

# ChessPy

Lausanne, du 12 mai au 6 juin 2023

Mark Lovink

Fin 2

# Remerciements

*Je tiens à remercier Monsieur Helder Costa Lopes pour avoir pris de son temps pour répondre à mes questions.*

*Ainsi que Monsieur William Lovink pour avoir corrigé mon TPI.*

*Je veux aussi remercier tous les bêta-testeurs :*

*Monsieur Ken Lam,*

*Monsieur Marc Vô,*

*Monsieur Thilo Paetzel,*

*Monsieur Damian Marcinkiewicz ,*

*pour leurs avis sincères et leurs suggestions.*

Travail suivi par	
Helder Costa Lopes	Chef de projet
Oberson Bernard	Expert 1
Roger Malherbe	Expert 2

# Table des matières

1	Cahier des charges .....	4
2	Analyse préliminaire .....	6
2.1	Objectifs.....	6
2.2	Les points qui seront évalués .....	6
3	Planification initiale .....	7
4	Analyse / Conception.....	8
4.1	Opportunité.....	8
4.2	Méthodologie de travail.....	8
4.3	Les limites.....	8
4.4	Méthode d'implémentation.....	9
4.5	Schéma UML.....	10
4.6	Maquette de l'application .....	11
4.7	Environnement Python .....	14
4.8	Répertoire Git .....	14
4.9	Bibliothèque .....	15
4.10	Technologies utilisées .....	15
4.11	Stratégie de test.....	15
4.12	Risques techniques .....	16
5	Réalisation.....	17
5.1	Répertoire de dossier .....	17
5.2	Boucle principale .....	18
5.3	Affichage du plateau d'échec.....	19
5.4	Affichage des pions et pièces .....	23
5.5	Mouvement.....	28
5.6	Coups légaux.....	31
5.7	Échec.....	40
5.8	Échec et mat.....	48
5.9	Sauvegarde de parties.....	51
5.10	Charger une partie sauvegardée .....	52
5.11	Affichage du dernier mouvement.....	53
5.12	Feedback utilisateur.....	56
5.13	Tests unitaires et performances.....	63
5.14	Résultats des tests effectués .....	66
5.15	Liste des documents fournis .....	68
6	Conclusions .....	68
6.1	Bilan de la planification .....	68
6.2	Bilan des objectifs.....	69
6.3	Bilan personnel.....	69
7	Annexes.....	70
7.1	Résumé du rapport du TPI / version succincte de la documentation .....	70
7.2	Glossaire .....	71
7.3	Sources – Webographie .....	72
7.4	Planification initiale .....	73
7.5	Journal de travail .....	83
7.6	Table des illustrations .....	97
7.7	Code source .....	100

# **1 Cahier des charges**

## **1.1 TITRE**

### ChessPy :

JEU D'ECHECS EN PYTHON AVEC PYGAME

## **1.2 MATÉRIEL ET LOGICIEL À DISPOSITION**

- Un ordinateur avec un processeur rapide et suffisamment de RAM pour faire fonctionner les outils de développement de manière fluide.
- Une installation de Python 3.x avec les packages nécessaires pour le développement de jeux avec Pygame.
- L'IDE PyCharm Community Edition ou un autre environnement de développement intégré de Python (VSCode).
- Les ressources graphiques nécessaires pour la création de l'interface utilisateur graphique, telles que des images pour les pièces d'échecs et le plateau de jeu.
- Des spécifications détaillées pour le développement du jeu, y compris les exigences fonctionnelles et non fonctionnelles, ainsi que les critères d'acceptation.
- Conformité ETML : Respect des normes de codage de l'école.

## **1.3 PRÉREQUIS**

- Bonne connaissance du langage de programmation Python, en particulier des concepts de programmation orientée objet.
- Bonne connaissance de la bibliothèque Pygame pour la création d'interfaces graphiques et d'applications de jeu.
- Connaissance des règles officielles du jeu d'échecs de la FIDE.
- Expérience préalable dans le développement de projets utilisant Python et Pygame.
- Le candidat doit également être capable de travailler de manière autonome, de résoudre des problèmes techniques et de communiquer efficacement avec le chef de projet.
- Des compétences en conception de jeu et en optimisation de la performance seront également utiles.
- Il convient de noter que la réalisation préalable par le candidat d'un ou deux projets basés sur les technologies utilisées est fortement recommandée pour garantir l'efficacité et la qualité du développement de ce projet.

## 1.4 DESCRIPTIF DU PROJET

PyChess est un jeu d'échecs en Python utilisant la bibliothèque Pygame pour créer une interface utilisateur graphique. Les fonctionnalités du jeu incluent la possibilité de jouer (contre l'ordinateur) ou contre un autre joueur, la sélection de différents niveaux de difficulté pour l'ordinateur, la sauvegarde et le chargement de parties, ainsi que la visualisation de l'historique des mouvements et des coups légaux.

### 1.4.1 Fonctionnalités principales :

- Interface utilisateur graphique intuitive pour jouer aux échecs.
- Option de jeu contre l'ordinateur **ou** un autre joueur **en mode local**.
- Choix de différents niveaux de difficulté pour l'adversaire informatique.
- Fonctionnalités de sauvegarde et de chargement des parties en cours.
- Consultation de l'historique des mouvements et des coups légaux.
- Mise en évidence des mouvements possibles pour chaque pièce sélectionnée.
- Affichage clair et détaillé des mouvements effectués au cours d'une partie.

### 1.4.2 Collecte et gestion du feedback des utilisateurs :

- Prévoir une date dans le planning initial pour recueillir les retours des utilisateurs, qui serviront à évaluer et améliorer les fonctionnalités du jeu. Les méthodes de collecte du feedback et l'intégration des données recueillies dans les mises à jour ultérieures seront déterminées.

### 1.4.3 Tests sur le système Windows :

- Mettre en place une série de tests unitaires, d'intégration et de performance pour vérifier la qualité et la fiabilité du jeu sur le système Windows, assurant ainsi qu'il répond aux normes de qualité attendues.

### 1.4.4 Utilisation de Git :

- Le candidat doit utiliser Git pour gérer les versions du projet et partager le dépôt distant avec le chef de projet et les experts. En créant des branches, effectuant des commits fréquents et résolvant les conflits, le candidat facilitera la collaboration et assurera un suivi efficace des modifications apportées au code.

## 2 Analyse préliminaire

Dans le cadre d'un Travail Pratique Individuel, TPI, de l'ETML. Le but est de créer un jeu d'échecs, respectant les règles de la FIDE\*. Le jeu sera réalisé en utilisant le langage Python et sa bibliothèque Pygame.

### 2.1 Objectifs

- Réaliser et afficher un plateau d'échiquier.
- Réaliser et afficher les pièces et pions sur le plateau.
- Réaliser et afficher les mouvements de chaque pièce et pion.
- Afficher les mouvements réalisés durant une partie.
- Consulter les mouvements réalisés durant une partie.
- Pouvoir sauvegarder une partie en cours.
- Pouvoir continuer la partie sauvegardée.
- Tester les performances du programme.
- Réaliser un formulaire pour des bêta-testeurs\*.
- Faire une gestion des versions à l'aide de Git.
- Faire un suivi de l'avancement du projet

### 2.2 Les points qui seront évalués

- La qualité du code, pas de répétition ou de fonction inutile.
- Utilisation de branches pour les fonctionnalités et les correctifs, commits\* fréquents (par exemple 2 par semaine), gestion efficace des conflits.
- Respect des normes de codage de l'école.
- Le déplacement des pions et des pièces avec leurs mouvements légaux respectifs, en respectant les règles définies par la FIDE.
- Évaluation de la conception et de l'ergonomie de l'interface utilisateur
- Le feedback des utilisateurs (échelle de satisfaction, commentaires qualitatifs)
- Évaluation de la fluidité et de la réactivité du jeu
- Le candidat doit effectuer des mises à jour quotidiennes d'un journal de travail

---

\*Voir Glossaire



### 3 Planification initiale

Présentation de la planification initiale sous forme graphique.

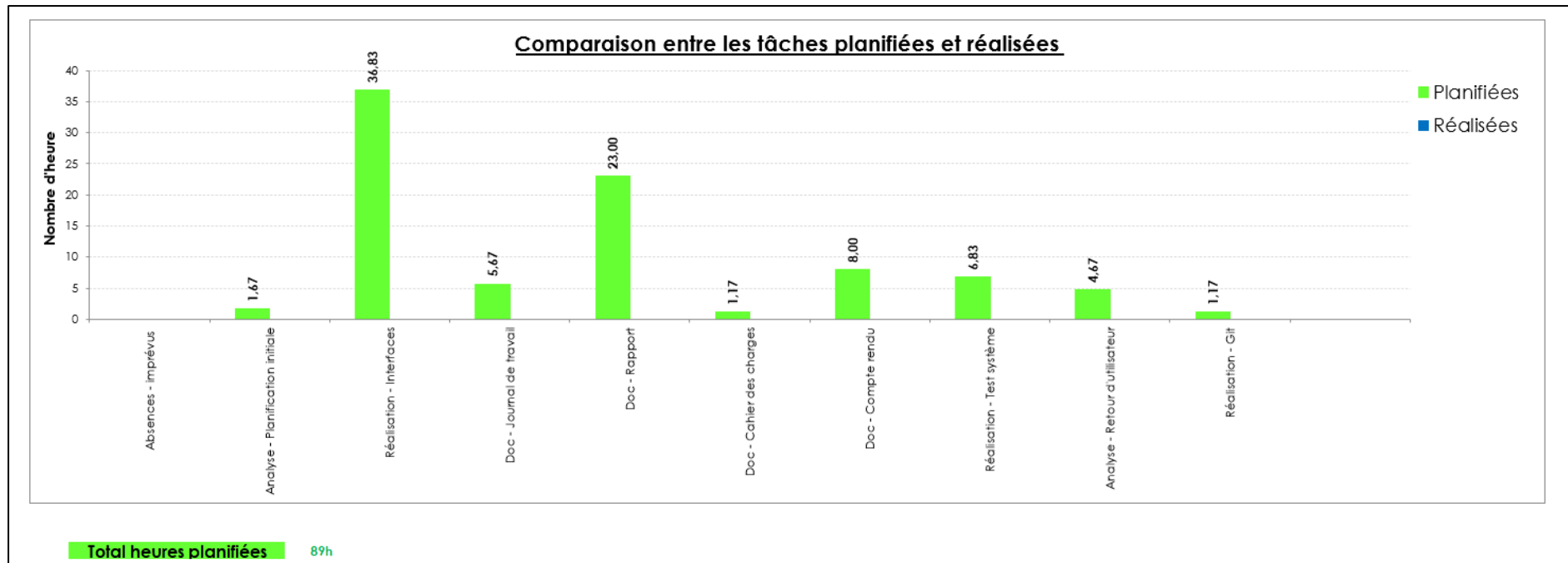


Figure 1 Planification initiale

## 4 Analyse / Conception

### 4.1 Opportunité

Ce projet me permet de me perfectionner sur l'utilisation de Python ainsi que l'une de ses bibliothèques Pygame mais également de pouvoir de me m'améliorer avec des outils de gestion de projet tels que Git et pouvoir gérer un projet dans un dépôt GitHub. Ce projet me permet d'avoir une vue d'ensemble sur la charge de travail à effectuer et sur la gestion du temps à disposition.

### 4.2 Méthodologie de travail

Pour la réalisation de ce projet j'ai choisi la méthode des 6 pas. Cela consiste à découper le projet ou une fonctionnalité en 6 étapes comme le graphique ci-dessous.



Cette méthode débute avec une analyse détaillée du cahier des charges ou les objectifs, les attentes et les exigences du projet sont spécifiées.

Une fois que les points essentiels ont été relevés, ils permettent de se concentrer sur le design de la maquette de l'application ainsi que la conception d'un graphique UML. Les maquettes permettant de faire une décision sur le produit final livrable à la fin du projet, ainsi la réalisation de l'application peut être débutée avec les réalisations de différentes fonctionnalités.

C'est-à-dire que lors de la mise en place de ces fonctionnalités, la solution est développée puis testée avec les méthodes de test prédéfinies.

Une fois ces tests finis et approuvés, ils peuvent être déployés.

Cette méthode permet de se concentrer sur un objectif à la fois. En contrepartie le danger de cette méthode c'est la gestion et la non-flexibilité du temps.

### 4.3 Les limites

Une fois le projet terminé, le programme permet à deux utilisateurs de jouer sur le même ordinateur, en local. Il a été développé dans le but du perfectionnement de connaissances. Dans ces circonstances, le jeu a été conçu pour des parties amicales, plutôt que pour des compétitions, et il lui manque certaines fonctionnalités essentielles présentes dans de vrais jeux.



#### 4.4 Méthode d'implémentation

La réalisation de l'application se base sur les 5 principes SOLID.

SOLID est un acronyme qui représente cinq principes de conceptions largement utilisées en programmation orientée objet pour créer des systèmes logiciels plus maintenables et flexibles

Les principes SOLID sont les suivants :

Principe de responsabilité unique (Single Responsibility Principle - SRP) : Une classe ne doit avoir qu'une seule raison de changer. Cela signifie qu'une classe ne devrait être responsable que d'une seule tâche ou d'une seule responsabilité. En maintenant la responsabilité d'une classe limitée, on facilite les modifications futures et on évite les effets de bord.

Principe d'ouverture/fermeture (Open/Closed Principle - OCP) : Les entités logicielles (classes, modules, fonctions, etc.) doivent être ouvertes à l'extension, mais fermées à la modification. Cela signifie qu'il est possible d'ajouter de nouvelles fonctionnalités en étendant le code existant plutôt qu'en le modifiant directement. Cela favorise un système plus évolutif et facilite la réutilisation du code existant.

Principe de substitution de Liskov (Liskov Substitution Principle - LSP) : Les objets d'une classe dérivée doivent pouvoir être substitués à ceux d'une classe de base sans altérer la cohérence du programme. Cela signifie que les sous-classes doivent respecter le contrat défini par la classe de base et ne pas introduire de comportement imprévu ou indésirable.

Principe de ségrégation des interfaces (Interface Segregation Principle - ISP) : Les interfaces spécifiques aux clients sont préférées aux interfaces générales. Plutôt que de créer des interfaces volumineuses et complexes qui sont utilisées par différents clients, il est préférable de créer des interfaces plus spécialisées qui répondent aux besoins spécifiques de chaque client. Cela évite aux clients de dépendre de fonctionnalités inutiles et réduit le couplage entre les classes.

Principe d'inversion de dépendance (Dependency Inversion Principle - DIP) : Les modules de haut niveau ne doivent pas dépendre des modules de bas niveau. Les deux doivent plutôt dépendre d'abstractions. Ce principe favorise une meilleure modularité et une plus grande flexibilité en inversant les dépendances entre les modules, ce qui facilite le remplacement de dépendances et permet de réaliser des tests unitaires plus facilement.\*

---

\*Source [https://fr.wikipedia.org/wiki/SOLID\\_\(informatique\)](https://fr.wikipedia.org/wiki/SOLID_(informatique)) "SOLID (informatique)"

## 4.5 Schéma UML

Ce schéma contient toutes les données nécessaires dans la réalisation du projet selon le cahier des charges.

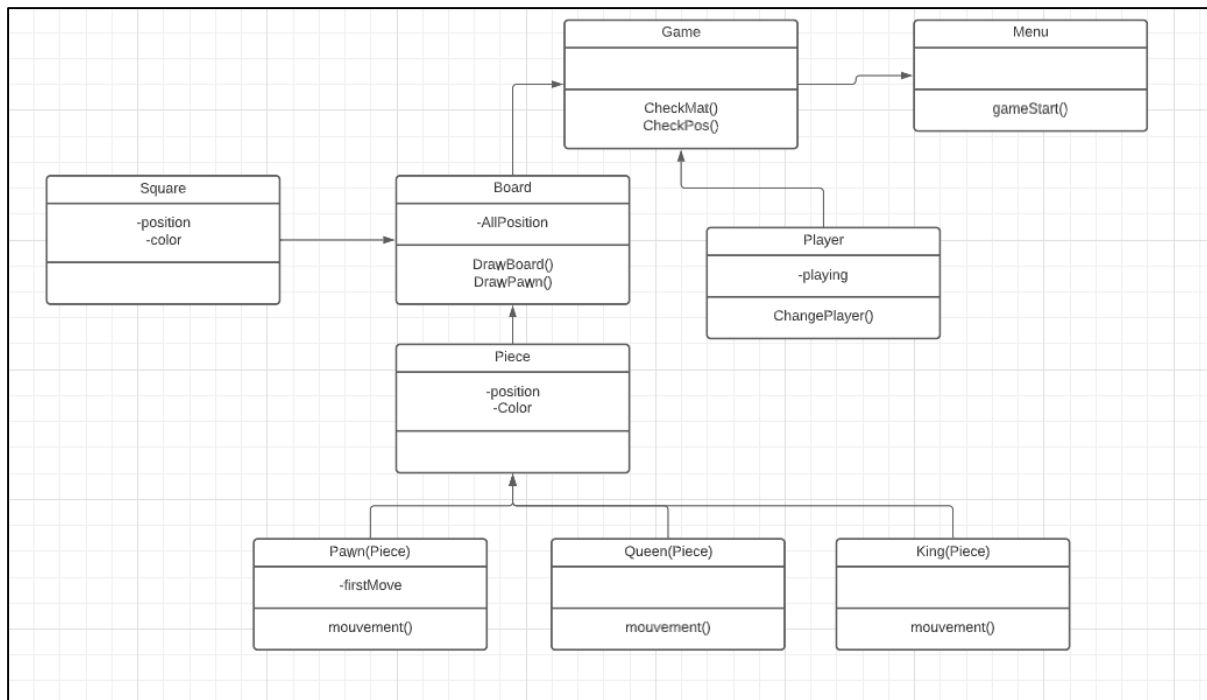


Figure 2 Schéma UML version 1

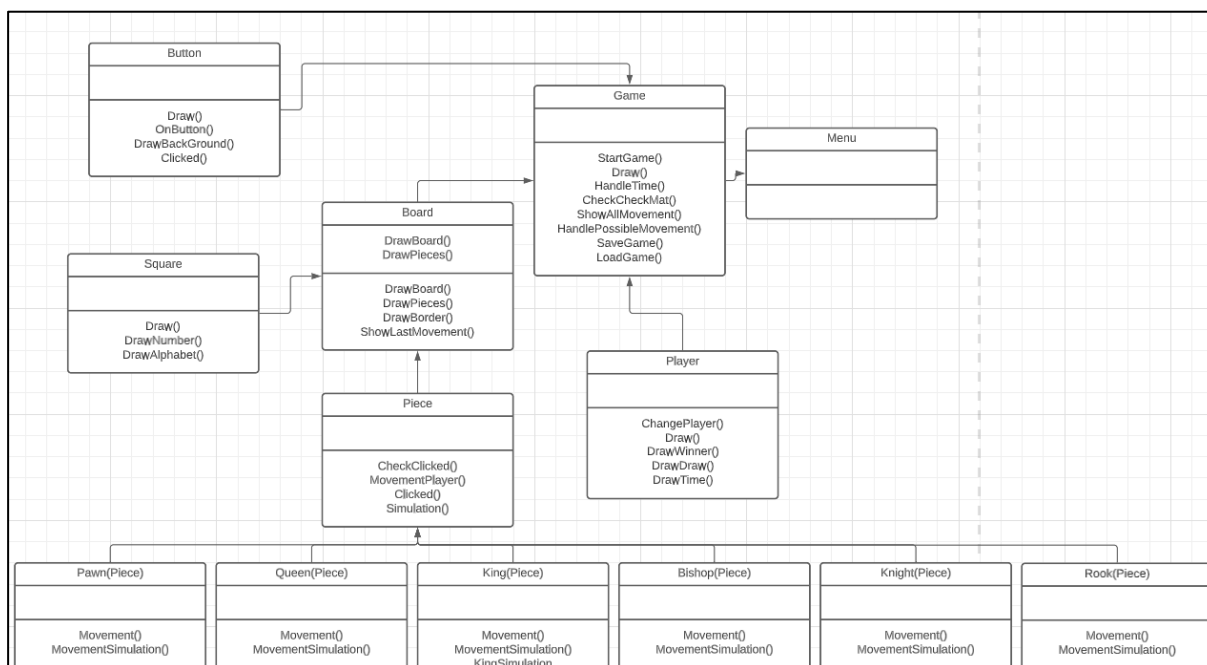


Figure 3 Schéma UML version final

#### 4.6 Maquette de l'application

Les images présentées ci-dessous illustrent la conception du programme «ChessPy» grâce aux maquettes Figma.

