

---

CENTRALESUPÉLEC

PROJET VISION PAR ORDINATEUR « VISUALCHESS »

---

# Reconstitution automatique d'une partie d'échecs

---

*Auteurs :*  
Mathian AKANATI  
Guilhem PRINCE

*Encadrant :*  
Céline HUDELOT

Avril 2023



CentraleSupélec

---

# Table des matières

<b>1</b>	<b>Introduction et contexte</b>	<b>2</b>
<b>2</b>	<b>Choix d'implémentation</b>	<b>2</b>
<b>3</b>	<b>Implémentation des modules</b>	<b>3</b>
3.1	Reconnaissance et découpage des cases . . . . .	3
3.1.1	Introduction . . . . .	3
3.1.2	Description de la pipeline . . . . .	3
3.1.3	Difficultés rencontrées . . . . .	7
3.2	Détection d'une pièce et de sa couleur sur une case . . . . .	7
3.3	Suivi de la partie . . . . .	8
3.4	Traitement des coups joués . . . . .	9
3.4.1	Introduction aux notations . . . . .	9
3.4.2	Implémentation . . . . .	9
<b>4</b>	<b>Résultats</b>	<b>9</b>
4.1	Sur Chess.com . . . . .	9
4.2	Sur échiquier réel . . . . .	10
4.2.1	Reconnaissance et Découpage des cases . . . . .	10
4.2.2	Détection d'une pièce et de sa couleur sur une case . . . . .	10
<b>5</b>	<b>Pistes d'amélioration</b>	<b>11</b>
<b>6</b>	<b>Conclusion</b>	<b>12</b>
<b>7</b>	<b>Bibliographie</b>	<b>13</b>

# 1 Introduction et contexte

Le jeu d'échecs est une activité passionnante connaissant un essor important ces dernières années. La pratique de ce jeu a été grandement popularisée par les confinements répétés à travers le monde. En effet, il existe de nombreuses manières d'apprendre à jouer en autodidacte (chaîne Youtube, applications dédiées aux échecs...) et il est possible de jouer contre des milliers de personnes depuis son domicile. La série « The Queen's Gambit » et la présence au plus haut niveau de Magnus Carlsen, considéré par beaucoup comme étant le meilleur joueur d'échec de l'histoire ont également contribué à cet essor.

Pendant des années, lors des parties longues, un officiel bougeait les pièces manuellement sur un grand tableau derrière les joueurs pour montrer au public les coups joués. Cela permettait aux commentateurs d'analyser la partie en direct, et aux spectateurs de réfléchir aux coups suivants. Cette méthode est tout de même fastidieuse, sujette aux erreurs, et inadaptée pour des parties jouées en cadence rapide ou très rapide (blitz). De nos jours, le jeu d'échecs s'est démocratisé, et s'est adapté à internet : des milliers de personnes regardent les compétitions en direct. La solution manuelle est toujours possible, où quelqu'un retransmet les coups sur un échiquier virtuel, mais en pratique il y a beaucoup d'erreurs, et cela se fait avec un délai non négligeable (surtout pour les parties rapides encore une fois). C'est pourquoi la reconstitution des parties (en direct ou après coup) pourrait être faite à l'aide de vision par ordinateur, si l'on dispose d'un flux vidéo ou des photos de chacun des coups d'une partie. Il est facile d'imaginer un système rapide, proche d'instantané, et sans erreurs pour retransmettre et permettre d'assister à une partie sans soucis.

Notre projet propose un outil pour convertir des parties jouées sur un réel échiquier en un format interprétable par un ordinateur afin de faciliter la diffusion et l'analyse des parties. Nous avons donc travaillé sur un algorithme qui transcrit dans un format interprétable par l'ordinateur une succession d'images d'échiquier. Notre code est disponible sur ce dépôt github.

## 2 Choix d'implémentation

Tous nos programmes ont été conçus en python. Nous utilisons des Jupyter Notebook et un fichier `utils.py` qui répertorie l'ensemble des fonctions d'intérêt. Le fichier `'main.ipynb'` est celui à exécuter pour faire fonctionner l'algorithme. Les autres Jupyter Notebook sont les fichiers nous ayant permis de concevoir et tester les fonctions de `'utils.py'`.

Par souci de simplification, nous prenons en entrée une image par nombre de coups joués, et non une vidéo. La première image fournie doit être celle de la position initiale, lorsque qu'aucun coup n'a été joué. Les images doivent être renommées "0", "1", "2", "3", en format `.png` ou `.jpg`. Pour le bon fonctionnement de nos algorithmes, il faut lui fournir des images claires, avec une bonne luminosité prise de dessus, en évitant les ombres. Il faut également que les pièces blanches soient localisées à gauche de l'échiquier et que les pièces noires soient situées à droite (Figure 2), comme si nous avions une caméra ou un spectateur qui assiste à une partie sur le côté, du dessus.

