilise la méthode `deplacer_piece()` de l'objet `piece` pour mettre à jour les attributs de position de la pièce.

```
def capturer_piece(self, ligne, colonne):

    piece_capturee = self.Plateau[ligne][colonne]

    if piece_capturee:
```

```
        self.Plateau[ligne][colonne] = 0
        del piece_capturee
```

La méthode `capturer_piece(self, ligne, colonne)` est utilisée pour capturer une pièce située à une position spécifique sur le plateau. Si une pièce est présente à cet emplacement, elle est capturée et retirée du plateau. Cela est réalisé en mettant à zéro la case correspondante dans la liste `Plateau` et en supprimant l'objet de pièce capturée.

```
def enlever_piece(self, x, y, num, pieces):
    # enleve une piece de l'echiquier
    for i in range(len(pieces)):
        if pieces[i].x == x and pieces[i].y == y and i != num:
            return i
    # prise en passant
    if pieces[num].nom == "Pion" and pieces[num].couleur == "blanc":
        for i in range(len(pieces)):
            if pieces[i].x == x and pieces[i].y == y + c:
                return i
    if pieces[num].nom == "Pion" and pieces[num].couleur == "noir":
        for i in range(len(pieces)):
            if pieces[i].x == x and pieces[i].y == y - c:
                return i
    return -1
```

Cette méthode `enlever_piece(self, x, y, num, pieces)` est utilisée pour enlever une pièce du plateau d'échecs. Elle prend en paramètre les coordonnées `x` et `y` de la position de la pièce à enlever, l'indice `num` de la pièce à enlever dans la liste `pieces`, et la liste `pieces` contenant toutes les pièces.Elle parcourt la liste `pieces` pour rechercher une pièce autre que celle spécifiée par l'indice `num` qui se trouve aux coordonnées `x` et `y`. Si une telle pièce est trouvée, elle retourne son indice dans la liste `pieces`. En cas de prise en passant, si la pièce spécifiée par `num` est un pion et qu'elle est blanche, elle recherche une pièce noire à une position adjacente appropriée pour la prise en passant. De même, si le pion est noir, elle recherche une pièce blanche pour la prise en passant. Si une pièce appropriée est trouvée, elle retourne son indice dans la liste `pieces`. Si aucune pièce à enlever n'est trouvée, elle retourne -1 pour indiquer qu'aucune pièce n'a été enlevée.

```
def dessiner_pieces(self):
    for r in range(self.Lignes):
        for c in range(self.Colonnes):
            piece = self.Plateau[r][c]
            if piece:
```

piece.dessiner_piece(self.Victoire)

La méthode `dessiner_pieces(self)` est utilisée pour dessiner toutes les pièces présentes sur le plateau. Elle parcourt toutes les cases du plateau et récupère la pièce présente à chaque position. Si une pièce est présente à cette position (c'est-à-dire si `piece` n'est pas None), elle appelle la méthode `dessiner_piece()` de cette pièce en lui passant la surface `self.Victoire` sur laquelle la pièce doit être dessinée.

```
def dessiner_plateau(self):
    self.Victoire.fill(MarronClair)
    for r in range(self.Lignes):
        for c in range(r % 2, self.Colonnes, 2):
            couleur_case = Marron if (r + c) % 2 == 0 else MarronClair
            pygame.draw.rect(self.Victoire, couleur_case, (c * self.Case, r * self.Case, self.Case, self.Case))
```

La méthode `dessiner_plateau(self)` est utilisée pour dessiner le plateau de jeu lui-même. Elle remplit d'abord la surface `self.Victoire` avec une couleur claire (probablement une couleur de fond). Ensuite, elle parcourt toutes les lignes et colonnes du plateau en alternant les couleurs des cases pour créer un motif de damier. Elle utilise la fonction `pygame.draw.rect()` pour dessiner chaque case du plateau avec la couleur appropriée, en fonction de sa position dans le damier.