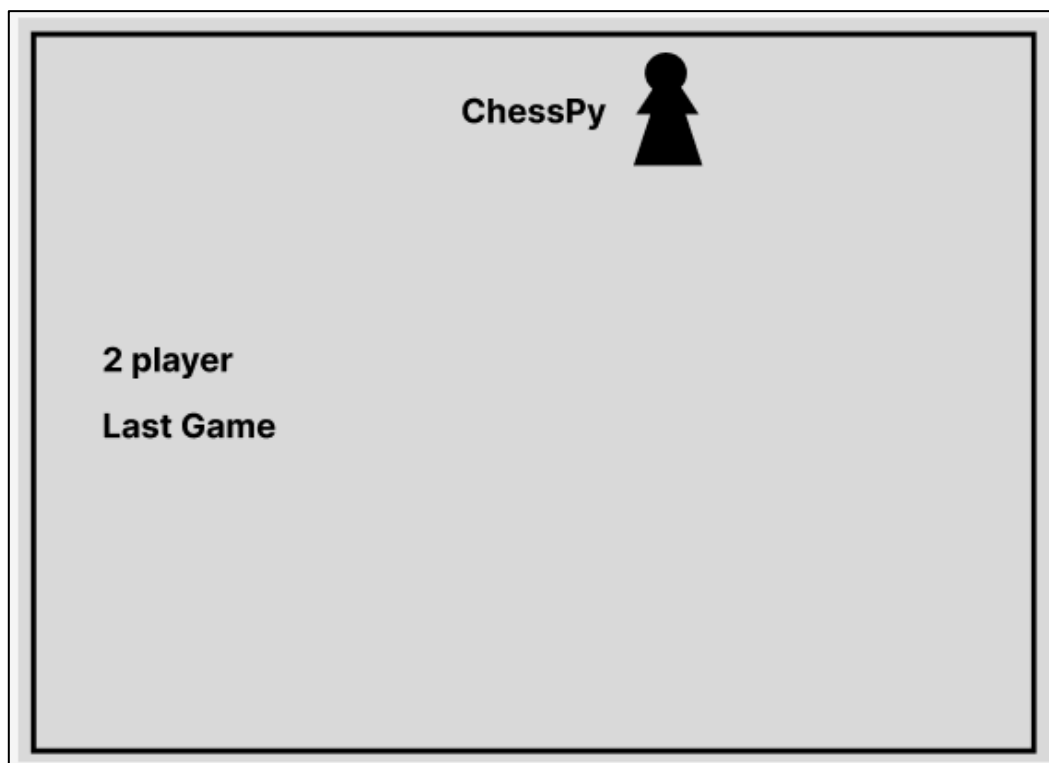


Figure 4 Figma page d'accueil

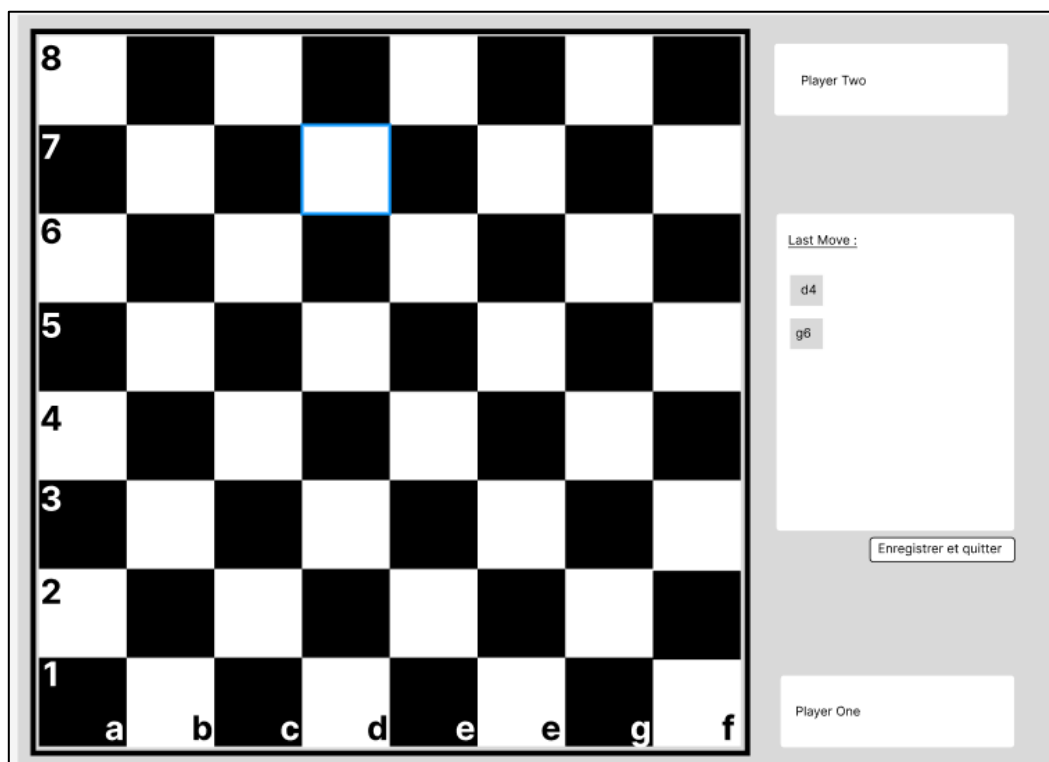


Figure 5 Figma plateau vierge

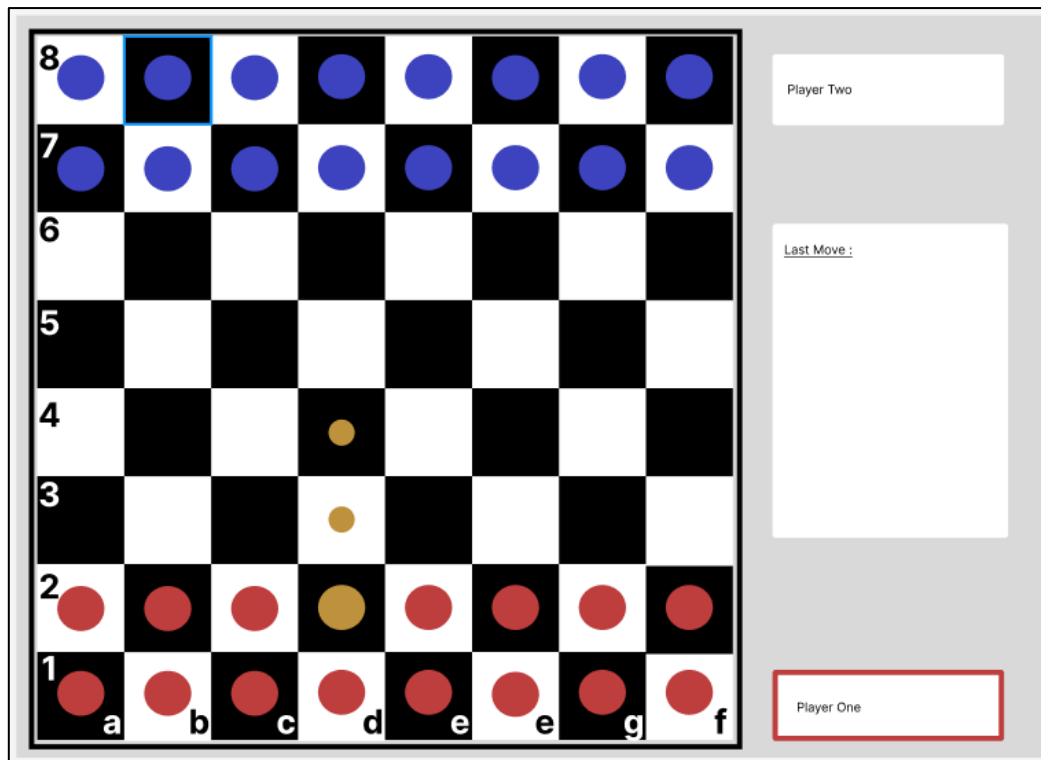


Figure 6 Figma joueur 1

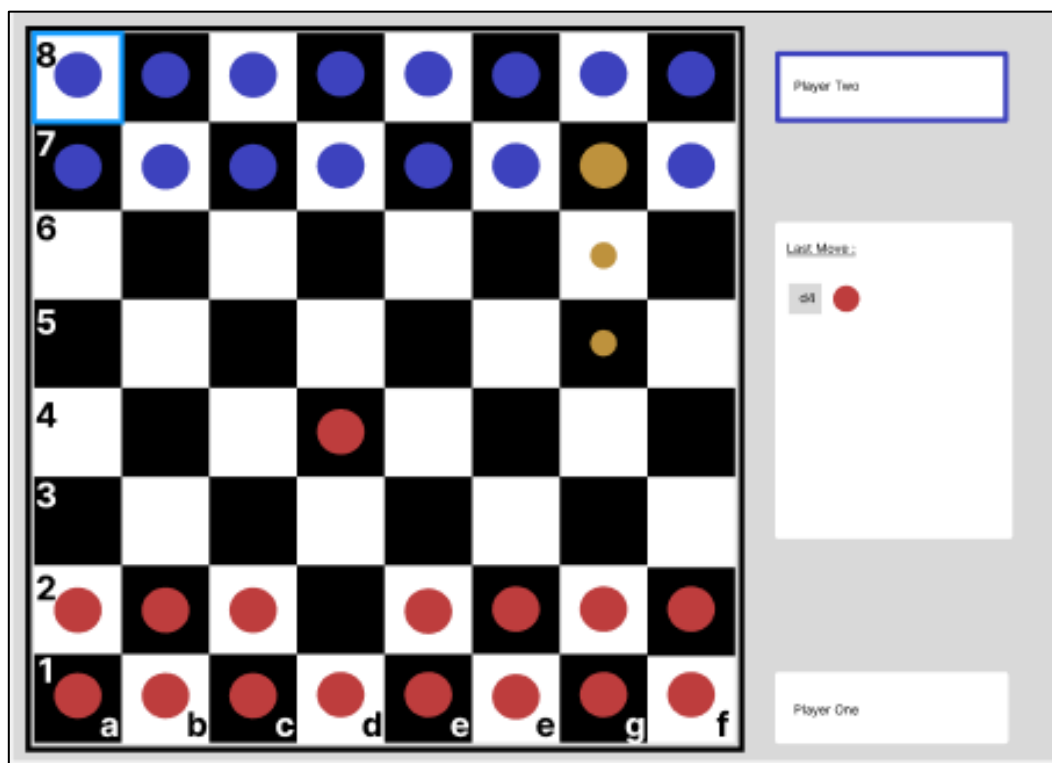


Figure 7 Figma joueur 2

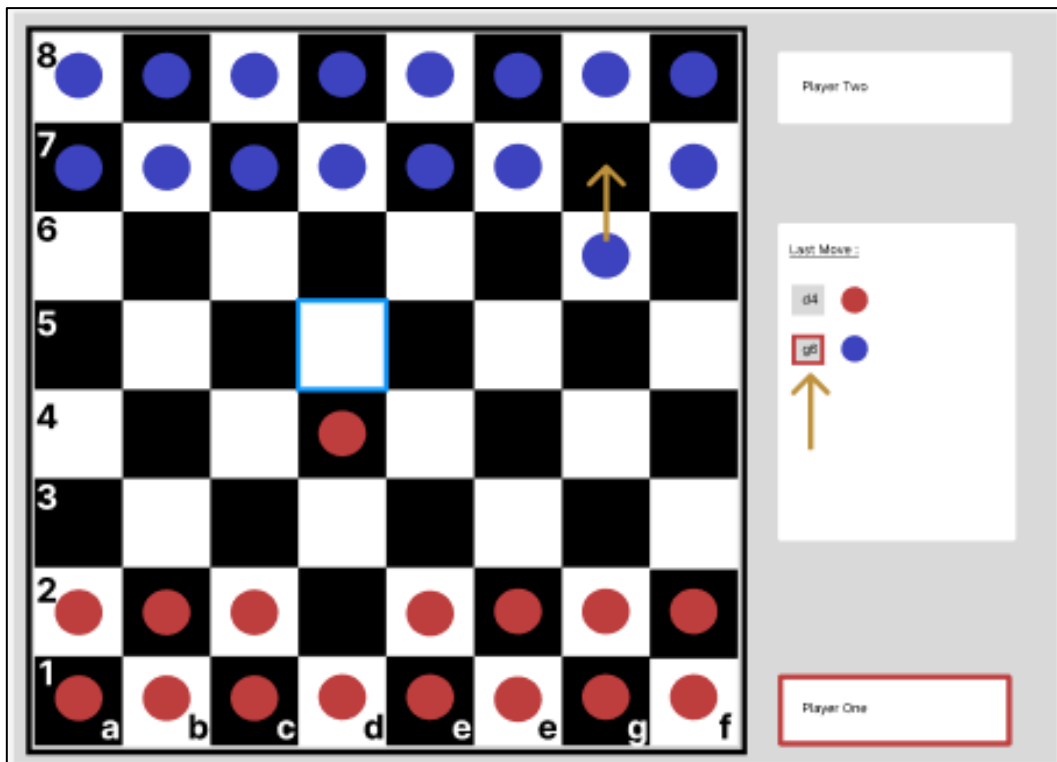


Figure 8 Figma dernier mouvement

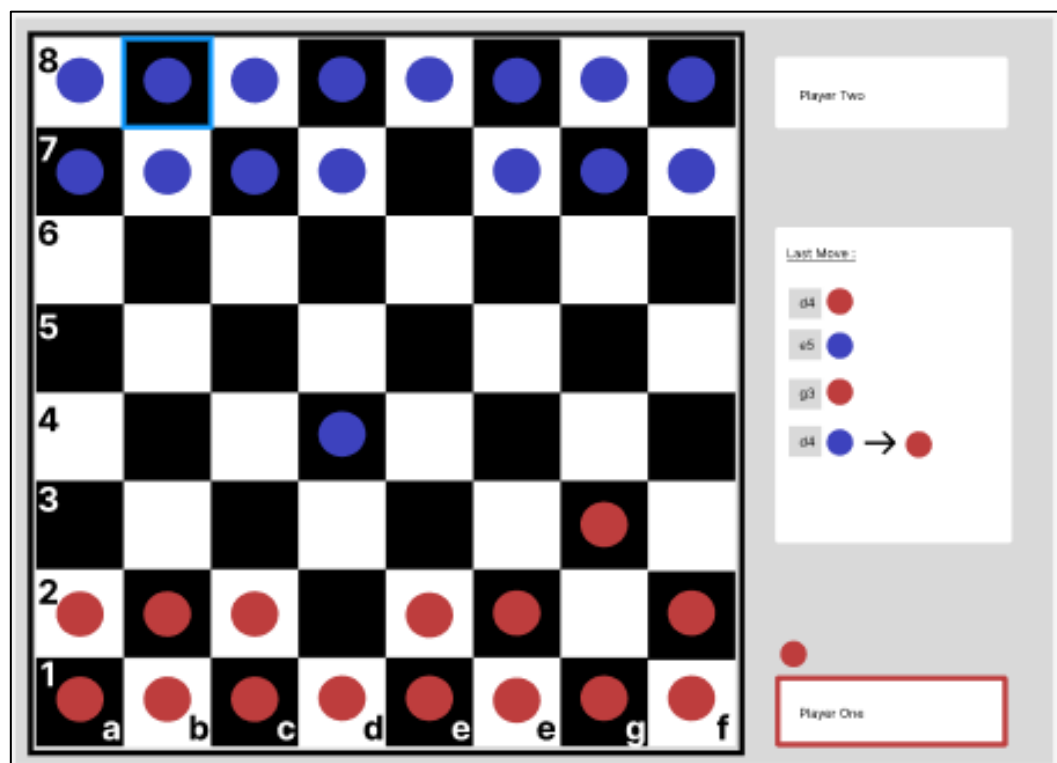


Figure 9 Figma pièce prise

## 4.7 Environnement Python

Ce projet est réalisé grâce au langage de programmation Python qui a besoin de quelques configurations et d'importation de bibliothèque avant de pouvoir être utilisé. La première étape est d'installer Python version 3.11.2 et d'ajouter Python au chemin d'accès. Cela permettra d'ajouter automatiquement Python à la variable d'environnement PATH.

Pour l'utilisation de la bibliothèque Pygame, l'utilisation de cette ligne de commande est nécessaire dans un terminal.

```
pip install pygame
```

Figure 10 Installation PyGame

## 4.8 Répertoire Git

Pour une gestion des versions efficaces et partagées, le projet est en tous temps disponibles dans un répertoire Git.

Pour cela, il est nécessaire d'utiliser la commande suivante dans un terminal pour initialiser Git dans le dossier souhaité.

```
PS D:\TPI\ChessPy> git init
```

Figure 11 Git initialisation

Puis, ces lignes de commandes permettent de connecter ce dossier au répertoire en ligne se trouvant sur GitHub et de mettre à jour le dossier et le répertoire.

```
PS D:\TPI\ChessPy> git remote add origin https://github.com/LovIMark/TPI_ChessPy_MarkLovink.git
PS D:\TPI\ChessPy> git pull https://github.com/LovIMark/TPI_ChessPy_MarkLovink.git
```

Figure 12 Git connexion

Une fois ces étapes effectuées, la création d'une nouvelle branche pour le développement est possible grâce aux lignes de commande ci-dessous.

```
PS D:\TPI\ChessPy> git checkout -b dev
Switched to a new branch 'dev'
PS D:\TPI\ChessPy> git branch -a
* dev
main
```

Figure 13 Git nouvelle branche

Finalement, un fichier permettant d'ignorer certains fichiers lors des mises à jour se nommant « .gitignore » est créé.



 .git	15.05.2023 08:18	Dossier de fichiers	
 .gitignore	15.05.2023 08:38	Document texte	1 Ko

Figure 14 Git ".gitignore"

## 4.9 Bibliothèque

Pygame est une bibliothèque populaire pour développer des jeux et des applications multimédias en Python. Il fournit des fonctions telles que la création de graphiques, la gestion des entrées de l'utilisateur, la lecture de sons et de musique et la manipulation d'images. Pygame simplifie le processus de création de jeux en fournissant une interface simple pour gérer les éléments importants tels que les fenêtres, les sprites\* et les événements.

## 4.10 Technologies utilisées

Nom du logiciel / bibliothèque	Version
VSCode	1.78.2
Python	3.11.2
Pygame	2.3.0
ChatGPT	GPT-3.5

## 4.11 Stratégie de test

La réalisation de tests se fait principalement de deux manières différentes : des tests fonctionnels, comme l'affichage de données, et des tests utilisateurs, comme les réactions du programme durant une vraie partie.

Nom de test	Scénario	Résultats attendus
Affichage du plateau	A l'ouverture du programme, un plateau d'échec s'affiche.	Une fois le jeu lancé, un plateau d'échec constitué de plusieurs carrés (noirs et blancs) s'affiche de manière régulière.
Affichage des pièces	A l'ouverture du programme, les pièces et pions s'affichent sur le plateau.	Une fois le plateau complètement affiché, les pions et pièces apparaissent à leur place respective sur le plateau de jeu. Chaque pièce a sa propre forme, chaque joueur a une couleur de pièces distinct.
Déplacements légaux des pièces	Les pièces, une fois en mouvement, ont une restriction sur leurs mouvements.	Chaque pièce se déplace seulement selon ses mouvements respectifs désignés par la FIDE.
Affichage des déplacements légaux des pièces	Pour faciliter l'utilisation du jeu, donc les mouvements possibles de chaque pièce, un affichage des déplacements est présent.	Une fois un pion ou pièce sélectionné par l'utilisateur, un affichage instinctif est présent sur le plateau de jeu.

Mouvements des pièces (utilisateur)	L'utilisateur du programme peut déplacer ses pièces	Une pièce peut être sélectionnée à l'aide de la souris et sa position est instantanément changée pour devenir la même que la position de la souris.
Nouvelle position (utilisateur)	L'utilisateur du programme a la possibilité de changer ses pièces d'échec de position sur le plateau	Une fois la pièce sélectionnée et déplacée l'utilisateur utilise le click de la souris pour confirmer la nouvelle position de la pièce (dans la limite des mouvements possibles)
Prise de pièce	Si les pions ont la même position sur le plateau, une des pièces a été perdue	La nouvelle pièce s'affiche correctement sur le plateau, elle garde ses mouvements, la pièce prise est affichée sous le nom de son propriétaire
Enregistrement des coups	Lors d'un déplacement de pièce ou pion ces derniers sont enregistrés	A côté du plateau de jeu, une liste avec le dernier déplacement est disponible
Affichage du dernier mouvement	Un bouton est disponible pour afficher le dernier mouvement de pièce effectué	L'affichage clair du dernier déplacement est disponible et aucun joueur ne peut déplacer ses pions durant ce temps
Situation d'échec	Des restrictions de mouvement sont implémentées	Pour empêcher le joueur de mettre son roi en échec, une restriction de mouvement des pièces est affichée, si le roi est déjà en échec, les pions pourront se déplacer seulement pour le protéger.
Situation d'échec et mat	Un joueur ne peut plus protéger son roi	Tout mouvement est bloqué et une fenêtre affiche le nom du gagnant

#### 4.12 Risques techniques

Le risque technique dans le projet est la manipulation de caractère pour créer les différentes pièces et pions à la place d'images.

Un deuxième risque pour la bonne réussite de ChessPY est la récente familiarisation avec le langage Python.

\*Voir Glossaire



## 5 Réalisation

Cette partie du document contient les points-clés permettant de réaliser le projet pratiquement dans son intégralité. Les étapes seront concentrées sur les fonctionnalités mentionnées précédemment.

### 5.1 Répertoire de dossier

Le répertoire de dossier pour réaliser ce projet se présente comme ceci pour faciliter la compréhension et la modification des fichiers :

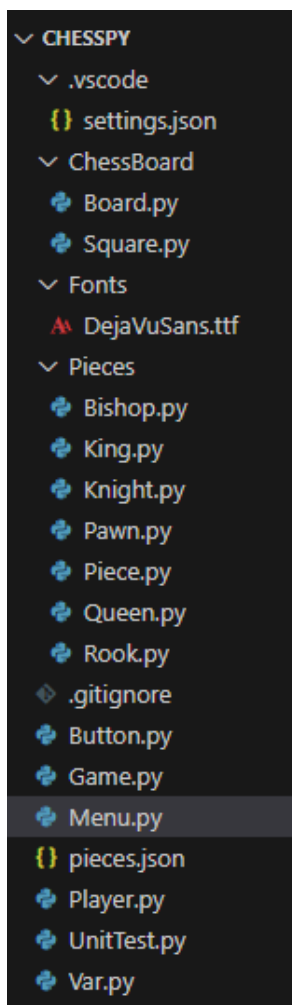


Figure 15 Répertoire de travail

Les pièces sont séparées du plateau de jeu et de la partie graphique du jeu, constitués des boutons, du menu et du jeu.

## 5.2 Boucle principale

Pygame permet de créer des fenêtres en plus de la gestion d'évènement. L'implémentation d'un jeu grâce à cette bibliothèque se fait sur la base d'une boucle qui met à jour automatiquement les variables et fonctions se trouvant dedans. La première étape est d'importer et d'instancier la bibliothèque avant de définir les dimensions de la fenêtre souhaitées comme montré dans l'image ci-dessous :

```
#Import of library |
import pygame

pygame.init()

HEIGHT=800
WIDTH=1200
#set the dimension of the window
window = pygame.display.set_mode((WIDTH, HEIGHT))
```

Figure 16 PyGame Init

La boucle principale se constitue seulement d'une boucle « While » et de la gestion d'évènement si l'utilisateur souhaite fermer la fenêtre

```
Run=True
#Main loop that display the pygame window and the game(board,pawn,pieces)
while Run:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            quit()
    pygame.display.flip()
```

Figure 17 Pygame boucle principale

### 5.3 Affichage du plateau d'échec

Pour l'affichage du plateau de jeu, donc du quadrillage noir et blanc du jeu ainsi que de son bord, il faut d'abord créer une classe « Square » qui est chaque carré contenu dans le plateau.

```
class Square():  
    def __init__(self, x, y, col, row, size, color, empty=True, name=None):  
        super().__init__()  
        self.rect = pygame.Rect(x, y, size, size)  
        self.image = pygame.Surface((size, size))  
        self.image.fill(color)  
        self.col=col  
        self.row=row  
        self.empty=empty  
        self.name=name
```

Figure 18 Classe "Square"

L'image ci-dessus contient les différentes variables pour la réalisation d'un carré :

- self.rect est la dimension de chaque carré
- self.image est toute la surface du carré
- self.image.fill remplit toute la surface du carré d'une couleur (noir ou blanc)
- self.col est la valeur sur quelle colonne du plateau se trouve le carré
- self.row est la valeur sur quelle ligne du plateau se trouve le carré
- self.empty est une variable permettant de savoir si le carré est vide ou non

Une fois la classe « Square » conçue, l'intégration de cette dernière dans la classe « Board » est nécessaire.

```
class Board():  
  
    def __init__(self,x,y):  
        self.size=8*WIDTH_SQUARE  
        self.border=5  
        self.x=x  
        self.y=y  
        self.showLastMovement=False  
        self.piecesPos=[]  
        self.squares=[]  
        self.allMovement=[]  
        self.piecesDie=[]  
        self.checkPos=[]  
        self.Drawself()  
        self.DrawPieces()
```

Figure 19 Classe "Board"

La classe est constituée des variables suivantes :

- self.size est la dimension totale du plateau de jeu
- self.border est l'épaisseur du bord du plateau
- self.x est la position de départ du plateau sur l'axe des abscisses
- self.y est la position de départ du plateau sur l'axe des ordonnées
- self.showLastShowMovement est une variable permettant de savoir si le dernier mouvement joué doit être affiché
- self.piecesPos est un tableau à deux dimensions contenant toutes les positions des pièces
- self.squares est un tableau à deux dimensions contenant tous les carrés de la classe « Square »
- self.allMovement est un tableau contenant les mouvements joués
- self.piecesDie est une liste de toutes les pièces prises lors d'un jeu
- self.checkPos est une liste de toutes les positions possibles entre la pièce adverse et le roi
- self.DrawBoard() initie la fonction récupérant toutes les positions du quadrillage
- self.DrawPieces() initie la fonction récupérant toutes les positions des pièces du plateau

Pour remplir le tableau des carrés, il suffit de deux boucles avec comme limite la dimension de plateau d'échec normale, 8x8, où la couleur de ce dernier change un carré sur deux, comme montré dans la fonction ci-dessous :

```
#Function that save in a table the position of all the square for the board
def DrawBoard(self):
    self.Square = [[0] * COL for _ in range(ROW)]

    for row in range(ROW):
        for col in range(COL):
            if col%2==0 and row%2==0:
                self.Square[col][row]=Square(self.x+(WIDTHSQUARE*col),self.y+(WIDTHSQUARE*row),col,row,WIDTHSQUARE,WHITE)
            elif col%1==0 and row%2==0:
                self.Square[col][row]=Square(self.x+(WIDTHSQUARE*col),self.y+(WIDTHSQUARE*row),col,row,WIDTHSQUARE,BLACK)
            elif col%2==0 and row%1==0:
                self.Square[col][row]=Square(self.x+(WIDTHSQUARE*col),self.y+(WIDTHSQUARE*row),col,row,WIDTHSQUARE,BLACK)
            else:
                self.Square[col][row]=Square(self.x+(WIDTHSQUARE*col),self.y+(WIDTHSQUARE*row),col,row,WIDTHSQUARE,WHITE)

    return self.Square
```

Figure 20 Fonction "DrawBoard"

Dans le cas présent, la première ligne permet d'obliger la variable self.squares de devenir une table à deux dimensions.