En sortie, nous renvoyons la liste des coups de la partie, sous le format PGN, qui se présente comme tel (voir 3.4.1) :

`1.e4 e5 2.Nf3 Nc6 3.Bb5` (ouverture espagnole, ou Ruy Lopez)

Les blancs jouent pion e4, les noirs répondent pion e5, c'est le premier coup de la partie. Les blancs jouent Cavalier en f3, les noirs Cavalier en c6, et ainsi de suite.



FIGURE 1 – Image d'un coup joué par blanc sur un vrai échiquier

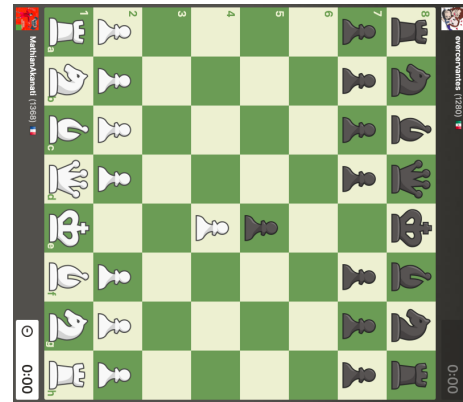


FIGURE 2 – Image d'une réponse jouée par noir sur chess.com

## 3 Implémentation des modules

### 3.1 Reconnaissance et découpage des cases

#### 3.1.1 Introduction

La première chose à implémenter pour notre projet est la détection des différentes cases de l'échiquier. Ainsi, ce module prend en entrée la photo d'un échiquier, et renvoie un dictionnaire de cette forme :

```
{'a1':[(97, 6), (175, 6), (97, 94), (175, 94)], 'a2':[(175, 6), (265, 6), (175, 94), (265, 94)], ...}
```

Les 64 clés sont les coordonnées échiquiennes des cases, et les valeurs sont les listes des coordonnées en pixels des quatre coins de la case : ( $posX, posY$ ). Les cases sont nommées par le nom de leur colonne (de 'a' à 'h') et le nom de leur ligne (de 1 à 8).

Avec une caméra fixe et un échiquier fixe, on pourrait ne faire tourner ce module qu'une seule fois. Cela dit, un joueur peut par inadvertance décaler l'échiquier, ou dans le cas de plusieurs photos prises à la main, l'appareil n'est pas toujours situé au même endroit. C'est pourquoi nous avons décidé de d'appliquer ce module sur chaque photo de la partie.

#### 3.1.2 Description de la pipeline

Une fois l'image importée, nous la passons dans un filtre gaussien de noyau de taille 5, puis nous appliquons ensuite un seuil binaire dessus. Nous avons essayé de calculer les filtres de Sobel en x et en y, pour déterminer les lignes et segments, avant d'utiliser finalement les contours, obtenus grâce au filtre de Canny.