Pour l'affichage du bord de plateau de jeu la figure ci-dessous montre l'utilisation de la fonction « pygame.draw.rect », qui est une fonction de la bibliothèque PyGame permettant d'afficher un rectangle en y insérant la fenêtre sur laquelle il va s'afficher, sa couleur et la position dans cette fenêtre.

```
#Function that draw the border of the board
def DrawBorder(self,window):
    pygame.draw.rect(window, BLACK, (self.x-(self.border*2),self.y-(self.border*2),self.size+(self.border*4),self.size+(self.border*4)),self.border)
```

Figure 21 Fonction "DrawBoarder"

Enfin il suffit d'implémenter la classe « board » dans le fichier principal.

```
board=Board(20,20)
```

Figure 22 Implémentation classe "Board"

Insérer le tableau contenant les positions dans la fonction d'affichage du jeu complet.

```
# Function that get all the squares positions and display it in the pygame window
def Draw(board,window):

    for row in range(ROW):
        for col in range(COL):
            window.blit(board.Square[col][row].image, board.Square[col][row].rect)
```

Figure 23 Fonction affichage complet du jeu

Finalement faire appel à la fonction pour le bord du jeu et insérer la fonction d'affichage du jeu complet dans la boucle principale.

```
while Run:
    window.fill(GREY)
    board.DrawBorder(window)
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            quit()
    Draw(board,window)
    pygame.display.flip()
```

Figure 24 Boucle principale avec affichage

Voici le résultat une fois le jeu lancé.

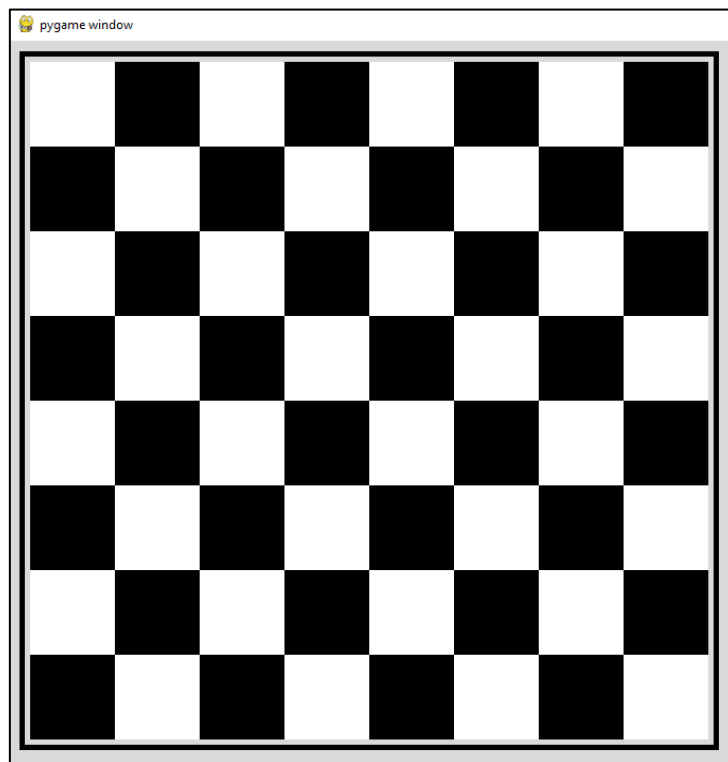


Figure 25 Affichage du plateau d'échec

## 5.4 Affichage des pions et pièces

Pour avoir un affichage complet d'un jeu d'échec, il existe deux types de pièces : les pions qui composent toutes une ligne du plateau et les pièces composées du roi, de la dame, des fous, des cavaliers et des tours qui à eux composent une autre ligne.

Une classe parente « Piece » est créée pour faciliter la réutilisation de certaines variables se trouvant dans toutes les classes de pièces et pions.

```
class Piece():  
  
    def __init__(self,x,y,size,color,col,row,name):  
        super().__init__()  
        self.rect = pygame.Rect(x, y, size, size)  
        self.font = pygame.font.Font("Fonts/DejaVuSans.ttf", size)  
        self.color=color  
        self.name=name  
        self.col=col  
        self.row=row  
        self.clicked=False  
        self.possibleMoves=[]
```

Figure 26 Classe parent "Piece"

Cette classe contient plusieurs variables telles que :

- self.rect est toute la surface de la pièce grâce à PyGame
- self.font est la liste des polices de caractère utilisées pour l'affichage des chaque pièces
- self.color est la couleur de la pièce
- self.col est la valeur sur quelle colonne du plateau se trouve la pièce
- self.row est la valeur sur quelle ligne du plateau se trouve la pièce
- self.clicked est une variable permettant de savoir si une pièce a été cliquée ou non
- self.possibleMoves est une liste qui contient tous les mouvements possibles de chaque pièce

### 5.4.1 Pions

Les pions n'ont qu'une classe héritant de la class « Piece ».

```
class Pawn(Piece):  
  
    def __init__(self,x,y,size,color,unicode,col,row):  
        super().__init__(x,y,size,color,col,row,"pawn")  
        self.image = self.font.render(unicode, True, color)  
        self.firstMove=False
```

Figure 27 Classe "Pawn"

Cette classe est constituée des variables suivantes :

- self.image est une variable contenant le caractère pour afficher un pion
- self.firstMove est une variable permettant de savoir si le pion a déjà été déplacé

Pour l'implémentation de cette dernière, une fonction « DrawPieces » dans la classe « Board » est utilisée, cette fonction permet de remplir le tableau « piecesPos » précédemment présenté.

```
def DrawPieces(self):  
    self.Square = [[0] * COL for _ in range(ROW)]  
    for row in range(8):  
        for col in range(8):  
            if row==1 :  
                self.PiecesPos[col][row]=Pawn(self.x+(WIDTHSQUARE*col),self.y+(WIDTHSQUARE*row),WIDTHSQUARE, BLUE,PAWN,col,row)  
                self.Square[col][row].empty=False  
            elif row==6:  
                self.PiecesPos[col][row]=Pawn(self.x+(WIDTHSQUARE*col),self.y+(WIDTHSQUARE*row),WIDTHSQUARE, RED,PAWN,col,row)  
                self.Square[col][row].empty=False  
  
    return self.PiecesPos
```

Figure 28 Fonction "DrawPieces" pions

Cette fonction utilise une double boucle parcourant tout le tableau en y insérant les pions avec leurs positions sur la deuxième et septième ligne du plateau de jeu.



### 5.4.2 Pièces

Les pièces héritent aussi de la classe « Piece » mais possèdent pour chaque types une classe singulière :

```
class King(Piece):  
  
    def __init__(self,x,y,size,color,unicode,col,row):  
        super().__init__(x,y,size,color,col,row,"king")  
        self.image = self.font.render(unicode, True, color)  
        self.kingMoves = [(0, 1), (1, 1), (-1, 1), (0, -1), (1, -1), (-1, -1), (1, 0), (-1, 0)]
```

Figure 29 Classe "King"

```
class Queen(Piece):  
  
    def __init__(self,x,y,size,color,unicode,col,row):  
        super().__init__(x,y,size,color,col,row,"queen")  
        self.image = self.font.render(unicode, True, color)  
        self.queenMoves = [(1, 1), (-1, 1), (1, -1), (-1, -1), (1, 0), (-1, 0), (0, 1), (0, -1)]
```

Figure 30 Classe "Queen"

```
class Bishop(Piece):  
  
    def __init__(self,x,y,size,color,unicode,col,row):  
        super().__init__(x,y,size,color,col,row,"bishop")  
        self.image = self.font.render(unicode, True, color)  
        self.bishopMoves = [(1, 1), (-1, 1), (1, -1), (-1, -1)]
```

Figure 31 Classe "Bishop"

```
class Rook(Piece):  
  
    def __init__(self,x,y,size,color,unicode,col,row,id):  
        super().__init__(x,y,size,color,col,row,"rook",id)  
        self.image = self.font.render(unicode, True, color)  
        self.rookMoves=[(0,1),(1,0),(0,-1),(-1,0)]
```

Figure 32 Classe "Rook"

```
class Knight(Piece):  
  
    def __init__(self,x,y,size,color,unicode,col,row):  
        super().__init__(x,y,size,color,col,row,"knight")  
        self.image = self.font.render(unicode, True, color)  
        self.knightMoves = [(1, 2), (2, 1), (-1, 2), (-2, 1), (1, -2), (2, -1), (-1, -2), (-2, -1)]
```

Figure 33"Classe "Knight"

Chaque classe possède un « name » différent.

Elles sont constituées des variables suivantes :

- self.image est une variable contenant le caractère pour afficher un pion
- self.\*name\*Moves est une liste qui contient les mouvements possibles de chaque pièce

Pour l'implémentation de toutes ces classes, la fonction « DrawPieces » de la classe « Board » est de nouveau utilisée, cette fonction permet de remplir le tableau « piecesPos » précédemment présenté.

```
def DrawPieces(self):
    self.PiecesPos = [[0] * COL for _ in range(ROW)]
    for row in range(8):
        for col in range(8):
            if row==0:
                if col==0 or col==7:
                    self.PiecesPos[col][row]=Rook(self.x+(WIDTHSQUARE*col),self.y+(WIDTHSQUARE*row),WIDTHSQUARE, BLUE,ROOK,col,row)
                    self.Square[col][row].empty=False
                elif col==1 or col==6:
                    self.PiecesPos[col][row]=Knight(self.x+(WIDTHSQUARE*col),self.y+(WIDTHSQUARE*row),WIDTHSQUARE, BLUE,KNIGHT,col,row)
                    self.Square[col][row].empty=False
                elif col==2 or col==5:
                    self.PiecesPos[col][row]=Bishop(self.x+(WIDTHSQUARE*col),self.y+(WIDTHSQUARE*row),WIDTHSQUARE, BLUE,BISHOP,col,row)
                    self.Square[col][row].empty=False
                elif col==3:
                    self.PiecesPos[col][row]=Queen(self.x+(WIDTHSQUARE*col),self.y+(WIDTHSQUARE*row),WIDTHSQUARE, BLUE,QUEEN,col,row)
                    self.Square[col][row].empty=False
                elif col==4:
                    self.PiecesPos[col][row]=King(self.x+(WIDTHSQUARE*col),self.y+(WIDTHSQUARE*row),WIDTHSQUARE, BLUE,KING,col,row)
                    self.Square[col][row].empty=False
            elif row==7:
                if col==0 or col==7:
                    self.PiecesPos[col][row]=Rook(self.x+(WIDTHSQUARE*col),self.y+(WIDTHSQUARE*row),WIDTHSQUARE, RED,ROOK,col,row)
                    self.Square[col][row].empty=False
                elif col==1 or col==6:
                    self.PiecesPos[col][row]=Knight(self.x+(WIDTHSQUARE*col),self.y+(WIDTHSQUARE*row),WIDTHSQUARE, RED,KNIGHT,col,row)
                    self.Square[col][row].empty=False
                elif col==2 or col==5:
                    self.PiecesPos[col][row]=Bishop(self.x+(WIDTHSQUARE*col),self.y+(WIDTHSQUARE*row),WIDTHSQUARE, RED,BISHOP,col,row)
                    self.Square[col][row].empty=False
                elif col==3:
                    self.PiecesPos[col][row]=Queen(self.x+(WIDTHSQUARE*col),self.y+(WIDTHSQUARE*row),WIDTHSQUARE, RED,QUEEN,col,row)
                    self.Square[col][row].empty=False
                elif col==4:
                    self.PiecesPos[col][row]=King(self.x+(WIDTHSQUARE*col),self.y+(WIDTHSQUARE*row),WIDTHSQUARE, RED,KING,col,row)
                    self.Square[col][row].empty=False
```

Figure 34 Fonction "DrawPieces" pièces

L'utilisation de la double boucle précédemment présentée dans le chapitre **Pions 5.4.1** permet de placer correctement toutes les pièces sur le plateau d'échec.

Finalement, l'affichage complet de toutes les pièces et pions est possible grâce à la même fonction utilisée pour l'affichage du plateau de jeu dans le fichier principal du jeu :

```
def Draw(board,window,players):  
    for row in range(ROW):  
        for col in range(COL):  
            if board.PiecesPos[col][row]!=0 and not board.PiecesPos[col][row].clicked :  
                window.blit(board.PiecesPos[col][row].image,(board.PiecesPos[col][row].col*WIDTHSQUARE+board.x,board.PiecesPos[col][row].row*WIDTHSQUARE+board.y))
```

Figure 35 Fonction affichage complet du jeu

Si la position de la double boucle dans la liste n'est pas vide il affiche l'image de la pièce ou du pion avec leurs position dans la fenêtre du jeu.

La figure ci-dessous montre le résultat des deux derniers chapitres :

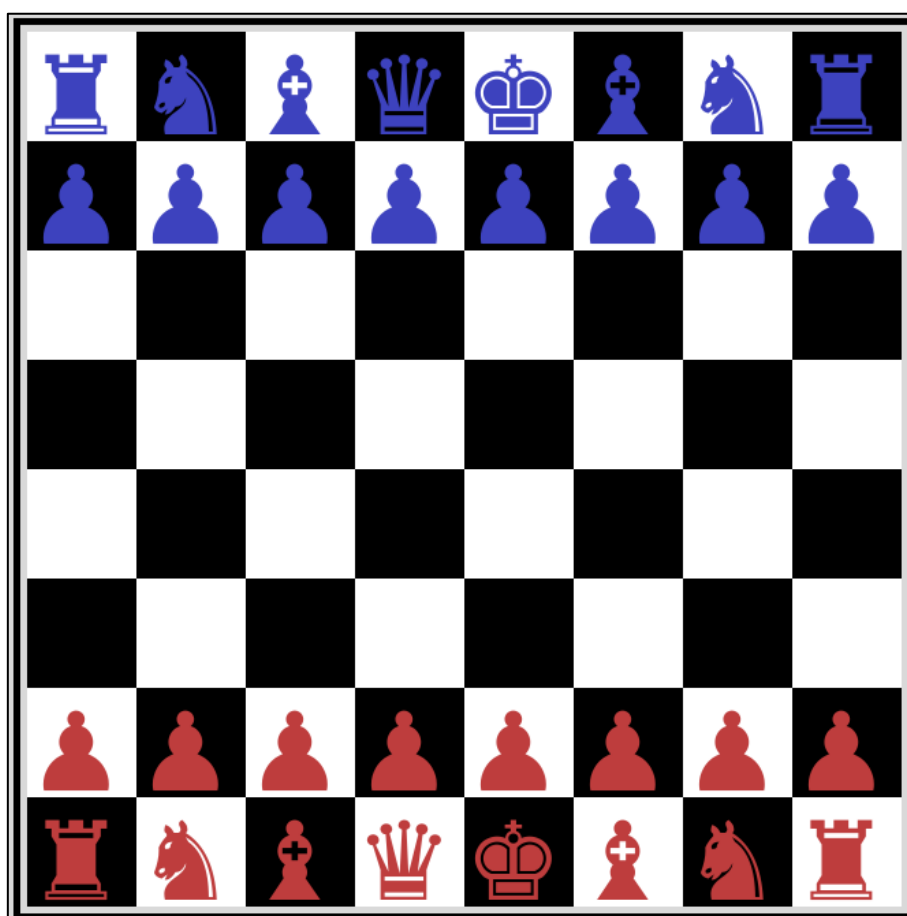


Figure 36 Affichage du plateau et pièces

## 5.5 Mouvement

Dans un jeu d'échec, chaque joueur peut déplacer ses pions. Dans le cas de PyChess cette action est possible grâce aux événements « MOUSEMOTION » qui permettent de suivre les mouvements de la souris et « MOUSEBUTTONDOWN » qui permet de suivre les cliques de la souris se trouvant dans la bibliothèque PyGame.

Dans la boucle principale du jeu, après la vérification de la sortie du jeu grâce à l'événement « QUIT », l'ajouts des gestions d'événement est nécessaire :

```
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        pygame.quit()
        quit()
    elif event.type == pygame.MOUSEMOTION :
        posMouse = pygame.mouse.get_pos()
    elif event.type == pygame.MOUSEBUTTONDOWN :
        posMouse = pygame.mouse.get_pos()
```

Figure 37 Evènements souris

La variable « posMouse » récupère et met à jour la position de la souris après un clique ou un déplacement de souris.

Toutes les pièces d'un jeu d'échec ayant le même moyen de déplacement, la souris, deux fonctions sont créées dans la classe parente « Piece » et implémentées dans la boucle principale « MouvementPlayer() » permettant à chaque pièce de suivre la souris et « Clicked() » permettant de gérer le moment de suivi des pièces :

```
elif event.type == pygame.MOUSEBUTTONDOWN :
    posMouse = pygame.mouse.get_pos()
    for row in range(ROW):
        for col in range(COL):
            if board.PiecesPos[col][row]!=0:
                if playerOne.playing and board.PiecesPos[col][row].color==playerOne.color:
                    if board.PiecesPos[col][row].rect.collidepoint(posMouse):
                        board.PiecesPos[col][row].Clicked(posMouse,players,board)
                        stopLoops=True
                        break
```

Figure 38 Evènement cliqué

Pour chaque pièce existante dans le tableau à deux dimensions « piecesPos » et la vérification du tour du joueur, la boucle vérifie si la position de la souris et sur une des pièces du joueur grâce à la fonction « collidepoint » de PyGame, Si c'est le cas la fonction « Clicked() » se trouvant dans la classe « Piece » est appelée.

Dans la figure ci-dessous la fonction « Clicked() » est présentée :

```
def Clicked(self, posMouse, players, board):
    # If mouse on piece and another piece has not been selected
    if not self.CheckClicked(board) and self.rect==
pygame.Rect(self.col*WIDTH_SQUARE+board.x, self.row*WIDTH_SQUARE+board.y, WIDTH_SQUARE, WIDTH_SQUARE) :
    self.clicked=True
else:
    self.clicked=False
```

Figure 39 Fonction "Clicked"

Cette dernière fait appelle à la fonction CheckClicked() qui vérifie s'il n'y a pas déjà une pièce qui a été cliquée :

```
def CheckClicked(self, board):
    clicked=False
    for row in range(ROW):
        for col in range(COL):
            if board.PiecesPos[col][row]!=0 and board.PiecesPos[col][row].clicked:
                clicked =True
    return clicked
```

Figure 40 Vérification des pièces cliqué

Puis vérifie si la surface cliquée est bien la même que celle des pièces.  
Si c'est bien le cas, alors la pièce a été cliquée ce qui permet au programme de faire appel à la fonction « MouvementPlayer() ».

```
elif event.type == pygame.MOUSEMOTION :
    posMouse = pygame.mouse.get_pos()
    if not board.lastShowMovement:
        for row in range(ROW):
            for col in range(COL):
                if board.PiecesPos[col][row]!=0:
                    board.PiecesPos[col][row].MouvementPlayer(posMouse, board)
```

Figure 41 Suivi des pièces et souris

Pour chaque pièce sur le plateau, comme le montre la fonction ci-dessous, si l'une d'entre elles est cliquée le centre de la pièce est égale à la position de la souris

```
def MouvementPlayer(self, posMouse, board):
    if self.clicked :
        self.rect.center=posMouse
    elif self.rect!=pygame.Rect(self.col*WIDTHSQUARE+board.x, self.row*WIDTHSQUARE+board.y, WIDTHSQUARE, WIDTHSQUARE) :
        self.rect=pygame.Rect(self.col*WIDTHSQUARE+board.x, self.row*WIDTHSQUARE+board.y, WIDTHSQUARE, WIDTHSQUARE)
```

Figure 42 Suivi de surface cliquable des pièces

Cependant si aucune pièce n'est sélectionnée mais que la surface de la pièce est différente de la pièce, la surface est automatiquement remplacée sur la pièce.  
Cela évite la possibilité de bouger une pièce sans être dessus.

Le joueur ayant choisi la nouvelle position de sa pièce, le programme attend le prochain clic de souris pour fixer la pièce au plateau.

Pour se faire, dans la fonction « Clicked() » le jeu vérifie que le joueur ait bien lâché la pièce et vérifie si le mouvement est possible grâce au tableau « possibleMoves », expliqué dans le prochain chapitre **5.6 Coups légaux**.

Puis, cette fonction modifie les différentes valeurs des tableaux, pour l'affichage et les variables de lignes et colonnes de la pièce, pour correspondre à sa nouvelle position :

```
if not self.clicked:
    for row in range(ROW):
        for col in range(COL):
            if board.squares[col][row].rect.collidepoint(posMouse) :
                if self.possibleMoves[col][row]:
                    if not board.squares[col][row].empty:
                        board.piecesDie.append(board.piecesPos[col][row])
                        board.allMovement.append((self.col,self.row,col,row))
                        board.piecesPos[col][row]=board.piecesPos[self.col][self.row]
                        board.piecesPos[self.col][self.row]=0
                        board.squares[self.col][self.row].empty=True
                        board.squares[col][row].empty=False
                        self.col=col
                        self.row=row
                        self.rect=pygame.Rect(self.col*WIDTH_SQUARE+board.x,self.row*WIDTH_SQUARE+board.y,WIDTH_SQUARE,WIDTH_SQUARE)
                        self.firstMove=False
                    for obj in players:
                        obj.ChangePlayer()
```

Figure 43 Nouvelle position pièce

La fonction permet aussi de modifier le tour du joueur ainsi que d'ajouter dans la liste contenant toutes les pièces prises, la pièce perdue lors de ce tour.

Voici le résultat obtenu dans un scénario tel qu'un pion rouge atterrit sur la même case qu'un pion bleu, le pion bleu finit dans la liste des pièces prises du joueur bleu :

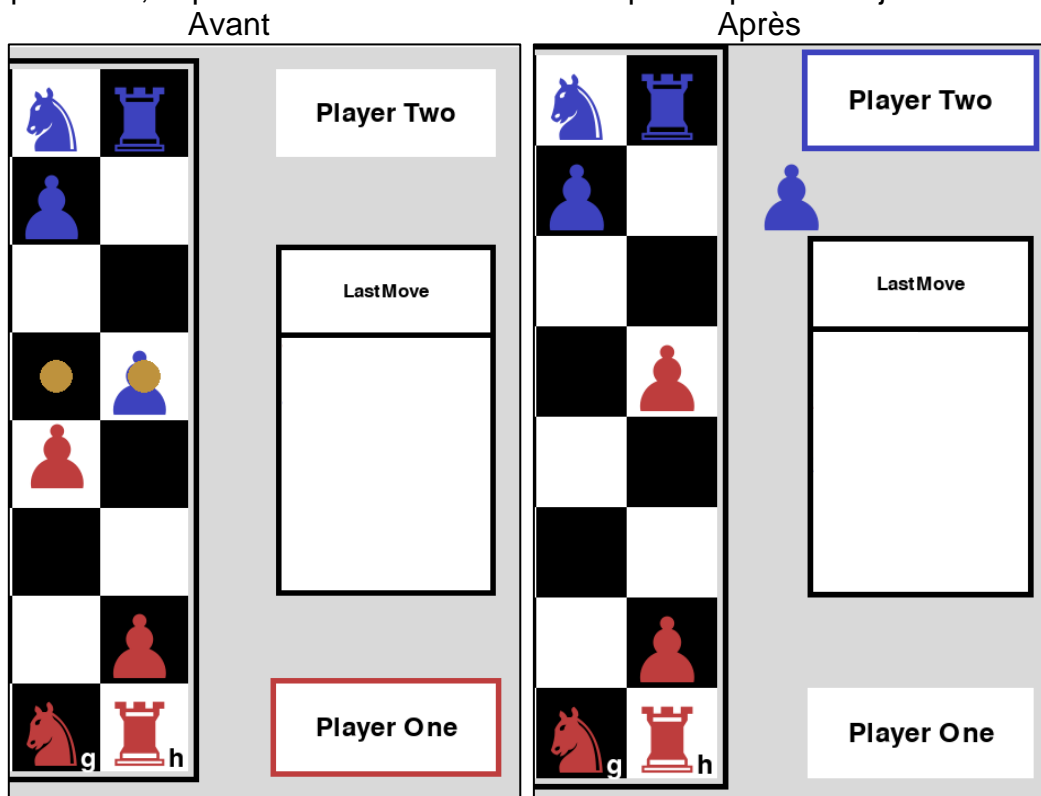


Figure 44 Résultat prise d'une pièce avant

Figure 45 Résultat prise de pièce après

## 5.6 Coups légaux

Pour compléter le jeu d'échec, certaines restrictions de mouvement pour les pièces doivent être implémentées.

Cette tâche est réalisée grâce aux fonctions « Mouvement » se trouvant dans chaque classe de pièces et pions ces derniers ayant tous des mouvements spécifiques.

Pour la vérification des coups légaux, l'utilisation du tableau « possibleMoves » existant dans chaque classe de pièces est nécessaire afin de réaliser la vérification dite dans le chapitre **5.5 Mouvement** à la **Figure 43**.

Une fonction vérifiant toutes les possibilités de tous les pions présents sur le plateau est implémentée et appelée dans la boucle principale du jeu :

```
def HandlePossibleMouvement(self,board):  
    for row in range(ROW):  
        for col in range(COL):  
            if board.piecesPos[col][row]!=0:  
                board.piecesPos[col][row].Mouvement(board)
```

Figure 46 Fonction des mouvement légaux

Cette fonction permet d'appeler toutes les fonctions vérifiant les mouvements légaux de toutes les pièces.

```
Run=True  
#Main loop that display the pygame window and the game(board,pawn,pieces)  
while Run:  
    self.window.fill(GREY)  
    board.DrawBorder(self.window)  
    for event in pygame.event.get(): ...  
    self.HandlePossibleMouvement(board)
```

Figure 47 Appelle fonction des mouvements légaux

Cette fonction est mise à jour à chaque itération de boucle.

### 5.6.1 Roi

Pour vérifier les possibilités de mouvement du roi, il faut déjà connaître ses coups légaux :

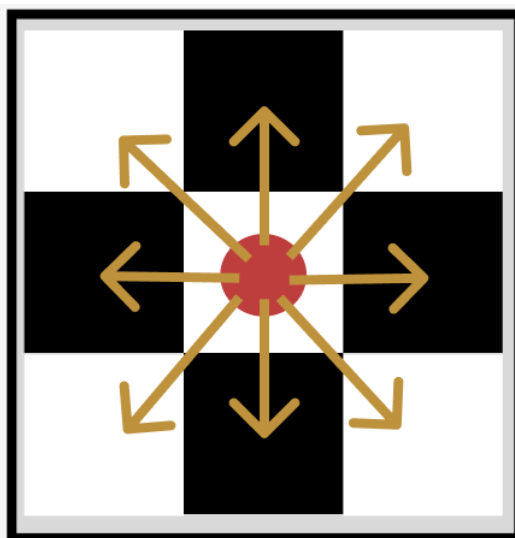


Figure 48 Coups légaux roi

Les mouvements sont présentés avec l'axe des abscisses et des ordonnées, en lien avec les lignes et les colonnes.

Pouvant se déplacer d'une case dans toutes les directions, la déclaration de ces derniers se font dans la classe du Roi, comme exposé dans l'image ci-dessous :

```
self.kingMoves = [(0, 1), (1, 1), (-1, 1), (0, -1), (1, -1), (-1, -1), (1, 0), (-1, 0)]
```

Figure 49 Mouvement du roi

Une fois ses coups légaux déclarés, sa fonction « Movement » parcourt sa liste de coups et permet d'enregistrer ses mouvements possibles en vérifiant, si la case est libre, si la case est occupée par une pièce de même couleur ou bien si une case est occupée par une pièce de couleur différente.

```
def Mouvement(self, board):
    #Create a two-dimensional table that save all the possible moves
    self.possibleMoves = [[0] * COL for i in range(ROW)]

    for obj in self.kingMoves:
        col = self.col + obj[0]
        row = self.row + obj[1]
        if 0 <= col < 8 and 0 <= row < 8 and board.squares[col][row].empty :
            self.possibleMoves[col][row] = True
        elif 0 <= col < 8 and 0 <= row < 8 and not board.squares[col][row].empty :
            if board.piecesPos[col][row] != 0 :
                if board.piecesPos[col][row].color != self.color:
                    self.possibleMoves[col][row] = True

    self.KingSimulation(board, self.possibleMoves)
```

Figure 50 Fonction "Movement" roi



### 5.6.2 Cavalier

Les mouvements du cavalier diffèrent de ceux du roi mais on la même base, qui est de se déplacer d'un d'une case précise de sa position actuelle.

Cette caractéristique permet d'utiliser la même fonction « Movement » que celle du roi présentée dans la **Figure 51**, il suffit de changer « self.kingMoves » avec les coups légaux du cavalier et enlever la méthode de vérification « KingSimulation ».

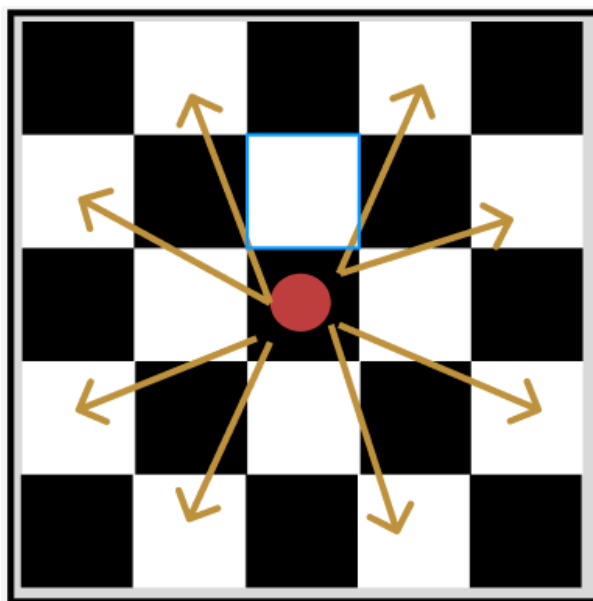


Figure 51 Coups légaux cavalier

La figure ci-dessus présente tous les mouvements du cavalier.

La figure ci-dessous présente les mêmes mouvements avec les lignes et colonnes du plateau :

```
self.knightMoves = [(1, 2), (2, 1), (-1, 2), (-2, 1), (1, -2), (2, -1), (-1, -2), (-2, -1)]
```

Figure 52 Mouvement du cavalier

Voici le résultat du cavalier :

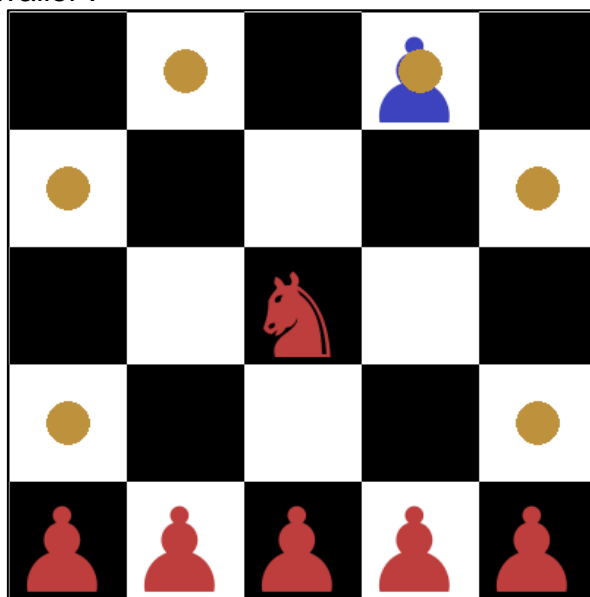


Figure 53 Résultat cavalier

### 5.6.3 Dame

La dame possède la plus grande capacité de déplacement de toutes les pièces, en plus de pouvoir se déplacer dans toutes les directions, comme montré sur la figure ci-dessous, elle peut se déplacer tout le long du plateau de jeu.

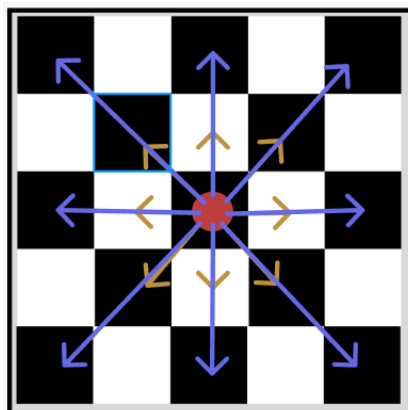


Figure 54 Coups légaux dame

Pour vérifier les possibilités de mouvement de la dame, la fonction « Movement » est modifiée pour respecter ces critères.

La direction des mouvements est d'abord implémentée dans une liste.

```
self.queenMoves = [(1, 1), (-1, 1), (1, -1), (-1, -1), (1, 0), (-1, 0), (0, 1), (0, -1)]
```

Figure 55 Mouvement de la dame

Une fois implémentée, la liste est parcourue dans la fonction, comme précédemment, mais cette fois une autre boucle est déclarée pour vérifier toutes les cases le long du plateau.

```
def Mouvement(self, board):
    self.possibleMoves.clear()
    self.possibleMoves = [[0] * COL for _ in range(ROW)]
    if self.clicked:
        for obj in self.queenMoves:
            for i in range(1, 8):
                col = self.col + obj[0] * i
                row = self.row + obj[1] * i
                if 0 <= col < 8 and 0 <= row < 8 and board.squares[col][row].empty:
                    self.possibleMoves[col][row] = True
                elif 0 <= col < 8 and 0 <= row < 8 and not board.squares[col][row].empty:
                    if board.piecesPos[col][row].color != self.color:
                        self.possibleMoves[col][row] = True
                    break
            else:
                break
```

Figure 56 Fonction "Movement" dame

Dans la figure ci-dessous, la boucle « for i in range (1, 8) » : cette fonction permet précisément de vérifier les 8 cases dans chaque direction à partir de la position de la dame. Les lignes et colonnes vérifiées sont changées en multipliant ces derniers.

Les « break » permettent de s'arrêter lors de la rencontre d'une pièce adverse ou d'une pièce de même couleur.

### 5.6.4 Fou

Pour les mouvements du fou et ses coups légaux, la même fonction « Movement » est utilisée, la seule différence sont ses coups légaux

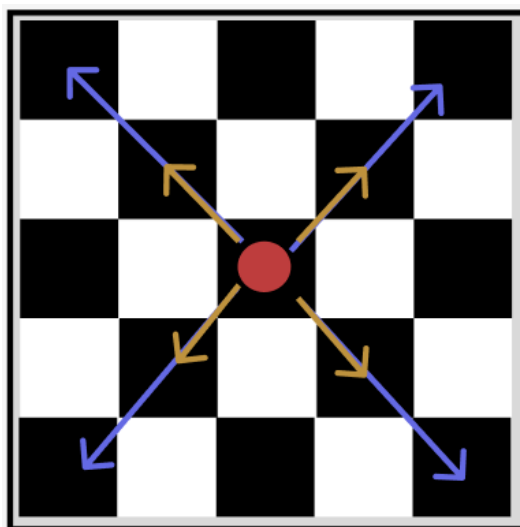


Figure 57 Coups légaux fou

La figure ci-dessus présente tous les mouvements du fou.

La figure ci-dessous présente les mêmes mouvements avec les lignes et colonnes du plateau :

```
self.bishopMoves = [(1, 1), (-1, 1), (1, -1), (-1, -1)]
```

Figure 58 Mouvement du fou

Voici le résultat du fou.

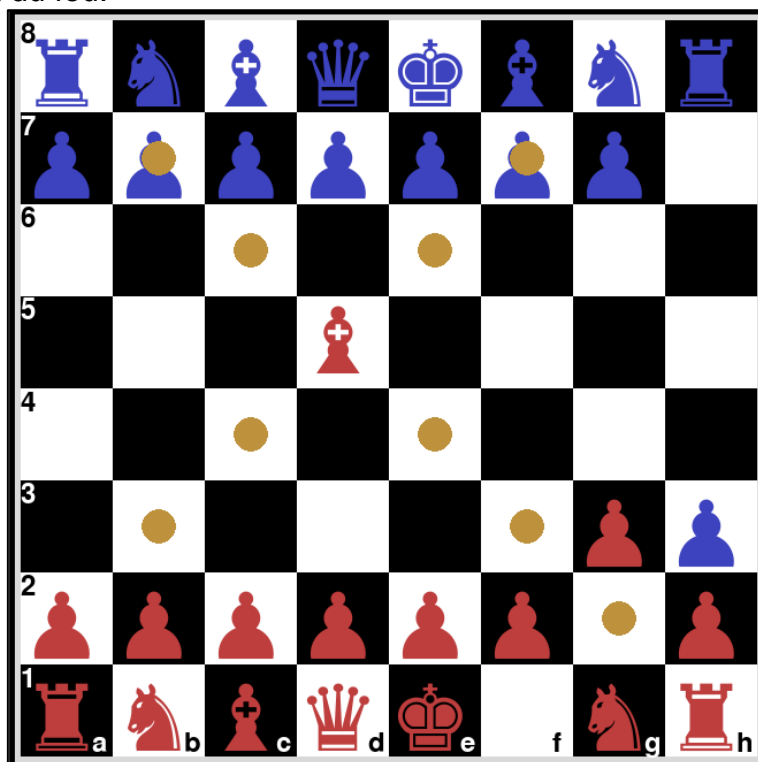


Figure 59 Résultat fou

### 5.6.5 Tour

Pour finir avec les mouvements des pièces, la tour utilise aussi la même fonction « Movement » décrite précédemment.

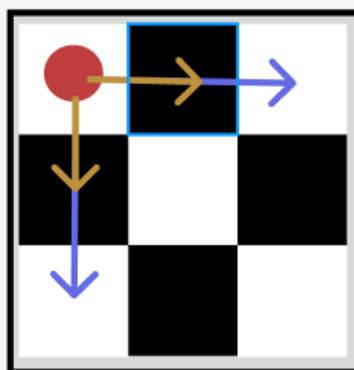


Figure 60 Coups légaux tour

Cette fois-ci, elle sera parcourue avec les coups légaux de la tour.

```
self.rookMoves=[(0,1),(1,0),(0,-1),(-1,0)]
```

Figure 61 Mouvement de la tour

Voici le résultat de la tour :

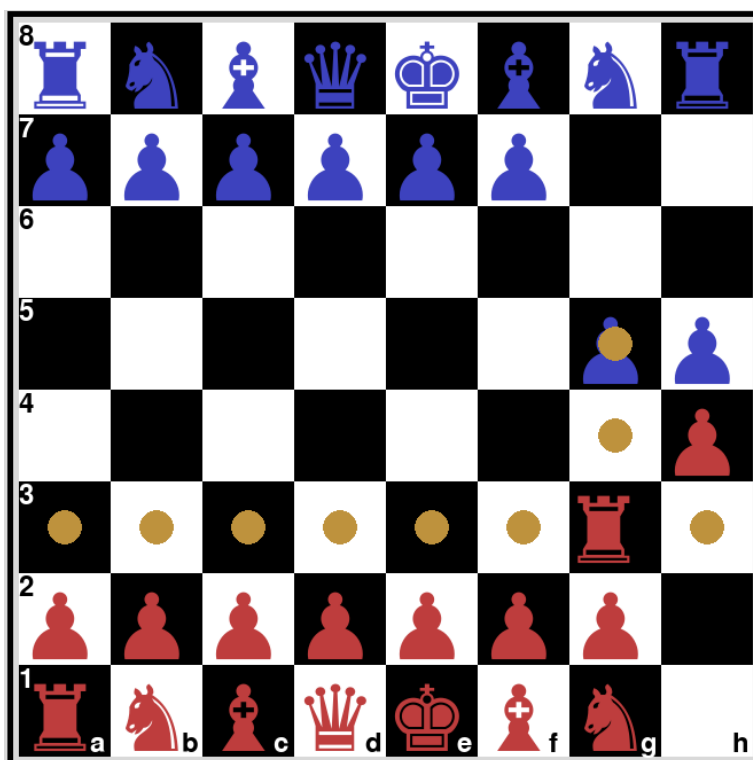


Figure 62 Résultat tour

### 5.6.6 Pion

Finalement, le dernier type de pièces existant dans les échecs sont les pions qui ont des critères de déplacement différents des autres pièces, car leur premier mouvement diffère du reste de leurs déplacements.

Pour se faire la classe du pion possède une variable supplémentaire « firstMove » qui permet de vérifier s'ils ont déjà effectué un déplacement ou non.

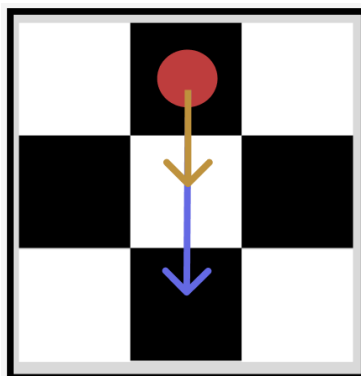


Figure 63 Coups légaux pion

Les figures ci-dessous présentent ces deux possibilités en fonction de leurs couleurs. Pour les pions bleus :

```
if self.color==BLUE :  
    if self.row < COL-2 and board.squares[self.col][self.row+2].empty and  
        board.squares[self.col][self.row+1].empty and self.firstMove:  
        self.possibleMoves[self.col][self.row+2]=True  
    if self.row < COL-1 and board.squares[self.col][self.row+1].empty :  
        self.possibleMoves[self.col][self.row+1]=True
```

Figure 64 Mouvement du pion bleu

Pour les pions rouges :

```
if self.color==RED |:  
    if self.row > 0 and board.squares[self.col][self.row-2].empty and  
        board.squares[self.col][self.row-1].empty and self.firstMove  
        self.possibleMoves[self.col][self.row-2]=True  
    if self.row > 0 and board.squares[self.col][self.row-1].empty :  
        self.possibleMoves[self.col][self.row-1]=True
```

Figure 65 Mouvement du pion rouge

Le pion en plus des exceptions précédentes, a un système de prise également différent. Pour prendre une pièce cette dernière doit être en diagonale du pion :

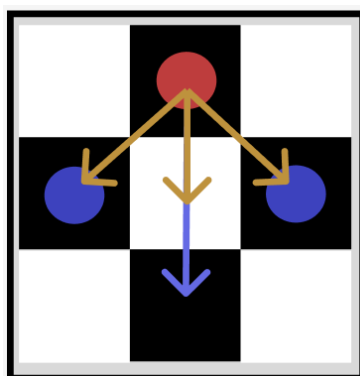


Figure 66 Coups légaux pion en attaque

Les figures ci-dessous présentent les ces deux possibilités en plus du déplacement normal en fonction de leurs couleurs.

Pour les pions bleus, possibilité d'attaque à droite :

```
if self.row < COL-1 and self.col >= 0 and self.col < 7 :
    if board.piecesPos[self.col+1][self.row+1]!=0 :
        if board.piecesPos[self.col+1][self.row+1].color!=self.color:
            self.possibleMoves[self.col+1][self.row+1]=True
        if board.piecesPos[self.col+1][self.row+1].name=="king": ...
```

Figure 67 Mouvement attaque pion bleu droite

Possibilité d'attaque à gauche :

```
if self.row < COL-1 and self.col > 0 and self.col <= 7 :
    if board.piecesPos[self.col-1][self.row+1]!=0 :
        if board.piecesPos[self.col-1][self.row+1].color!=self.color:
            self.possibleMoves[self.col-1][self.row+1]=True
        if board.piecesPos[self.col-1][self.row+1].name=="king": ...
```

Figure 68 Mouvement attaque pion bleu gauche

Pour les pions rouges, possibilité d'attaque à droite :

```
if self.row > 0 and self.col >= 0 and self.col < 7 :
    if board.piecesPos[self.col+1][self.row-1]!=0 :
        if board.piecesPos[self.col+1][self.row-1].color==BLUE:
            self.possibleMoves[self.col+1][self.row-1]=True
        if board.piecesPos[self.col+1][self.row-1].name=="king": ...
```

Figure 69 Mouvement attaque pion rouge droite

Possibilité d'attaque à gauche :

```
if self.row > 0 and self.col > 0 and self.col <= 7 :
    if board.piecesPos[self.col-1][self.row-1]!=0 :
        if board.piecesPos[self.col-1][self.row-1].color==BLUE:
            self.possibleMoves[self.col-1][self.row-1]=True
        if board.piecesPos[self.col-1][self.row-1].name=="king": ...
```

Figure 70 Mouvement attaque pion rouge gauche

Une fois que toutes les listes "possibleMoves" sont remplies, le jeu permet d'afficher les mouvements possibles pour aider les personnes en ayant besoin, avec un petit rond de couleur différente.

Pour ce faire, l'ajoute d'une condition dans la boucle principale d'affichage est implémentée, pour n'afficher l'aide que lorsque la pièce est cliquée.

```
for row in range(ROW):
    for col in range(COL):
        if board.piecesPos[col][row]!=0 and board.piecesPos[col][row].clicked:
            for row2 in range(ROW):
                for col2 in range(COL):
                    if board.piecesPos[col][row].possibleMoves[col2][row2]!=0:
                        pygame.draw.circle
```

Figure 71 Affichage des mouvements possibles

Voici les résultats du pion :



Figure 72 Résultat mouvement des pions

## 5.7 Échec

Dans les règles de la FIDE, si le roi d'un joueur est menacé d'être capturé au prochain coup de l'adversaire, on dit qu'il est en échec. Lorsqu'un roi est en échec, le joueur dont le roi est menacé doit prendre des mesures pour sortir de cette situation. Il existe trois options pour sortir d'échec :

- Déplacer le roi : le joueur peut déplacer son roi vers une case non menacée où il ne sera pas en échec.
- Bloquer la menace : le joueur peut placer une autre pièce entre son roi et la pièce menaçante afin de bloquer la ligne d'attaque.
- Capturer la menace : le joueur peut capturer la pièce qui menace son roi, la retirant ainsi de l'échiquier et mettant fin à l'échec.

### 5.7.1 Mise en échec

Pour vérifier que le roi soit en échec, le mouvement de toutes les pièces est modifié, ils ont une condition supplémentaire.

Si, durant le calcul de la possibilité de mouvement de la pièce, cette dernière bute contre le roi adverse, toutes les pièces adverses voient leurs valeur « check » changer en « vrai ».

```
if board.piecesPos[col][row].name=="king":
    for row2 in range(ROW):
        for col2 in range(COL):
            if board.piecesPos[col2][row2]!=0 and board.piecesPos[col2][row2].color!=self.color :
                board.piecesPos[col2][row2].check=True

    board.checkPos=self.getPositionsBetween(row,col,self.row,self.col)
```

Figure 73 Pièces en échec

Ci-dessus, une fonction "getPositionBetween" est appelée. Elle permet d'enregistrer dans une liste toutes les positions disponibles entre le roi adverse se trouvant en échec et la pièce menaçant le roi.

Cette liste est utilisée pour bloquer ou capturer la menace.

### 5.7.2 Position d'échec

La fonction "getPositionBetween" est utilisée pour les pièces qui peuvent se déplacer sur toute la longueur du plateau de jeu, comme la tour, la dame et le fou. Sinon, seules la ligne et la colonne de la menace sont enregistrées dans la liste "checkPos".

```
if board.piecesPos[col][row].name=="king":
    for row2 in range(ROW):
        for col2 in range(COL):
            if board.piecesPos[col2][row2]!=0 and board.piecesPos[col2][row2].color!=self.color :
                board.piecesPos[col2][row2].check=True

    board.checkPos=([self.row,self.col])
```

Figure 74 Liste simple "checkPos"



Pour les pièces pouvant se déplacer sur toute la longueur du plateau, la fonction crée une liste vierge qui ajoute automatiquement la ligne et la colonne de la menace.

```
def getPositionsBetween(self, kingRow, kingCol, rookRow, rookCol):  
    positionsBetween = []  
    positionsBetween.append((rookRow, rookCol))
```

Figure 75 Menace position

Pour la tour, la fonction vérifie si la pièce et le roi sont sur la même ligne ou sur la même colonne.

Puis, elle observe dans quelle direction la simulation de la ligne ou colonne va se faire en regardant si le roi est placé en dessous, bien au-dessus, à droite ou à gauche de la pièce.

```
if kingRow == rookRow:  
    if kingCol < rookCol:  
        postionColBetween= 1  
    else:  
        postionColBetween=-1  
  
elif kingCol == rookCol:  
    if kingRow < rookRow:  
        postionRowBetween= 1  
    else:  
        postionRowBetween=-1
```

Figure 76 Menace direction simulation tour

Une fois ces deux informations obtenues, tant que la simulation de la ligne ou colonne n'est pas à la même valeur que la position du roi, il va ajouter la position de la simulation dans la liste. Ci-dessous se trouve la vérification des colonnes.

```
col = kingCol + postionColBetween  
while col != rookCol:  
    positionsBetween.append((kingRow, col))  
    col += postionColBetween
```

Figure 77 Menace tour colonne

Ci-dessous se trouve la vérification des lignes.

```
row = kingRow + postionRowBetween  
while row != rookRow:  
    positionsBetween.append((row, kingCol))  
    row += postionRowBetween
```

Figure 78 Menace tour ligne

Pour la dame, la fonction vérifie si la pièce et le roi sont sur la même ligne, ou sur la même colonne ou sur la même diagonale.

D'abord, la fonction vérifie la distance entre la ligne et colonne du roi et de la dame :

```
distanceRowBetween = queenRow - kingRow
distanceColBetween = queenCol - kingCol
```

Figure 79 Menace distance roi et dame

Puis, elle observe dans quelle direction la simulation de la ligne ou colonne va se faire en regardant si le roi est placé en dessous, bien au-dessus, à droite ou à gauche de la pièce.

```
if distanceRowBetween > 0:
    postionRowBetween = 1
elif distanceRowBetween < 0:
    postionRowBetween = -1
else:
    postionRowBetween = 0

if distanceColBetween > 0:
    postionColBetween = 1
elif distanceColBetween < 0:
    postionColBetween = -1
else:
    postionColBetween = 0
```

Figure 80 Menace direction simulation dame

Une fois ces informations obtenues, la fonction ajoute à la liste les positions entre la pièce et le roi et retourne cette liste.

```
row= kingRow + postionRowBetween
col= kingCol + postionColBetween
while row != queenRow or col != queenCol:
    positionsBetween.append((row, col))
    row += postionRowBetween
    col += postionColBetween

return positionsBetween
```

Figure 81 Menace positions entre roi et dame

Finalement, pour la vérification du fou, sa fonction repose sur les mêmes principes mais en vérifiant seulement les diagonales.  
La vérification de direction de la simulation.

```
if kingRow < bishopRow:
    distanceRowBetween = 1
else:
    distanceRowBetween = -1

if kingCol < bishopCol:
    distanceColBetween = 1
else:
    distanceColBetween = -1
```

Figure 82 Menace direction fou

Puis, elle parcourt la distance entre le roi et le fou et ajoute les positions disponibles dans la liste.

```
row = kingRow + distanceRowBetween
col = kingCol + distanceColBetween
while row != bishopRow and col != bishopCol:
    positionsBetween.append((row, col))
    row += distanceRowBetween
    col += distanceColBetween
return positionsBetween
```

Figure 83 Menace positions entre roi et fou

### 5.7.3 Simulation d'échec

Il est important de noter que, selon les règles de la FIDE, un joueur ne peut pas effectuer un coup qui placerait ou laisserait son propre roi en échec.

Pour restreindre les mouvements des pièces pour ne pas effectuer de coup illégal, le programme vérifie à tout moment et pour toutes les pièces que, si un déplacement est effectué, celui-ci ne met pas en danger son propre roi, en simulant tous les coups possibles du prochain tour.

Pour ce faire, dans toutes les méthodes « Movement » de mouvement des pièces, une fonction « Simulation » simulant le prochain tour de la classe « Piece » est appelée.

```
def Mouvement(self,board):  
  
    #Create a two-dimensional table that save all the possible moves  
    self.possibleMoves= [[0] * COL for i in range(ROW)]  
    #if not self.check and not self.Simulation(board):  
    for obj in self.rookMoves: ...  
    if self.check or self.Simulation(board) :
```

Figure 84 Appel « Simulation »

Cette fonction va tout d'abord faire une copie intégrale du plateau de jeu.

```
def Simulation(self,board):  
    ### Copy the actual board  
    simulateBoard=[[0] * COL for i in range(ROW)]  
    for row in range(ROW):  
        for col in range(COL):  
            simulateBoard[col][row]=copy.copy(board.piecesPos[col][row])
```

Figure 85 Simulation du plateau

Ensuite, elle va retirer la pièce dont on souhaite vérifier les mouvements et simuler tous les mouvements possibles des pièces adverses pour le prochain tour, en prenant en compte l'absence de la pièce

```
###Remove the piece to simulate a move  
simulateBoard[self.col][self.row]=0  
###Simulate the next round with the actual piece removed  
for row in range(ROW):  
    for col in range(COL):  
        if simulateBoard[col][row]!=0 and simulateBoard[col][row].color!=self.color :  
            simulateBoard[col][row].MouvementSimulation(simulateBoard,board)
```

Figure 86 Simulation du prochain tour

La méthode « MouvementSimulation » reprend exactement la même méthode de mouvement de chaque pièce respective mais enlève la condition de simulation pour éviter une boucle infinie.

Finalement, une fois les mouvements simulés, le programme vérifie l'état de son roi, si celui-ci n'est pas en échec la fonction retourne faux, mais si le roi est en échec, la fonction retourne vrai.

```
for row in range(ROW):
    for col in range(COL):
        if simulateBoard[col][row]!=0 and simulateBoard[col][row].color==self.color :
            if simulateBoard[col][row].check:
                return True

return False
```

Figure 87 Simulation vérification échec

La méthode qui permet de restreindre les mouvements des pièces, si une simulation de cas d'échec s'avère vraie, est présentée dans le chapitre suivant **Protection et capture 5.7.4**.

La classe « King » détient sa propre simulation, car il ne peut pas lui-même se mettre en échec, avec la méthode « KingSimulation ».

Cette méthode reprend les mêmes caractéristiques que celle des autres simulations. Mais, au lieu d'enlever la pièce avec « simulateBoard[self.col][self.row]=0 » de la **Figure 86**, cette simulation vérifie tous les mouvements possibles du roi.

```
for obj in self.kingMoves:
    cols = self.col + obj[0]
    rows = self.row + obj[1]
```

Figure 88 Simulation mouvement du roi

Cette figure ci-dessus permet de vérifier tous les mouvements du roi, si ces mouvements sont dans la limite du plateau la simulation s'effectue.

```
###Change the kings position with all is possible moves
if 0 <= cols < 8 and 0 <= rows < 8 and possibleMoves[cols][rows]:
    ...
    ###Simulate the next round with the new position of the king
    ...
    ### If the new position of the king puts himself in check
    ...
    ### Reset simulation variable
    ...
    if board.piecesPos[cols][rows]==0:
        board.squares[cols][rows].empty=True
    else:
        board.squares[cols][rows].empty=False
return possibleMoves
```

Figure 89 Restriction de mouvement du roi

Si la simulation du nouveau positionnement du roi retourne un cas d'échec, la fonction restreint les mouvements du roi en enlevant la position simulée de la liste des mouvements possibles.

Voici le résultat lors d'un scénario d'attaque du fou bleu, les mouvements du roi sont possibles sans sa mise en danger.

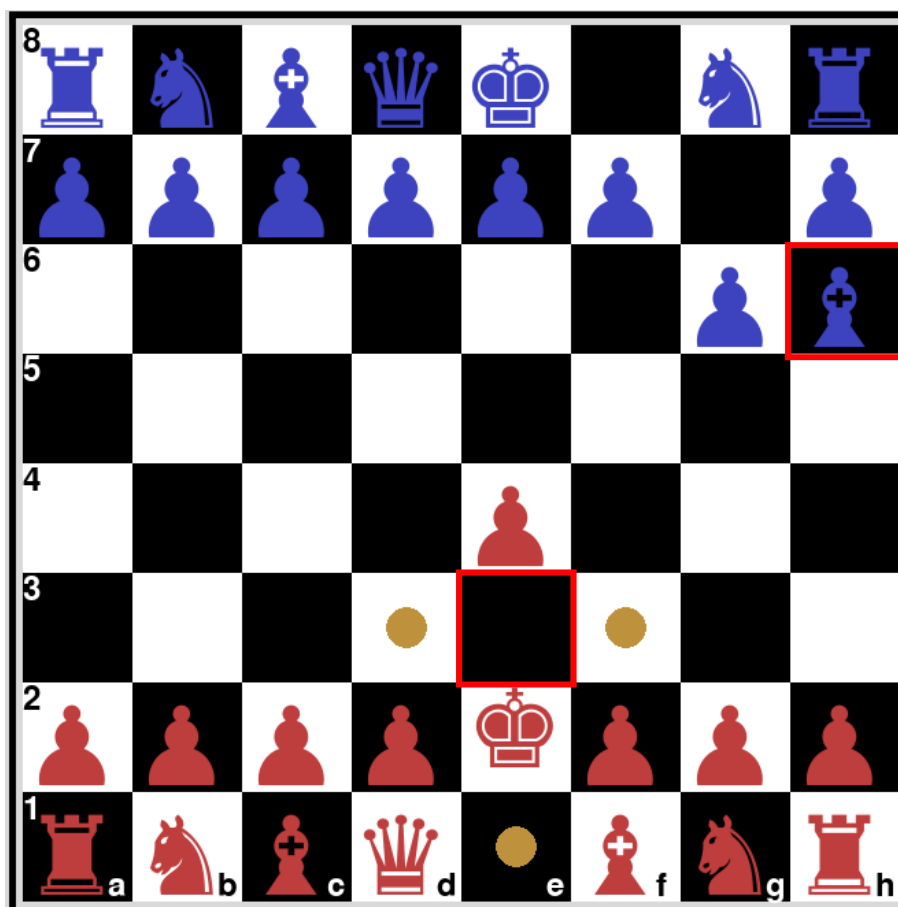


Figure 90 Résultats restriction mouvement du roi

#### 5.7.4 Protection et capture

Grâce à la variable « check » de chaque pièce, le programme sait si l'un des rois est en échec. Selon les règles de la FIDE, le joueur ne peut pas laisser son propre roi en échec, sa protection est donc nécessaire pour ne pas perdre la partie.

Afin de sortir le roi de cette situation, chaque méthode de mouvement des pièces possède une restriction.

La condition pour restreindre les mouvements vérifie si le roi est effectivement en échec ou si la simulation du prochain coup aboutisse à un cas d'échec du roi.

Une fois la condition vérifiée, l'algorithme compare les mouvements possibles, en cas normal, avec les positions enregistrées dans « checkPos ».

```
### If the possible movement of the piece equals the position to protect the king
possibleMoves= [[0] * COL for i in range(ROW)]
for a in range(len(board.checkPos)):
    if self.possibleMoves[board.checkPos[a][1]][board.checkPos[a][0]]!=0:
        possibleMoves[board.checkPos[a][1]][board.checkPos[a][0]]=True
### Previous position list equals new position list
self.possibleMoves=possibleMoves
```

Figure 91 Restriction mouvement en cas échec

Dans ce scénario le fou bleu met en échec le roi rouge.

Ici, aucune pièce ne peut se déplacer sauf le pion. Cependant, même si le pion n'a pas encore bougé, ses mouvements sont restreints à une seule case pour protéger le roi

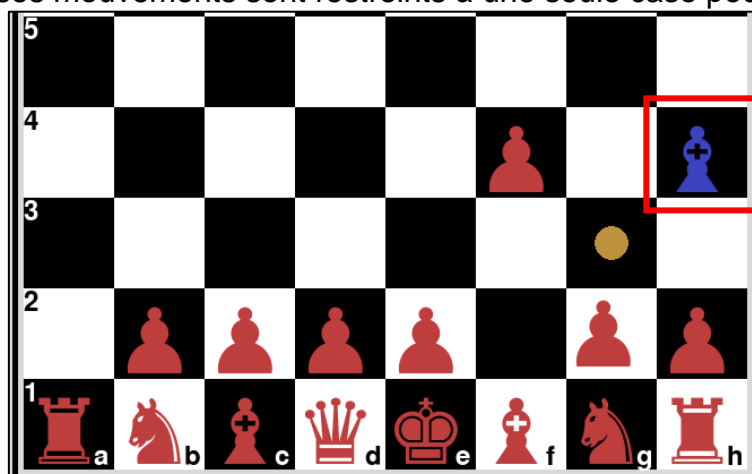


Figure 92 Résultats protection roi échec

Dans ce second scénario, si la tour rouge se déplace à droite ou à gauche, elle met inévitablement son roi en échec.

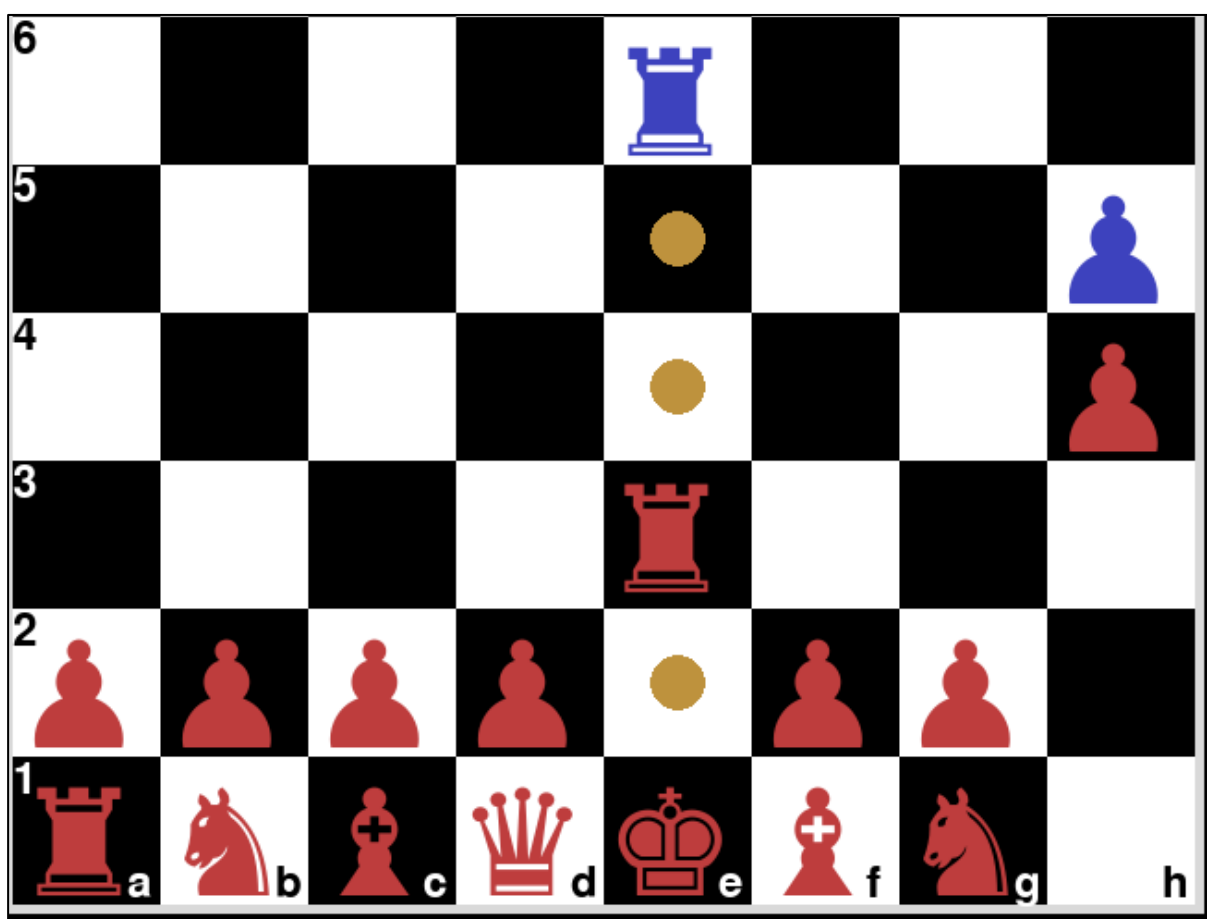


Figure 93 Résultat simulation échec

Ses mouvements sont restreints afin de pouvoir se déplacer tout en protégeant son roi.

## 5.8 Échec et mat

Une fonction permettant de vérifier si le joueur est en échec et mat est appelée dans la boucle principale du jeu.

```
while run:
    self.window.fill(GREY)
    board.DrawBorder(self.window)
    for event in pygame.event.get(): ...
    if self.CheckCheckMat(board,players):
        self.GameOn=False
```

Figure 94 Boucle principale "CheckCheckMate"

Cette fonction va parcourir toutes les pièces de la même couleur du joueur et vérifier si les pièces ont des mouvements disponibles. Si c'est le cas, le programme continue.

```
def CheckCheckMat(self,board,players):
    kingCol=0
    kingRow=0
    if players[0].playing:
        player=players[0]
    if players[1].playing:
        player=players[1]
    for row in range(ROW):
        for col in range(COL):
            if board.piecesPos[col][row]!=0 and board.piecesPos[col][row].color==player.color:
                for rowP in range(ROW):
                    for colP in range(COL):
                        if board.piecesPos[col][row].possibleMoves[colP][rowP]!=0:
                            return False
            if board.piecesPos[col][row].name=="king":
                kingCol=col
                kingRow=row
```

Figure 95 Fonction "CheckCheckMate" retour faux

Mais si au contraire les pièces n'ont plus de possibilité de mouvements et que le roi est en échec, le jeu s'arrête et affiche le gagnant.



Une dernière fonctionnalité est présente dans la fonction : elle vérifie si le roi n'est pas en échec dans ce cas le jeu le finit sur un match nul :

```
def CheckCheckMate(self, board, players):  
    kingCol=0  
    kingRow=0  
    if players[0].playing: ...  
    if players[1].playing: ...  
    for row in range(ROW): ...  
  
    if board.piecesPos[kingCol][kingRow].check:  
        if players[0].playing:  
            players[1].winning=True  
        if players[1].playing:  
            players[0].winning=True  
    else:  
        for obj in players:  
            obj.draw=True  
    return True
```

Figure 96 Fonction "CheckCheckMate retour vrai

Pour l'affichage du cas gagnant ou du match nul, une condition est ajoutée dans la méthode affichant toute la partie. Cette condition vérifie si l'un des cas est vrai, et affiche la réponse appropriée si c'est le cas :

```
###Draw players  
for obj in players:  
    obj.Draw(window)  
    if obj.winning:  
        obj.DrawWinner(window, board)  
    if obj.draw:  
        obj.DrawDraw(window, board)
```

Figure 97 Afficher le gagnant ou match nul

Dans les figures présentées ci-dessous, les méthodes "DrawWinner" et "DrawDraw" sont affichées.

```
def DrawWinner(self,window,board):  
    font = pygame.font.Font(None, 42)  
    text=font.render("{0} : à gagné".format(self.name), True, BLACK)  
    rect=pygame.Rect(board.size/2+board.x-(text.get_width()/2),board.size/2+board.y-80,300,80)  
    texte_rect = text.get_rect(center=rect.center)  
    pygame.draw.rect(window, WHITE, rect)  
    pygame.draw.rect(window, self.color, rect,5)  
    window.blit(text, texte_rect)
```

Figure 98 Fonction "DrawWinner"

```
def DrawDraw(self,window,board):  
    font = pygame.font.Font(None, 42)  
    text=font.render("Match nul", True, BLACK)  
    rect=pygame.Rect(board.size/2+board.x-(text.get_width()/2),board.size/2+board.y-80,300,80)  
    texte_rect = text.get_rect(center=rect.center)  
    pygame.draw.rect(window, WHITE, rect)  
    pygame.draw.rect(window, self.color, rect,5)  
    window.blit(text, texte_rect)
```

Figure 99 Fonction "DrawDraw"

Ces deux méthodes permettent d'afficher un rectangle contenant le texte approprié. En cas d'échec et mat, voici le résultat du jeu.



Figure 100 Gagnant échec et mat

## 5.9 Sauvegarde de parties

Le programme propose une fonctionnalité permettant d'enregistrer une partie en cours pour la finir ou pour même analyser les mouvements ultérieurement.

Afin d'accomplir cela, deux fonctions dans la classe « Game » sont implémentées : « saveGame » et « loadGame ».

La première permet d'enregistrer les mouvements joués de la partie, les positions de chaque pièce avec leurs identifiants et le tour du joueur dans un fichier JSON\*.

```
def SaveGame(self, board, players):
    piecesData = []
    if players[0].playing:
        player=players[0].name
    else:
        player=players[1].name
    for row in range(ROW):
        for col in range(COL):
            if board.piecesPos[col][row]!=0:
                pieceData = {
                    'id': board.piecesPos[col][row].id,
                    'col': board.piecesPos[col][row].col,
                    'row': board.piecesPos[col][row].row,
                    'player':player,
                    'AllMovement':board.allMovement
                }
                piecesData.append(pieceData)
    filePath = 'pieces.json'
    # Open a JSON file with "w" to write the list of all piece information into it
    with open(filePath, 'w') as file:
        json.dump(piecesData, file)
```

Figure 101 fonction "SaveGame"

Pour l'ouverture et l'écriture du fichier, la méthode « json.Dump() » importée de la bibliothèque « JSON » est utilisée.

---

\*Voir Glossaire

## 5.10 Charger une partie sauvegardée

Pour charger et lire les informations de la partie précédente enregistrées dans le fichier JSON, une autre méthode de la bibliothèque « JSON » est utilisée : « `json.load()` » Cette méthode est appelée dans la fonction « `loadGame` », celle-ci permet d'enregistrer les informations sur le fichier dans une liste.

```
def LoadGame(self,board,players):
    filePath = 'pieces.json'

    # Open the JSON file with "r" to read the list of all piece information
    with open(filePath, 'r') as file:
        piecesData = json.load(file)

    pieces = []

    for pieceData in piecesData:
        id = pieceData['id']
        col = pieceData['col']
        row = pieceData['row']
        player = pieceData['player']
        AllMovement= pieceData['AllMovement']
        pieces.append((id,col,row,player,AllMovement))
```

Figure 102 Fonction "LoadGame"

Toujours dans la même fonction, cette liste est ensuite parcourue avant le début du jeu, pour changer les positions des pièces et des pions sur le plateau de jeu. Grâce aux identifiants, la colonne et ligne de la pièce sont modifiées pour correspondre aux colonnes et lignes de la partie précédente.

```
for row in range(ROW):
    for col in range(COL):
        if board.piecesPos[col][row]!=0:
            # Run through the list with the pieces information
            for i in range(len(pieces)):
                if board.piecesPos[col][row].id==pieces[i][0]:
                    if board.piecesPos[col][row].col!=pieces[i][1] or
                    board.piecesPos[col][row].row!=pieces[i][2]:
                        board.piecesPos[col][row].col=pieces[i][1]
                        board.piecesPos[col][row].row=pieces[i][2]
                        board.piecesPos[col][row].firstMove=True
                        tempPiece = board.piecesPos[col][row]
                        board.piecesPos[pieces[i][1]][pieces[i][2]]=tempPiece
                        board.piecesPos[col][row]=0
            break
```

Figure 103 Charger les positions des pièces

La liste contenant toutes les positions est modifiée par celle de la partie précédente.

```
#Get all movement of the JSON file
board.allMovement=AllMovement
```

Figure 104 Charger tous les mouvements

Finalement, la valeur du quadrillage, vide ou non, est modifiée avant de changer le tour du joueur.

```
for row in range(ROW):
    for col in range(COL):
        if board.piecesPos[col][row]!=0:
            board.squares[col][row].empty=False
        else:
            board.squares[col][row].empty=True

#Change the player playing
for obj in players:
    if obj.name==pieces[0][3]:
        obj.playing=True
    else:
        obj.playing=False
```

Figure 105 Charger le joueur

### 5.11 Affichage du dernier mouvement

Une des fonctions du jeu est de pouvoir afficher de manière claire le dernier coup joué. Cela peut être pratique lors d'une inattention d'un joueur durant le jeu.

Pour ce faire, lors d'un nouveau positionnement d'une pièce dans la fonction « clicked » de la classe « Piece », la colonne et la ligne de l'ancienne position de la pièce, ainsi que sa nouvelle colonne et ligne sont enregistrées dans une liste, ici représentée par « allMovement ».

```
if not self.clicked:
    for row in range(ROW):
        for col in range(COL):
            # Check on wich square is the position of the piece
            if board.squares[col][row].rect.collidepoint(posMouse) :
                #if the move is possible
                if self.possibleMoves[col][row]:
                    ### Change piece information
                    board.allMovement.append((self.col,self.row,col,row))
```

Figure 106 Enregistrement des mouvements

Cette liste est utilisée dans la fonction « showLastMovement ». Celle-ci parcourt les derniers éléments ajoutés dans la liste et change l'état du jeu actuel pour afficher le dernier mouvement en remplaçant les pièces sur le quadrillage, selon leurs positions du tour d'avant.

```
self.piecesPos[self.allMovement[oldCol][self.allMovement[oldRow]]] =
self.piecesPos[self.allMovement[newCol][self.allMovement[newRow]]]
```

Puis supprimer de la liste des pièces la nouvelle position temporairement.

```
self.piecesPos[self.allMovement[newCol][self.allMovement[newRow]]=0
```

La fonction permet aussi d'afficher les pièces même si elles étaient déjà prises, donc hors du jeu, grâce à la liste « piecesDie ». Cette liste est aussi parcourue en vérifiant si la colonne et ligne de la dernière pièce déplacée sont égales à la colonne et ligne du dernier élément dans la liste des pièces prises.

« showLastMovement » est appelée dans la boucle principale du jeu, lorsque le bouton d'affichage du dernier élément est sélectionné. Le jeu revient à la normale quand le bouton a été enfoncé à nouveau.

```
elif showLastMovementButton.rect.collidepoint(posMouse) :  
    showLastMovementButton.Clicked()  
    if len(board.allMovement)>0:  
        board.showLastMovement= not board.showLastMovement  
        board.ShowLastMovement()
```

Figure 107 Bouton "showLastMovement"

Pour un affichage plus clair du dernier mouvement, une fonctionnalité est ajoutée dans la boucle affichant tout le jeu.

Si le bouton de l'affichage du dernier bouton est appuyé, deux rectangles en or s'affichent autour des pièces

```
def Draw(self,board,window,players,showAllMovement,buttons):
    ###Draw last movement
    if board.showLastMovement:
        pygame.draw.rect
        pygame.draw.rect
```

Figure 108 Afficher les derniers mouvements

Ci-dessous ce trouve le résultat du chapitre.

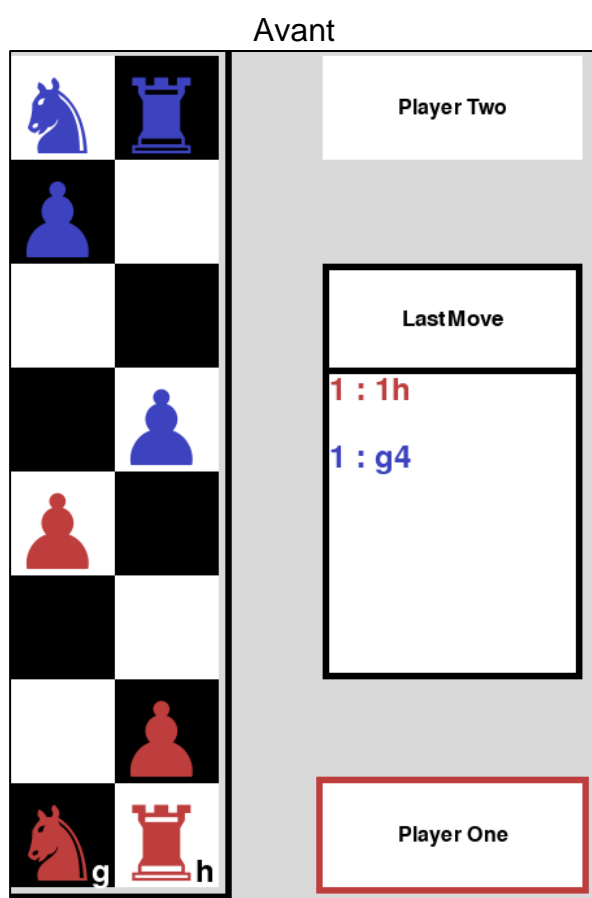


Figure 109 Résultats dernier mouvement avant

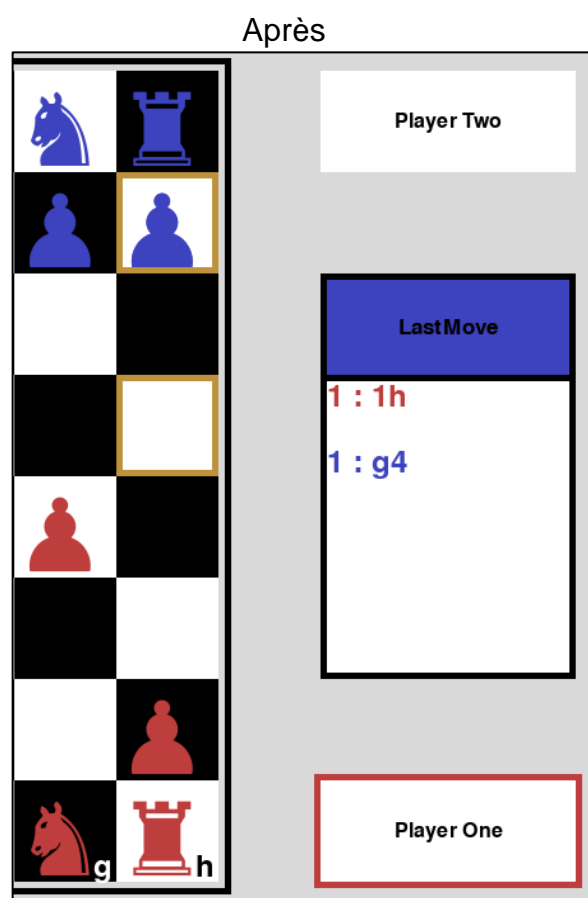


Figure 110 Résultats dernier mouvement après

## 5.12 Feedback utilisateur

Pour un meilleur développement, contenu et expérience utilisateur, un formulaire « forms » avec des questions ciblées sur le programme est développé et transmet à des bêta-testeurs.

- Avez-vous eu des problèmes lors du lancement de jeu ? Si oui, lesquels ? \*
- Avez-vous eu des problèmes lors d'une partie de jeu ? Si oui, lesquels ? \*
- Le programme vous semble-t-il ralentir à certains moments ou ne pas réagir lors de click ? Si oui, veuillez décrire ces moments. \*
- Avez-vous eu des problèmes avec le déplacement des pièces ? Si oui lesquels ? \*
- L'application vous semble-t-elle intuitive ? \*
- Comment trouvez-vous le design, aspect générale de l'application ? \*
- Avez-vous d'éventuelles suggestions pour des améliorations futur ?
- Quelle note donnez-vous à ce jeu ? \*
- Pourquoi avez-vous donné cette note ? \*

Les résultats du formulaire sont ensuite étudiés et des perfectionnements sont développés en voici quelques exemples :




Figure 111 Feedback plateau de jeu

La fonction « Draw » dans la boucle principale du jeu pour l'affichage est modifiée pour faire appel aux nouvelles fonctions implémentées dans la classe du quadrillage « Square » pour une meilleure perception du plateau et son environnement.

```
def Draw(self,board,window,players,showAllMovement,buttons):  
  
    ###Draw board  
    for row in range(ROW):  
        for col in range(COL):  
            board.squares[col][row].Draw(window)  
            if col==0:  
                board.squares[col][row].DrawNumber(window,ROW-row)  
            if row ==ROW-1:  
                board.squares[col][row].DrawAlphabet(window,col)
```

Figure 112 Fonction "draw" après Feedback

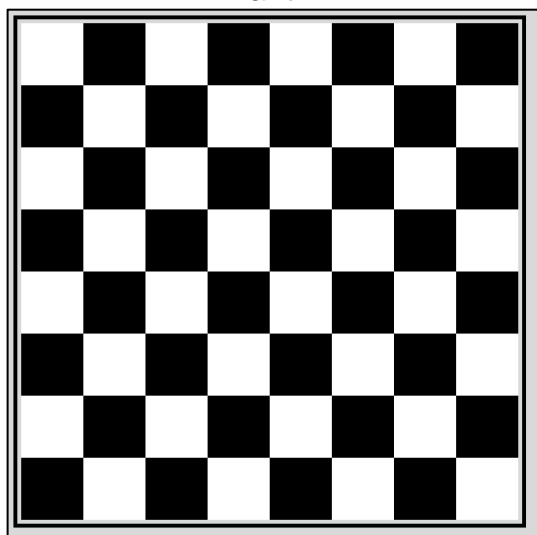
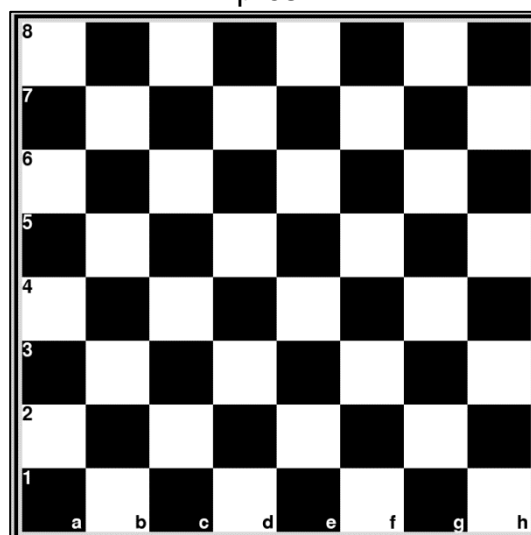
Les nouvelles fonctions dans cette classe permettent d'ajouter dans chaque coin des carrés de la première colonne et dernière ligne, des chiffres et des lettres



```
def DrawAlphabet(self,window,text):  
    if self.color==BLACK:  
        color=WHITE  
    else:  
        color=BLACK  
  
    alphabet= self.font.render("{0}".format(chr(text + 97)), True, color)  
    window.blit( alphabet, (self.rect.bottomright[0]-18,self.rect.bottomright[1]-24))
```

*Figure 113 Fonction "DrawNumber"*

Les deux fonctions utilisent respectivement les colonne et ligne pour leurs numéros. Pour l'affichage des lettres de l'alphabet, le programme fait appel à la méthode « chr() » intégrée en Python qui prend un code unicode en tant qu'argument et renvoie le caractère correspondant « +97 » permet de sortir la première lettre de l'alphabet. Ci-dessous les images du résultat des modifications :

*Avant**Figure 114 Avant feedback**Après**Figure 115 Après feedback*

Voici un autre exemple :

Petit de soucis de déplacement du roi (en étant en échec)

Figure 116 Feedback mouvement roi en échec

Le problème est dû à la simulation des mouvements possibles du roi. Pour régler le problème, il suffit de changer la variable « chech » du roi simulé en « faux ».

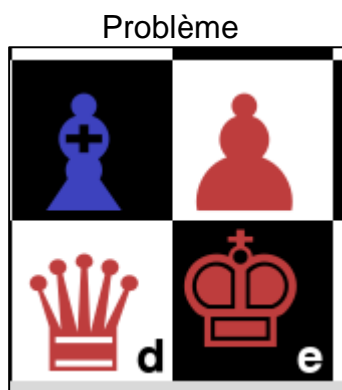


Figure 117 Feedback problème mouvement du roi



Figure 118 Feedback solution mouvement du roi

Une fois les feedback reçus, le questionnaire recueille les réponses et génère un rapport récapitulatif :

Question 1 du formulaire :

Avez-vous eu des problèmes lors du lancement de jeu ? Si oui lesquels ?

5 réponses

Non

non

Non.

Tip top

Figure 119 Feedback question 1

## Question 2 du formulaire :

Avez-vous eu des problèmes lors d'une parti de jeu ? Si oui lesquels ?

5 réponses

Non

Non.

Après être mis en échec, certains déplacements du roi devenaient impossibles malgré le fait qu'ils devraient l'être

Figure 120 Feedback question 2

## Question 3 du formulaire :

Le programme vous semble-t-il ralentir à certain moment ou ne pas réagir lors de click ? Si oui, veuillez décrire ces moments

5 réponses

Non

non

Non.

Figure 121 Feedback question 3

## Question 4 du formulaire :

Avez-vous eu des problèmes avec le déplacement des pièces ? Si oui lesquels ?

5 réponses

Non

non

Non.

Petit de soucis de déplacement du roi (en étant en échec)

Figure 122 Feedback question 4

## Question 5 du formulaire :

L'application vous semble-t-elle intuitive ?

5 réponses

Oui, rajouter une page de règles de jeu

oui

Oui.

oui magnifique

Oui

Figure 123 Feedback question 5

## Question 6 du formulaire :

Comment trouvez-vous le design de l'application ?

5 réponses

bien, il manque les chiffres et lettres sur le plateau

simple mais bon

Bien, simple et compréhensif.

magnifique

Bien

Figure 124 Feedback question 6

## Question 7 du formulaire :

Avez-vous d'éventuelles suggestions pour des améliorations futur ?

5 réponses

en-passant, promotions, castles

Ajouts d'une horloge ?

non

Un mode en ligne

Une page de description

Figure 125 Feedback question 7

## Question 8 du formulaire :

Quelle note donnez-vous à ce jeu ?

5 réponses

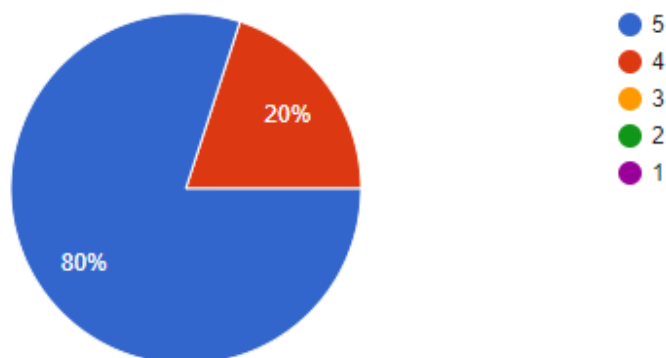


Figure 126 Feedback question 8

## Question 9 du formulaire :

Pourquoi avez-vous donné cette note ?

5 réponses

plusieurs éléments présents dans un jeu d'échecs normal manquent

D'après le descriptifs des fonctionnalités du jeu, je trouve que le jeu les respecte.

Le jeu fonctionne bien

magnifique fonctionnel malgré petit souci de déplacement mais sinon parfait magnifique

Je n'ai pas eu de problèmes et l'aide visuelle pour les déplacements est très pratique

*Figure 127 Feedback question 9*

Ces résultats mettent en évidence le fait que le graphisme du jeu est à la fois efficace et simple, sans rencontrer de problèmes majeurs lors du lancement ou des performances du jeu. Quelques problèmes mineurs liés aux déplacements ont été identifiés, mais ils ont été résolus.



La classe pour les tests unitaires est implémentée. Ensuite, sont déclarées les positions du roi et de la tour, puis les résultats attendus et les résultats renvoyer par la fonction, finalement la fonction permettant de faire le test est appelé.

```
class TestGetPositionsBetween(unittest.TestCase):
    def testGetPositionsBetweenColumn(self):
        kingCol=4
        kingRow =2
        rookCol=4
        rookRow=7

        expectedPositions = [(3, 4), (4, 4), (5, 4),(6,4)]
        positions = getPositionsBetween(kingRow, kingCol, rookRow, rookCol)
        self.assertEqual(positions, expectedPositions)
```

Figure 130 Test unitaire colonne

Le deuxième cas vérifié par le test unitaire, est le renvoi des bonnes positions, toujours entre la tour et le roi adverse en cas d'échec, mais cette fois-ci ce sont les positions des lignes qui sont les mêmes.

```
def testGetPositionsBetweenRow(self):
    kingCol=5
    kingRow = 4
    rookCol=7
    rookRow = 4
    expectedPositions = [ (4, 6)]
    positions = getPositionsBetween(kingRow, kingCol, rookRow, rookCol)
    self.assertEqual(positions, expectedPositions)
```

Figure 131 Test unitaire ligne

Le troisième cas vérifie le bon fonctionnement de la fonction en testant une situation où il n'y a pas de solution utile dans cette méthode.

Le positionnement du roi et de la tour est implémenté mais ils ne se trouvent si sur la même ligne, ni sur la même colonne.

```
def testGetPositionsBetweenNothing(self):
    kingCol= 1
    kingRow= 1
    rookCol= 2
    rookRow = 3
    expectedPositions = []
    positions = getPositionsBetween(kingRow, kingCol, rookRow, rookCol)
    self.assertEqual(positions, expectedPositions)
```

Figure 132 Test unitaire rien

La classe se finit par l'exécution des tests unitaires avec la condition suivante.



```
if __name__ == '__main__':  
    unittest.main()
```

Figure 133 Test unitaire lancement

Une fois le fichier, contenant les tests, exécuté, le terminal renvoie le résultat avec les erreurs, si existant :

```
D:\TPI\ChessPy>py unitTest.py  
...  
-----  
Ran 3 tests in 0.000s  
OK
```

Figure 134 Test unitaire résultats

### 5.14 Résultats des tests effectués

Nom de test	Scénario	Résultats attendu	Résultats obtenue
Affichage du plateau	A l'ouverture du programme, un plateau d'échec s'affiche	Une fois le jeu lancé, un plateau d'échec constitué de plusieurs carrés (noirs et blancs) s'affiche de manière régulière.	Ok
Affichage des pièces	A l'ouverture du programme, les pièces et pions s'affichent sur le plateau	Une fois le plateau complètement affiché, les pions et pièces apparaissent à leurs places respectives sur le plateau de jeu. Chaque pièce a sa propre forme, chaque joueur a une couleur de pièces distincte.	Ok
Déplacements légaux des pièces	Les pièces une fois en mouvement ont une restriction sur leurs mouvements	Chaque pièce se déplace seulement selon leurs mouvements respectifs désignés par la FIDE.	Ok
Affichage des déplacements légaux des pièces	Pour faciliter l'utilisations du jeu, donc les mouvements possibles de chaque pièce, un affichage des déplacements est présent	Une fois un pion ou pièce sélectionné par l'utilisateur, un affichage instinctif est présent sur le plateau de jeu.	Ok
Mouvements des pièces (utilisateur)	L'utilisateur du programme peut déplacer ses pièces	Une pièce peut être sélectionnée à l'aide de la souris et sa position est instantanément changée pour devenir la même que la position de la souris.	Ok
Nouvelle position (utilisateur)	L'utilisateur du programme a la possibilité de changer ses pièces d'échec de position sur le plateau	Une fois la pièce sélectionnée et déplacer l'utilisateur utilise le click de la souris pour confirmer la nouvelle position de la pièce (dans la limite des mouvement possibles)	Ok

Prise de pièce	Si les pions ont la même position sur le plateau, une des pièces a été perdue	La nouvelle pièce s'affiche correctement sur le plateau, elle garde ses mouvements, la pièce prise est affichée sous le nom de son propriétaire	Ok
Enregistrement des coups	Lors d'un déplacement de pièce ou pion, celui-ci est enregistré.	A côté du plateau de jeu, une liste avec le dernier déplacement est disponible	Ok
Affichage du dernier mouvement	Un bouton est disponible pour afficher le dernier mouvement de pièce effectué	L'affichage clair du dernier déplacement est disponible et aucun joueur ne peut déplacer leurs pions durant ce temps	Ok
Situation d'échec	Des restrictions de mouvement sont implémentées	Pour empêcher le joueur de mettre son roi en échec, une restriction de mouvement des pièces est affichée, si le roi est déjà en échec, les pions pourront se déplacer seulement pour le protéger.	Ok
Situation d'échec et mat	Un joueur ne peut plus protéger son roi	Tout mouvement est bloqué et une fenêtre affiche le nom du gagnant	Ok

## 5.15 Liste des documents fournis

Nom du document	Version
Planification initiale	1.0
Rapport de projet	8.0
Journal de travail	28.0
Code source	1.0

## 6 Conclusions

Ce chapitre permet de conclure le projet avec un bilan critique des objectifs attendus lors de ce projet, un bilan sur le temps consacré sur ce projet et un bilan personnel du candidat sur son ressenti face au projet et à sa réalisation.

### 6.1 Bilan de la planification

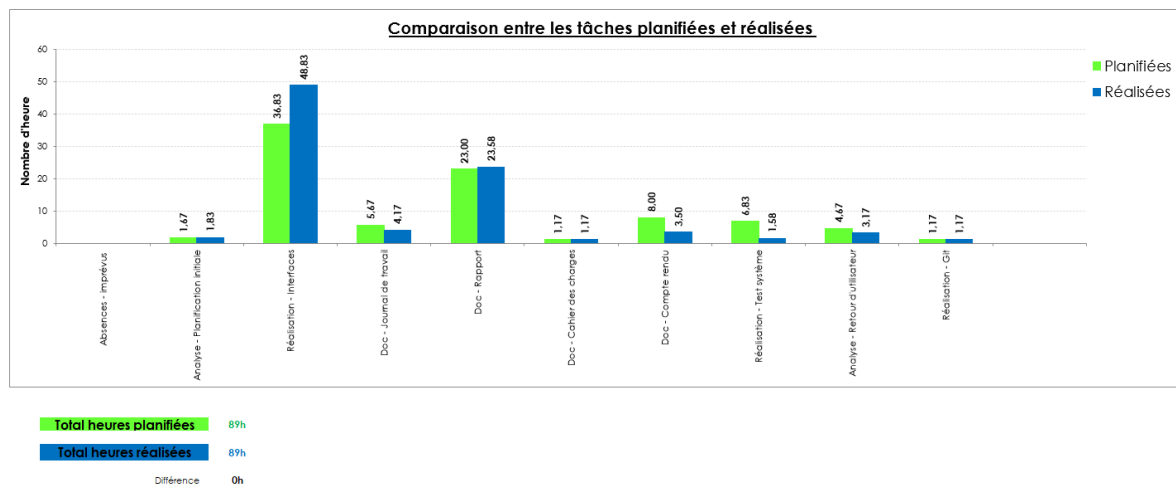


Figure 135 Comparaison entre les tâches planifiées et réalisées

Le graphique ci-dessus compare le temps en heure des tâches planifiées et réalisées. Une différence est visible entre la réalisation de l'interface et test unitaire ainsi que du compte rendu.

L'explication du pour le compte rendu tient au fait que les interactions entre les experts et moi-même ont été moins importantes que prévu, ce qui a permis plus de temps pour la réalisation.

Pour la réalisation de l'interface et des tests unitaires, ces tests ont été réalisés plus rapidement que ce que prévoyait la planification grâce à l'utilisation de Pygame et des bibliothèques Python. Ces bibliothèques offrent des fonctions qui facilitent la réalisation des tests. Ce temps en plus m'a permis d'améliorer, d'optimiser et d'ajouter des fonctionnalités en plus dans le programme réalisé.

## 6.2 Bilan des objectifs

Les objectifs fixés par le chef de projet ainsi que les exigences du cahier des charges ont été quasiment tous réalisés et testés, comme expliqué dans ce rapport. Ces objectifs implémentés dans le programme fonctionnent parfaitement sans avoir besoin de retouche. La seule fonctionnalité ne respectant pas complètement le cahier des charges est la consultation de l'historique des mouvements, seule la consultation des mouvements du dernier déplacement d'une pièce est possible.

L'option pour jouer contre un ordinateur a été réfléchi mais pas implémentée dans le programme, expliqué par le choix de se concentrer sur les objectifs évalués dans ce projet.

## 6.3 Bilan personnel

Dans l'ensemble, le projet a représenté un défi intéressant à relever. L'aspect le plus complexe selon moi a été les cas d'échec et de faire en sorte de restreindre les mouvements des pièces lors de ce cas, car il faut prendre en compte le tour d'après et donc simuler tous les mouvements possibles. C'est beaucoup de conditions à prendre en compte tout en essayant d'avoir un code le plus optimisé possible.

Un autre point difficile a été l'utilisation des « fonts » : la difficulté se trouvait dans la sérialisation\* d'une liste d'objets pour les enregistrer dans un fichier JSON, car la méthode permettant cet enregistrement de liste d'objets n'accepte pas les fonts. Pour la simulation des mouvements que j'ai réalisée grâce à une copie du plateau de jeu, la copie profonde de la liste d'objets était impossible avec les fonts. Mais j'ai toujours réussi à trouver une solution et je suis content de toutes les fonctionnalités implémentées. En revanche, je ne dissimule pas la frustration de n'avoir pas pu implémenter une possibilité de pouvoir jouer contre un ordinateur. C'est un aspect d'amélioration que mon programme peut recevoir, en plus bien sûr de toutes les autres règles existant dans un jeu d'échecs, telles que la possibilité de roque, la prise en passant et les différentes positions d'égalité.

---

\*Voir Glossaire

## 7 Annexes

Ce chapitre présente des informations complémentaires relatives à la planification, au journal de travail et aux sources.

### 7.1 Résumé du rapport du TPI / version succincte de la documentation

Deux utilisateurs peuvent choisir sur le même ordinateur, dans un menu, de choisir entre charger la partie précédente ou lancer une nouvelle partie. Un plateau d'échec s'affiche avec les pièces respectives. Chaque utilisateur a la possibilité, à tour de rôle, de déplacer un pion en utilisant une aide visuelle qui affiche les mouvements légaux correspondants à chaque pièce. Le cas d'échec, avec la restriction de mouvement et le cas d'échec et mat sont implémentés, en respectant les règles de la FIDE. L'objectif de ce projet est de créer un jeu d'échecs qui est testé par des bêta-testeurs, avant d'étudier leurs réponses et de prendre en compte les améliorations suggérées.

J'ai tout d'abord procédé à une analyse préliminaire, suivi d'une analyse détaillée du projet. Cela m'a permis de ressortir les maquettes graphiques qui me permettent de construire l'application. À la suite de cela, j'ai commencé par implémenter et afficher un plateau d'échec vide avant de m'occuper de la création et de l'affichage des pièces et pions avec une classe respective. Pour le design des pièces et pions, des caractères de la liste des caractères « DejaVuSans » sont utilisés. Vient ensuite l'implémentation des mouvements et coups légaux de chaque pièce en suivant les règles de la FIDE. Puis, une fonctionnalité enregistrant tous les mouvements et affichant le dernier mouvement grâce à une aide visuelle est réalisée. Finalement, les cas d'échec et d'échec et mat sont créés et vérifiés à l'aide d'une simulation des coups possibles lors du prochain tour. La fin du projet est marquée par la création d'un formulaire pour des bêta-testeurs et par l'analyse des retours une fois ceux-ci reçus pour identifier de possibles améliorations.

Sur toutes les fonctionnalités demandées, seule la consultation des mouvements effectués est réduite à la consultation du dernier mouvement. Le graphisme du programme est considéré comme simple mais efficace par les testeurs. La note générale du jeu est globalement positive, avec une demande d'ajout des règles qui n'étaient pas initialement prévues dans le cadre de ce projet, telles que le roque et la prise en passant.

## 7.2 Glossaire

**Bêta-testeur** : Un bêta-testeur est une personne chargée de tester une version préliminaire ou en développement d'un logiciel, d'une application ou d'un jeu avant sa sortie officielle.

**Commits** : Un commit fait référence à l'action d'enregistrer les modifications effectuées sur un ensemble de fichiers source d'un projet dans un système de contrôle de version.

**FIDE** : La FIDE (Fédération Internationale des Échecs) est l'organisme international chargé de régir et de promouvoir le jeu d'échecs à l'échelle mondiale.

**JSON** : (JavaScript Object Notation) est un format de données léger et largement utilisé pour l'échange de données structurées entre différents systèmes. Il est basé sur une syntaxe simple et lisible par les humains

**Sérialisation** : La sérialisation est le processus de conversion d'un objet ou d'une structure de données en une séquence d'octets, qui peut ensuite être stockée, transférée ou utilisée ultérieurement. L'objectif de la sérialisation est de représenter les données de manière portable et indépendante du langage de programmation ou de la plate-forme utilisée.

**Sprite** : Un sprite est une image ou une animation graphique utilisée dans les jeux vidéo et les applications interactives. Il représente généralement un personnage, un objet ou un élément visuel du jeu.

### 7.3 Sources – Webographie

*Font pour échec*, dernière consultation le 12.05.2023

<https://dejavu-fonts.github.io/>

*Création d'un fichier gitignore*, GitGuardian, dernière consultation le 15.05.2023

<https://www.youtube.com/watch?v=3KrjEytC5qc>

*Gestion des événements PyGame*, dernière consultation le 15.05.2023

<https://www.pygame.org/docs/ref/event.html>

*Gestion des rects PyGame*, dernière consultation le 26.05.2023

<https://www.pygame.org/docs/ref/rect.html>

*Comment afficher les FPS avec PyGame*, CodersLegacy, dernière consultation le 15.05.2023

<https://coderslegacy.com/python/display-fps-pygame/>

*JSON : exemple complet*, CodersLegacy, JDN, dernière consultation le 23.05.2023

<https://www.journaldunet.com/web-tech/developpeur/1055681-le-format-json-ajax-et-jquery/1055683-json-exemple-complet>

*Python File Write*, CodersLegacy, W3Schools, dernière consultation le 23.05.2023

[https://www.w3schools.com/python/python\\_file\\_write.asp](https://www.w3schools.com/python/python_file_write.asp)

*Python File Open*, W3Schools, dernière consultation le 23.05.2023

[https://www.w3schools.com/python/python\\_file\\_open.asp](https://www.w3schools.com/python/python_file_open.asp)

*openai*, dernière consultation le 30.05.2023

<https://chat.openai.com/>

*SOLID (informatique)*, Wikipédia, dernière consultation le 23.05.2023

[https://fr.wikipedia.org/wiki/SOLID\\_\(informatique\)](https://fr.wikipedia.org/wiki/SOLID_(informatique))

*PRINCIPES SOLID SIMPLIFIÉS*, Tawfik Nouri, dernière consultation le 23.05.2023

<https://latavernedutesteur.fr/2020/04/28/principes-solid-simplifies-1-5-responsabilite-unique/>

*Tests unitaires avec le module Python unittest*, Hafeezul Kareem Shaik, dernière consultation le 30.05.2023

<https://geekflare.com/fr/unit-testing-with-python-unittest/>





## 7.4 Planification initiale

Séquence 1			Date: jeudi, 11 mai 2023	Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...
Doc - Journal de travail	8	0h40min	Création du journal de travail	
Analyse - Planification initiale	8	0h40min	Réalisation de la planification initiale	
Doc - Journal de travail	4	0h20min	Remplir la séquence 1 du journal	
Doc - Cahier des charges	14	1h10min	Visite expert 1 + lire + (modifier) + signer cahier des charges	
Doc - Compte rendu	4	0h20min	Envoi d'un mail avec la planification initiale aux experts et chef de projet	
Total tranche	38	3h10min		

Séquence 2			Date: vendredi, 12 mai 2023	Matin
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...
Analyse - Planification initiale	12	1h	Finition de la planification	
Réalisation - Interfaces	24	2h	Conception de l'application	
Doc - Rapport	12	1h	Réalisation d'un squelette du rapport	
Total tranche	48	4h		



Séquence 3			Date: vendredi, 12 mai 2023		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Doc - Compte rendu	4	0h20min	Envoie d'un mail avec la planification initial aux experts et chef de projet		
Réalisation - Git	14	1h10min	Realisation d'un repos git et liaison avec VSCode		
Doc - Rapport	12	1h	Réalisation d'un squelette du rapport		
Doc - Journal de travail	2	0h10min	Remplir la séquence 2+3 du journal		
Réalisation - Interfaces	6	0h30min	Conception de l'application		
Total tranche	38	3h10min			

Séquence 4			Date: lundi, 15 mai 2023		Matin
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Doc - Compte rendu	3	0h15min	Réponse aux mails si existants		
Réalisation - Interfaces	12	1h	Mise en place des fichiers et librairie à télécharger		
Réalisation - Interfaces	23	1h55min	Création du plateau de jeu pour l'application		
Total tranche	38	3h10min			

Séquence 5			Date: lundi, 15 mai 2023		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Doc - Journal de travail	4	0h20min	Remplir la séquence 4+5 du journal		
Réalisation - Interfaces	34	2h50min	Création de pièces pour l'application		
Total tranche	38	3h10min			



Séquence 6			Date: mardi, 16 mai 2023		Matin
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Réalisation - Interfaces	20	1h40min	Création de pions pour l'application		
Réalisation - Interfaces	18	1h30min	Affichage des éléments sur le plateau		
Total tranche	38	3h10min			

Séquence 7			Date: mardi, 16 mai 2023		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Doc - Compte rendu	4	0h20min	Envoie d'un mail avec le journal de travail et rapport aux experts et chef de projet		
Doc - Journal de travail	4	0h20min	Remplir la séquence 6+7 du journal		
Réalisation - Interfaces	12	1h	Affichage des éléments sur le plateau		
Doc - Rapport	18	1h30min	Mise à jour du rapport		
Total tranche	38	3h10min			

Séquence 8			Date: mercredi, 17 mai 2023		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Doc - Journal de travail	4	0h20min	Remplir la séquence 8 du journal		
Doc - Compte rendu	3	0h15min	Réponse aux mails si existants		
Réalisation - Interfaces	12	1h	Affichage des éléments sur le plateau		
Réalisation - Interfaces	9	0h45min	Mouvement des pièces sur le plateau		
Total tranche	28	2h20min			



Séquence 9			Date: lundi, 22 mai 2023		Matin
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Doc - Compte rendu	3	0h15min	Réponse aux mails si existants		
Réalisation - Interfaces	12	1h	Mouvement des pièces sur le plateau		
Réalisation - Interfaces	23	1h55min	Conditions de prises de pièce		
Total tranche	38	3h10min			

Séquence 10			Date: lundi, 22 mai 2023		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Doc - Compte rendu	4	0h20min	Envoie d'un mail avec le journal de travail et rapport aux experts et chef de projet		
Doc - Journal de travail	4	0h20min	Remplir la séquence 9+10 du journal		
Réalisation - Interfaces	18	1h30min	Conditions de prises de pièce		
Réalisation - Interfaces	12	1h	Possibilité de jouer à 2		
Total tranche	38	3h10min			

Séquence 11			Date: mardi, 23 mai 2023		Matin
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Réalisation - Interfaces	20	1h40min	Possibilité de jouer à 2		
Réalisation - Interfaces	18	1h30min	Enregistrement des coups joués		
Total tranche	38	3h10min			



Séquence 12			Date: mardi, 23 mai 2023		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Doc - Compte rendu	4	0h20min	Envoie d'un mail avec le journal de travail et rapport aux experts et chef de projet		
Doc - Journal de travail	4	0h20min	Remplir la séquence 11+12 du journal		
Réalisation - Interfaces	12	1h	Enregistrement des parties jouées		
Doc - Rapport	18	1h30min	Mise à jour du rapport		
Total tranche	38	3h10min			

Séquence 13			Date: mercredi, 24 mai 2023		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Doc - Journal de travail	4	0h20min	Remplir la séquence 13 du journal		
Doc - Compte rendu	3	0h15min	Réponse aux mails si existants		
Réalisation - Interfaces	11	0h55min	Création d'un menu pour l'application		
Réalisation - Interfaces	10	0h50min	Ajouts de commentaires manquants et corrections des variables aux conventions de l'ETML		
Total tranche	28	2h20min			



Séquence 14			Date: jeudi, 25 mai 2023		Matin
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Doc - Compte rendu	18	1h30min	Visite expert 2 + discussion sur l'avancement du projet		
Doc - Rapport	21	1h45min	Corrections des commentaires de M.Malherbe		
Réalisation - Interfaces	6	0h30min	Corrections des commentaires de M.Malherbe		
Doc - Compte rendu	3	0h15min	Réponse aux mails si existants		
Total tranche	48	4h			

Séquence 15			Date: jeudi, 25 mai 2023		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Doc - Journal de travail	4	0h20min	Remplir la séquence 14+15 du journal		
Réalisation - Interfaces	18	1h30min	Corrections des commentaires de M.Malherbe		
Réalisation - Test système	16	1h20min	Réalisation de tests unitaires		
Total tranche	38	3h10min			

Séquence 16			Date: vendredi, 26 mai 2023		Matin
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Réalisation - Test système	45	3h45min	Réalisation de tests unitaires		
Doc - Compte rendu	3	0h15min	Réponse aux mails si existants		
Total tranche	48	4h			



Séquence 17			Date: vendredi, 26 mai 2023		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Doc - Compte rendu	4	0h20min	Envoie d'un mail avec le journal de travail et rapport aux experts et chef de projet		
Doc - Journal de travail	4	0h20min	Remplir la séquence 16+17 du journal		
Doc - Rapport	30	2h30min	Mise à jour du rapport		
Total tranche	38	3h10min			

Séquence 18			Date: mardi, 30 mai 2023		Matin
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Doc - Compte rendu	3	0h15min	Réponse aux mails si existants		
Analyse - Retour d'utilisateur	35	2h55min	Création d'un document Forms pour les testeur de l'application		
Total tranche	38	3h10min			

Séquence 19			Date: mardi, 30 mai 2023		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Doc - Compte rendu	4	0h20min	Envoie d'un mail avec le journal de travail et rapport aux experts et chef de projet		
Doc - Journal de travail	4	0h20min	Remplir la séquence 18+19 du journal		
Doc - Rapport	30	2h30min	Mise à jour du rapport		
Total tranche	38	3h10min			



Séquence 20			Date: mercredi, 31 mai 2023		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Doc - Journal de travail	4	0h20min	Remplir la séquence 20 du journal		
Réalisation - Test système	21	1h45min	Test de performance de l'application		
Doc - Compte rendu	3	0h15min	Réponse aux mails si existants		
Total tranche	28	2h20min			

Séquence 21			Date: jeudi, 1 juin 2023		Matin
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Analyse - Retour d'utilisateur	21	1h45min	Analyse du retours et évaluations des testeur de l'application		
Réalisation - Interfaces	24	2h	Corrections des points retenue lors de l'analyse des retours de tests		
Doc - Compte rendu	3	0h15min	Réponse aux mails si existants		
Total tranche	48	4h			

Séquence 22			Date: jeudi, 1 juin 2023		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Réalisation - Interfaces	34	2h50min	Corrections des points retenue lors de l'analyse des retours de tests		
Doc - Journal de travail	4	0h20min	Remplir la séquence 21+22 du journal		
Total tranche	38	3h10min			





Séquence 23			Date: vendredi, 2 juin 2023		Matin
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Réalisation - Interfaces	27	2h15min	Corrections des points retenue lors de l'analyse des retours de tests		
Doc - Rapport	21	1h45min	Mise à jour du rapport		
Total tranche	48	4h			

Séquence 24			Date: vendredi, 2 juin 2023		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Doc - Compte rendu	4	0h20min	Envoie d'un mail avec le journal de travail et rapport aux experts et chef de projet		
Doc - Journal de travail	4	0h20min	Remplir la séquence 23+24 du journal		
Doc - Compte rendu	3	0h15min	Réponse aux mails si existants		
Réalisation - Interfaces	12	1h	Vérification des commentaires et variables dans le code		
Réalisation - Interfaces	15	1h15min	Dernière retouche du code		
Total tranche	38	3h10min			

Séquence 25			Date: lundi, 5 juin 2023		Matin
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Doc - Compte rendu	3	0h15min	Réponse aux mails si existants		
Doc - Rapport	35	2h55min	Mise à jour du rapport		
Total tranche	38	3h10min			



Séquence 26			Date: lundi, 5 juin 2023		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Doc - Journal de travail	4	0h20min	Remplir la séquence 25+26 du journal		
Doc - Rapport	34	2h50min	Mise à jour du rapport		
Total tranche	38	3h10min			

Séquence 27			Date: mardi, 6 juin 2023		Matin
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Doc - Rapport	35	2h55min	Mise à jour du rapport		
Doc - Compte rendu	3	0h15min	Réponse aux mails si existants		
Total tranche	38	3h10min			

Séquence 28			Date: mardi, 6 juin 2023		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Doc - Compte rendu	10	0h50min	Envoie du TPI terminé avec livrables		
Doc - Journal de travail	2	0h10min	Remplir la séquence 27+28 du journal		
Doc - Rapport	10	0h50min	Mise à jour du rapport		
Total tranche	22	1h50min			



## 7.5 Journal de travail

Séquence 1			Date: jeudi, 11 mai 2023	Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...
Doc - Journal de travail	8	0h40min	Création du journal de travail	
Analyse - Planification initiale	10	0h50min	Réalisation de la planification initial (séquence 1-22)	
Doc - Journal de travail	3	0h15min	Remplir la séquence 1 du journal	
Doc - Cahier des charges	14	1h10min	Visite de M.Oberson (expert), lire + signer cahier des charges	
Doc - Compte rendu	3	0h15min	Envoie d'un mail avec la planification initial aux experts et chef de projet	
Total tranche	38	3h10min		

Séquence 2			Date: vendredi, 12 mai 2023	Matin
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...
Doc - Compte rendu	2	0h10min	Répondre au mail de M. Oberson pour les mails groupé	
Analyse - Planification initiale	12	1h	Réalisation de la planification initial (séquence 22-28), correction séquence (séquence 1-22)	
Doc - Rapport	12	1h	Réalisation d'un squelette du rapport (rapport global)	
Réalisation - Interfaces	20	1h40min	Conception de l'application maquette (Schéma Figma du design de l'application)	
Doc - Journal de travail	2	0h10min	Remplir la séquence 2 du journal	
Total tranche	48	4h		



Séquence 3			Date: vendredi, 12 mai 2023		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Réalisation - Interfaces	6	0h30min	Conception de l'application UML (à continuer)	<a href="https://www.lucidchart.com/pages/fr">https://www.lucidchart.com/pages/fr</a>	
Doc - Rapport	12	1h	Réalisation d'un squelette du rapport (rapport global)		
Réalisation - Interfaces	8	0h40min	Mise en place des fichiers et librairie à télécharger	<a href="https://dejavu-fonts.github.io/">https://dejavu-fonts.github.io/</a>	
Réalisation - Git	8	0h40min	Réalisation d'un repos git et liaison avec VSCode (à continuer ,envoi d'un lien au expert fichier de code)	<a href="https://github.com/LovlMark/TPI_ChessPy_MarkLovink">https://github.com/LovlMark/TPI_ChessPy_MarkLovink</a> <a href="https://git-scm.com/docs/git">https://git-scm.com/docs/git</a>	
Doc - Journal de travail	1	0h5min	Remplir la séquence 3 du journal		
Doc - Compte rendu	3	0h15min	Envoi d'un mail avec la planification initial, rapport, maquette aux experts et chef de projet		
Total tranche	38	3h10min			

Séquence 4			Date: lundi, 15 mai 2023		Matin
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Réalisation - Interfaces	8	0h40min	Création des pièces et pions (à continuer)		
Réalisation - Git	6	0h30min	Réalisation d'un git ignore et d'une branch "dev" (réussi)	<a href="https://www.youtube.com/watch?v=3KqEytC5gc">https://www.youtube.com/watch?v=3KqEytC5gc</a>	
Doc - Journal de travail	1	0h5min	Remplir la séquence 4 du journal		
Réalisation - Interfaces	23	1h55min	Création du plateau de jeu pour l'application (réussi)		
Total tranche	38	3h10min			



Séquence 5			Date: lundi, 15 mai 2023		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Réalisation - Test système	4	0h20min	Affichage des FPS (réussi)	<a href="https://coderslegacy.com/python/display-fps-pygame/">https://coderslegacy.com/python/display-fps-pygame/</a>	
Doc - Journal de travail	1	0h5min	Remplir la séquence 5 du journal		
Réalisation - Interfaces	12	1h	Création des pièces et pions (réussi)		
Réalisation - Interfaces	6	0h30min	Affichages des pièces et pions (réussi) (mouvement par pièce à ajouter)		
Réalisation - Interfaces	6	0h30min	Conception de l'application UML V1 (fini)	<a href="https://www.lucidchart.com/pages/fr">https://www.lucidchart.com/pages/fr</a>	
Réalisation - Interfaces	9	0h45min	Interaction des pièces et pions avec la souris (à continuer)	<a href="https://www.pygame.org/docs/ref/event.html">https://www.pygame.org/docs/ref/event.html</a>	
Total tranche	38	3h10min			

Séquence 6			Date: mardi, 16 mai 2023		Matin
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Réalisation - Interfaces	12	1h	Interaction des pièces et pions avec la souris (réussi)		
Réalisation - Interfaces	19	1h35min	Changement de l'affichage des pions et pièces (affichage correcte mais problème avec les mouvement avec cette méthode, sans tableau) (réparé)		
Réalisation - Interfaces	6	0h30min	Mouvement possible des pièces (à continuer)		
Doc - Journal de travail	1	0h5min	Remplir la séquence 6 du journal		
Total tranche	38	3h10min			



Séquence 7			Date: mardi, 16 mai 2023		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Réalisation - Interfaces	16	1h20min	Mouvement possible des pièces (à continuer) (Pions et Roi effectué)		
Doc - Rapport	18	1h30min	Mise à jour du rapport (point réalisation/planif)		
Doc - Journal de travail	1	0h5min	Remplir la séquence 7 du journal		
Doc - Compte rendu	3	0h15min	Envoie d'un mail avec le journal de travail, rapport, maquette et UML aux experts et chef de projet		
Total tranche	38	3h10min			

Séquence 8			Date: mercredi, 17 mai 2023		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Réalisation - Interfaces	12	1h	Mouvement possible des pièces (à continuer) (Cavaliers)		
Réalisation - Interfaces	3	0h15min	Mise à jour mouvement des pièces (Roi)		
Réalisation - Interfaces	12	1h	Modification des nom des variables pour une meilleur compréhension		
Doc - Journal de travail	1	0h5min	Remplir la séquence 8 du journal		
Total tranche	28	2h20min			



Séquence 9			Date: lundi, 22 mai 2023		Matin
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Réalisation - Interfaces	6	0h30min	Mouvement possible des pièces (Tours)		
Réalisation - Interfaces	6	0h30min	Mise à jour mouvement de la classe pièces (réussi)		
Réalisation - Interfaces	12	1h	Création de la class "Player" (réussi)		
Réalisation - Interfaces	6	0h30min	Possibilité de jouer une fois sur deux (réussi)		
Réalisation - Interfaces	6	0h30min	Mouvement possible des pièces (à continuer) (Fou)		
Doc - Journal de travail	2	0h10min	Remplir la séquence 9 du journal		
Total tranche	38	3h10min			

Séquence 10			Date: lundi, 22 mai 2023		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Réalisation - Interfaces	3	0h15min	Mouvement possible du fou (à continuer) (Le fou peut sauter au dessus des pièces de même couleurs)		
Réalisation - Interfaces	3	0h15min	Mouvement possible de la reine (à continuer) (La reine peut sauter au dessus des pièces de même couleurs)		
Doc - Rapport	28	2h20min	Mise à jour du rapport (modification de la planif, réalisation chapitre 5.4) (ajout UML, réalisation chapitre 5.4-5.6.1) (à continuer)		
Doc - Journal de travail	1	0h5min	Remplir la séquence 10 du journal		
Doc - Compte rendu	3	0h15min	Envoie d'un mail avec le journal de travail, rapport, maquette et UML aux experts et chef de projet		
Total tranche	38	3h10min			



Séquence 11			Date: mardi, 23 mai 2023		Matin
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Réalisation - Interfaces	1	0h5min	Mouvement possible du fou (réussi) (break placé au mauvais endroit)		
Réalisation - Interfaces	3	0h15min	Mouvement possible de la reine (réussi) (break placé au mauvais endroit)		
Réalisation - Interfaces	3	0h15min	Mouvement possible de la tour (réussi) (optimisation du code)		
Réalisation - Interfaces	8	0h40min	Affichage des pièces morte (réussi) (Parcourir la liste avec toutes les pièces perdues)		
Réalisation - Interfaces	22	1h50min	Enregistrer une partie d'échec (à continuer) (tentative avec json/liste enregistrant tous les déplacement/ fichier text)	<a href="https://www.journaldunet.com/web-tech/developpeur/1055681-le-format-json-ajax-et-jquery/1055683-json-exemple-complet">https://www.journaldunet.com/web-tech/developpeur/1055681-le-format-json-ajax-et-jquery/1055683-json-exemple-complet</a>  <a href="https://www.w3schools.com/python/python_file_write.asp">https://www.w3schools.com/python/python_file_write.asp</a>	
Doc - Journal de travail	1	0h5min	Remplir la séquence 11 du journal		
Total tranche	38	3h10min			





Séquence 12			Date: mardi, 23 mai 2023		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Réalisation - Interfaces	9	0h45min	Enregistrer une partie d'échec (réussi) (modification des pièces pour avoir un id, enregistrement des id des pièces avec leurs lignes, colonnes et le joueur dans un fichier json)		
Réalisation - Interfaces	7	0h35min	Chargement de la partie d'échec (réussi) (mettre les informations du json dans une liste, la parcourir et modifier les colonnes et lignes des pièces ayant la même id)		
Réalisation - Interfaces	6	0h30min	Ajouts d'un classe bouton pour les interactions (réussi) (méthode pour les clics et l'affichage)		
Réalisation - Interfaces	6	0h30min	Ajout d'un fichier menu avec des boutons pour l'options du jeu (réussi) (bouton pour recharger une ancienne partie, bouton pour une nouvelle partie)		
Réalisation - Interfaces	3	0h15min	Ajout d'un bouton retour au menu dans le jeu (réussi)		
Doc - Compte rendu	3	0h15min	Envoie d'un mail avec le journal de travail, rapport, maquette et UML aux experts et chef de projet		
Réalisation - Interfaces	2	0h10min	Commenter le fichier "Bishop" (réussi)		
Doc - Journal de travail	2	0h10min	Remplir la séquence 12 du journal modification des séquences précédente		
Total tranche	38	3h10min			

Séquence 13			Date: mercredi, 24 mai 2023		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Doc - Compte rendu	4	0h20min	Visite expert 2, M.Malherbe+ discussion sur l'avancement du projet		
Réalisation - Interfaces	23	1h55min	Ajouts de commentaires manquants et corrections des variables aux conventions de l'ETML		
Doc - Journal de travail	1	0h5min	Remplir la séquence 13 du journal		
Total tranche	28	2h20min			



Séquence 14			Date: jeudi, 25 mai 2023		Matin
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Réalisation - Interfaces	18	1h30min	Enregistrement de tous les derniers mouvement des pièces et affichage de ces-derniers (réussi)		
Réalisation - Interfaces	6	0h30min	Ajouts de commentaires manquants et corrections des variables aux conventions de l'ETML		
Réalisation - Interfaces	23	1h55min	Affichage des 5 derniers mouvement joués (réussi)		
Doc - Journal de travail	1	0h5min	Remplir la séquence 14 du journal		
Total tranche	48	4h			

Séquence 15			Date: jeudi, 25 mai 2023		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Doc - Rapport	25	2h5min	Mise à jour du rapport (réalisation chapitre 5.6.1-5.6.5 + début de test performance) (à continuer)		
Réalisation - Interfaces	6	0h30min	Changement des boutons (design au clic et au survole) (réussi)		
Réalisation - Interfaces	6	0h30min	Enregistrement des coups déjà joué dans la parti dans le fichier JSON (réussi)		
Doc - Journal de travail	1	0h5min	Remplir la séquence 15 du journal		
Total tranche	38	3h10min			



Séquence 16			Date: vendredi, 26 mai 2023		Matin
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Réalisation - Interfaces	12	1h	Ajouts de commentaires manquants et corrections des variables aux conventions de l'ETML		
Réalisation - Interfaces	6	0h30min	Changer les joueurs et l'affichages des derniers mouvement (réussi)		
Réalisation - Interfaces	29	2h25min	Ajouts des fonction échec + échec et mat (état de echec) (réussi) (restraining le mouvement pour protéger le roi) (à continuer)		
Doc - Journal de travail	1	0h5min	Remplir la séquence 16 du journal		
Total tranche	48	4h			

Séquence 17			Date: vendredi, 26 mai 2023		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Réalisation - Interfaces	34	2h50min	Ajouts des fonction échec + échec et mat (restraining le mouvement pour protéger le roi) (réussi) (Simulation du prochain coup pour restreindre mouvement) (à continuer)		
Doc - Journal de travail	1	0h5min	Remplir la séquence 17 du journal		
Doc - Compte rendu	3	0h15min	Envoie d'un mail avec le journal de travail, rapport, maquette et UML aux experts et chef de projet		
Total tranche	38	3h10min			

Séquence 18			Date: mardi, 30 mai 2023		Matin
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Analyse - Retour d'utilisateur	6	0h30min	Création d'un formulaire pour les test de l'application		
Réalisation - Interfaces	31	2h35min	Ajouts des fonction échec + échec et mat (Simulation du prochain coup pour restreindre mouvement) (réussi) (pions à continuer)		
Doc - Journal de travail	1	0h5min	Remplir la séquence 18 du journal		
Total tranche	38	3h10min			



Séquence 19			Date: mardi, 30 mai 2023		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Réalisation - Test système	15	1h15min	Réalisation de test unitaire sur les position en cas d'échec entre la pièce et le roi adverse (réussi)	<a href="https://geekflare.com/fr/unit-testing-with-python-unittest/">https://geekflare.com/fr/unit-testing-with-python-unittest/</a>	
Analyse - Retour d'utilisateur	4	0h20min	Etude des retour déjà reçu des feedback des bêta-testeur		
Réalisation - Interfaces	6	0h30min	Modification des chiffres et lettres sur le pleatu de jeu du feedback (réussi)		
Doc - Rapport	8	0h40min	Mise à jour du rapport (réalisation chapitre des test + test performance) (à continuer)		
Doc - Journal de travail	2	0h10min	Remplir la séquence 19 du journal		
Doc - Compte rendu	3	0h15min	Envoie d'un mail avec le journal de travail, rapport, maquette et UML aux experts et chef de projet		
Total tranche	38	3h10min			
Séquence 20			Date: mercredi, 31 mai 2023		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Doc - Rapport	12	1h	Mise à jour du rapport (réalisation chapitre des test + test performance + répertoire + chapitre feedback) (à continuer)		
Réalisation - Interfaces	15	1h15min	Modification des restriction de pièces lors d'un cas d'echec si une pièce est déplacé par la simulation (création d'une fonction testSimulation) (à continuer)		
Doc - Journal de travail	1	0h5min	Remplir la séquence 20 du journal		
Total tranche	28	2h20min			



Séquence 21			Date: jeudi, 1 juin 2023	Matin
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...
Réalisation - Interfaces	12	1h	Modification des restriction de pièces lors d'un cas d'echeck si une pièce est déplacé par la simulation (suppression fonction testSimulation) (modification fonction des pièces pour n'autoriser que des déplacement pour protéger le roi) (réussi)	
Réalisation - Interfaces	18	1h30min	Création d'une méthode pour empêcher le roi de se mettre en échec lors d'un déplacement (réussi)	
Réalisation - Interfaces	4	0h20min	Modification de la fonction checkCheckMat pour définir le gagnant (réussi)	
Réalisation - Interfaces	6	0h30min	Modification des classe "rook", "queen", "bishop" pour prendre en compte les pièces et pions se trouvant en eux et les positions pour protéger le roi (réussi)	
Réalisation - Interfaces	6	0h30min	Vérification des commentaires et variables dans le code	
Doc - Journal de travail	2	0h10min	Remplir la séquence 21 du journal	
Total tranche	48	4h		

Séquence 22			Date: jeudi, 1 juin 2023	Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...
Réalisation - Interfaces	12	1h	Modification des classe "rook", "queen", "bishop" pour prendre en compte les pièces et pions se trouvant en eux et les positions pour protéger le roi (bug trouvé dans la fonction réaliser le matin) (réussi)	
Réalisation - Interfaces	6	0h30min	Vérification des commentaires et variables dans le code	
Réalisation - Interfaces	1	0h5min	Première version du jeu d'échec complètement fonctionnel	
Doc - Rapport	17	1h25min	Mise à jour du rapport (réalisation du chapitre objectif + affichage du dernier mouvement + feedback) (à continuer)	
Doc - Journal de travail	2	0h10min	Remplir la séquence 22 du journal	
Total tranche	38	3h10min		



Séquence 23			Date: vendredi, 2 juin 2023		Matin
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Analyse - Retour d'utilisateur	4	0h20min	Etude des retour déjà reçu des feedback des bêta-testeur		
Réalisation - Interfaces	4	0h20min	Modification de la boucle après une mise en échec après feedback (réussi)		
Doc - Rapport	38	3h10min	Mise à jour du rapport (réalisation du chapitre sauvegarde d'une partie + des maquettes + chargement d'une partie + affichage du dernier mouvement + échec et mat + échec) (à continuer)		
Doc - Journal de travail	2	0h10min	Remplir la séquence 23 du journal		
Total tranche	48	4h			

Séquence 24			Date: vendredi, 2 juin 2023		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Doc - Rapport	33	2h45min	Mise à jour du rapport (réalisation du chapitre échec et mat + modification et correction du rapport en entier) (à continuer)		
Doc - Journal de travail	2	0h10min	Remplir la séquence 24 du journal		
Doc - Compte rendu	3	0h15min	Envoie d'un mail avec le journal de travail, rapport, maquette et UML aux experts et chef de projet		
Total tranche	38	3h10min			



Séquence 25			Date: lundi, 5 juin 2023		Matin
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Doc - Rapport	12	1h	Mise à jour du rapport (correction des chapitres) (à continuer)		
Réalisation - Interfaces	6	0h30min	Modification des simulations (autorisation de prendre les positions entre menace et roi seulement si le roi n'est pas en échec) (réussi)		
Analyse - Retour d'utilisateur	12	1h	Etude des retour reçu des feedback des bêta-testeur		
Réalisation - Interfaces	6	0h30min	Ajout de chronomètre pour les joueur (réussi)		
Doc - Journal de travail	2	0h10min	Remplir la séquence 25 du journal		
Total tranche	38	3h10min			

Séquence 26			Date: lundi, 5 juin 2023		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Doc - Rapport	24	2h	Mise à jour du rapport (correction des chapitres + feed back + conclusion + annexes) (à continuer)		
Analyse - Retour d'utilisateur	12	1h	Etude des retour reçu des feedback des bêta-testeur		
Doc - Journal de travail	2	0h10min	Remplir la séquence 26 du journal		
Total tranche	38	3h10min			

Séquence 27			Date: mardi, 6 juin 2023		Matin
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Doc - Rapport	36	3h	Mise à jour du rapport (correction des chapitres + conclusion + annexes) (à continuer)		
Doc - Journal de travail	2	0h10min	Remplir la séquence 27 du journal		
Total tranche	38	3h10min			



Séquence 28			Date: mardi, 6 juin 2023	Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...
Doc - Rapport	8	0h40min	Mise à jour du rapport (correction des chapitres + conclusion + annexes)	
Doc - Compte rendu	12	1h	Envoi du TPI terminé avec livrables	
Doc - Journal de travail	2	0h10min	Remplir la séquence 24 du journal	
Total tranche	22	1h50min		



## 7.6 Table des illustrations

Figure 1 Planification initiale .....	7
Figure 2 Schéma UML version 1 .....	10
Figure 3 Schéma UML version final.....	10
Figure 4 Figma page d'accueil.....	11
Figure 5 Figma plateau vierge .....	11
Figure 6 Figma joueur 1 .....	12
Figure 7 Figma joueur 2 .....	12
Figure 8 Figma dernier mouvement.....	13
Figure 9 Figma pièce prise .....	13
Figure 10 Installation PyGame .....	14
Figure 11 Git initialisation .....	14
Figure 12 Git connexion .....	14
Figure 13 Git nouvelle branche.....	14
Figure 14 Git "gitignore".....	14
Figure 15 Répertoire de travail .....	17
Figure 16 PyGame Init.....	18
Figure 17 Pygame boucle principale .....	18
Figure 18 Classe "Square" .....	19
Figure 19 Classe "Board" .....	20
Figure 20 Fonction "DrawBoard" .....	21
Figure 21 Fonction "DrawBoarder" .....	21
Figure 22 Implémentation classe "Board" .....	21
Figure 23 Fonction affichage complet du jeu .....	21
Figure 24 Boucle principale avec affichage .....	22
Figure 25 Affichage du plateau d'échec.....	22
Figure 26 Classe parent "Piece" .....	23
Figure 27 Classe "Pawn" .....	24
Figure 28 Fonction "DrawPieces" pions.....	24
Figure 29 Classe "King" .....	25
Figure 30 Classe "Queen" .....	25
Figure 31 Classe "Bishop" .....	25
Figure 32 Classe "Rook" .....	25
Figure 33 "Classe "Knight" .....	25
Figure 34 Fonction "DrawPieces" pièces.....	26
Figure 35 Fonction affichage complet du jeu .....	27
Figure 36 Affichage du plateau et pièces .....	27
Figure 37 Evènements souris .....	28
Figure 38 Evènement cliqué .....	28
Figure 39 Fonction "Clicked" .....	29
Figure 40 Vérification des pièces cliqué .....	29
Figure 41 Suivi des pièces et souris .....	29
Figure 42 Suivi de surface cliquable des pièces.....	29
Figure 43 Nouvelle position pièce.....	30
Figure 44 Résultat prise d'une pièce avant .....	30
Figure 45 Résultat prise de pièce après .....	30
Figure 46 Fonction des mouvement légaux.....	31
Figure 47 Appelle fonction des mouvements légaux .....	31
Figure 48 Coups légaux roi.....	32

Figure 49 Mouvement du roi.....	32
Figure 50 Fonction "Movement" roi .....	32
Figure 51 Coups légaux cavalier .....	33
Figure 52 Mouvement du cavalier .....	33
Figure 53 Résultat cavalier .....	33
Figure 54 Coups légaux dame.....	34
Figure 55 Mouvement de la dame .....	34
Figure 56 Fonction "Movement" dame.....	34
Figure 57 Coups légaux fou.....	35
Figure 58 Mouvement du fou.....	35
Figure 59 Résultat fou .....	35
Figure 60 Coups légaux tour .....	36
Figure 61 Mouvement de la tour.....	36
Figure 62 Résultat tour .....	36
Figure 63 Coups légaux pion.....	37
Figure 64 Mouvement du pion bleu .....	37
Figure 65 Mouvement du pion rouge.....	37
Figure 66 Coups légaux pion en attaque .....	38
Figure 67 Mouvement attaque pion bleu droite .....	38
Figure 68 Mouvement attaque pion bleu gauche.....	38
Figure 69 Mouvement attaque pion rouge droite .....	38
Figure 70 Mouvement attaque pion rouge gauche .....	38
Figure 71 Affichage des mouvements possibles .....	39
Figure 72 Résultat mouvement des pions .....	39
Figure 73 Pièces en échec .....	40
Figure 74 Liste simple "checkPos" .....	40
Figure 75 Menace position .....	41
Figure 76 Menace direction simulation tour .....	41
Figure 77 Menace tour colonne .....	41
Figure 78 Menace tour ligne .....	41
Figure 79 Menace distance roi et dame .....	42
Figure 80 Menace direction simulation dame .....	42
Figure 81 Menace positions entre roi et dame.....	42
Figure 82 Menace direction fou .....	43
Figure 83 Menace positions entre roi et fou.....	43
Figure 84 Appel « Simulation ».....	44
Figure 85 Simulation du plateau .....	44
Figure 86 Simulation du prochain tour.....	44
Figure 87 Simulation vérification échec .....	45
Figure 88 Simulation mouvement du roi .....	45
Figure 89 Restriction de mouvement du roi .....	45
Figure 90 Résultats restriction mouvement du roi .....	46
Figure 91 Restriction mouvement en cas échec.....	46
Figure 92 Résultats protection roi échec .....	47
Figure 93 Résultat simulation échec.....	47
Figure 94 Boucle principale "CheckCheckMate" .....	48
Figure 95 Fonction "CheckCheckMate" retour faux .....	48
Figure 96 Fonction "CheckCheckMate" retour vrai .....	49
Figure 97 Afficher le gagnant ou match nul .....	49
Figure 98 Fonction "DrawWinner" .....	50

Figure 99 Fonction "DrawDraw" .....	50
Figure 100 Gagnant échec et mat .....	50
Figure 101 fonction "SaveGame" .....	51
Figure 102 Fonction "LoadGame" .....	52
Figure 103 Charger les positions des pièces.....	52
Figure 104 Charger tous les mouvements.....	52
Figure 105 Charger le joueur.....	53
Figure 106 Enregistrement des mouvements .....	53
Figure 107 Bouton "showLastMovement".....	54
Figure 108 Afficher les derniers mouvements .....	55
Figure 109 Résultats dernier mouvement avant .....	55
Figure 110 Résultats dernier mouvement après .....	55
Figure 111 Feedback plateau de jeu .....	56
Figure 112 Fonction "draw" après Feedback.....	56
Figure 113 Fonction "DrawNumber" .....	57
Figure 114 Avant feedback .....	57
Figure 115 Après feedback.....	57
Figure 116 Feedback mouvement roi en échec.....	58
Figure 117 Feedback problème mouvement du roi .....	58
Figure 118 Feedback solution mouvement du roi .....	58
Figure 119 Feedback question 1 .....	58
Figure 120 Feedback question 2 .....	59
Figure 121 Feedback question 3 .....	59
Figure 122 Feedback question 4 .....	59
Figure 123 Feedback question 5 .....	60
Figure 124 Feedback question 6 .....	60
Figure 125 Feedback question 7 .....	61
Figure 126 Feedback question 8 .....	61
Figure 127 Feedback question 9 .....	62
Figure 128 FPS .....	63
Figure 129 FPS résultat.....	63
Figure 130 Test unitaire colonne .....	64
Figure 131 Test unitaire ligne .....	64
Figure 132 Test unitaire rien.....	64
Figure 133 Test unitaire lancement .....	65
Figure 134 Test unitaire résultats .....	65
Figure 135 Comparaison entre les tâches planifiées et réalisées.....	68

## 7.7 Code source

[MarkLovink\\_ChessPy](#)