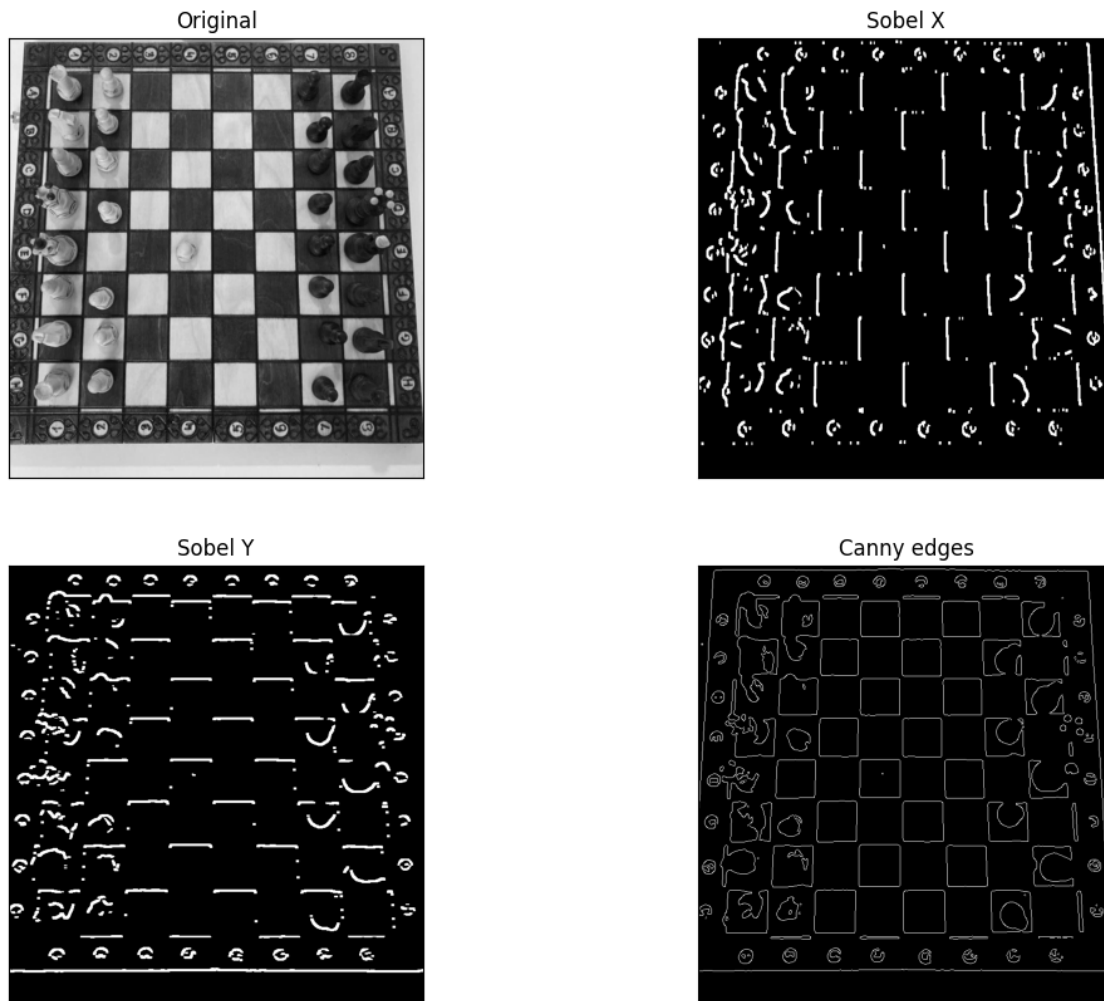


FIGURE 3 – Les différents filtres

L'objectif est de passer l'image résultante dans le détecteur de lignes de hough, en version probabiliste. Nous avons choisi la version probabiliste car celle-ci détecte aussi les segments, et non seulement les lignes qui traversent totalement l'image. Une manipulation sur la longueur des segments a été rajoutée : en calculant la médiane des longueurs, nous gardons seulement les segments ayant une longueur « proche » (à un nombre de pixels près à régler) de cette médiane. Ceci nous a permis d'isoler seulement des côtés de cases. Nous obtenons ces résultats :



FIGURE 4 – Échiquier réel



FIGURE 5 – Sur chess.com

Une fois ces lignes obtenues, qui coïncident généralement avec un des côtés d'une case, il faut prolonger ces lignes, pour obtenir la délimitation complète, entre deux rangées ou colonnes. Or, lors du prolongement de chacune de ces petites lignes, nous avons beaucoup de délimitations dupliquées. C'est alors le début d'une multitude de manipulations pour se ramener à la grille 8 par 8 voulue. La première étape consiste à isoler les intersections de toutes ces lignes : on observe alors des clusters de points autour des coins des cases, et quelques points anormaux, qui ne sont pas autour d'un coin d'une case.

Nous avons créé une fonction 'clusterize\_intersections', prenant en paramètre un  $\epsilon$  qui va grouper les points de distance inférieure à  $\epsilon$  dans des clusters. Ces clusters sont ensuite remplacés par leur barycentre (par la fonction 'group\_clusters'), qui représentera un coin de case. Le choix d'épsilon n'est pas le plus primordial ici, tant qu'il est plus petit que la longueur typique d'une case. Ainsi, les clusters peuvent même absorber quelques anomalies, ce qui facilite la prochaine étape.

On remarque que les anomalies restantes sont isolées (on a aussi clusterisé et réduit les anomalies) selon l'axe des X ou l'axe des Y. Ainsi, la fonction remove\_anomalies va enlever tout point qui n'a pas d'autres points de coordonnées X semblables et de coordonnées Y semblables. On reprend pour cela le principe des clusters utilisé précédemment, mais appliqué ici seulement sur une seule dimension.

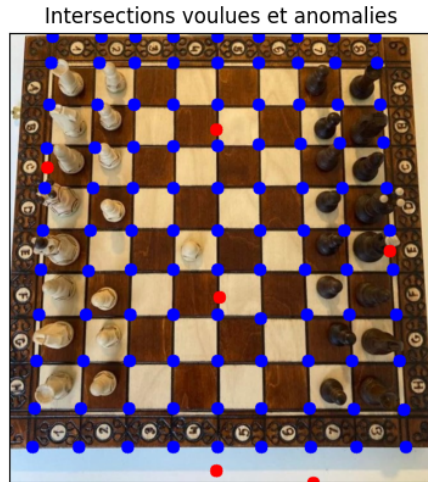


FIGURE 6 – En rouge les anomalies, en bleu les points d'intérêts

Une fois les anomalies retirées, il reste à retirer les éventuels bords de l'échiquier détectés. Une fois cela fait, il nous reste un point par coin de case. Une fois ceux-ci triés en X et en Y, on peut les ranger en un dictionnaire d'output comme vu précédemment. Voici un résultat quand l'on demande les coins des cases b5, e4, f1 et h3 :



FIGURE 7 – Output du module

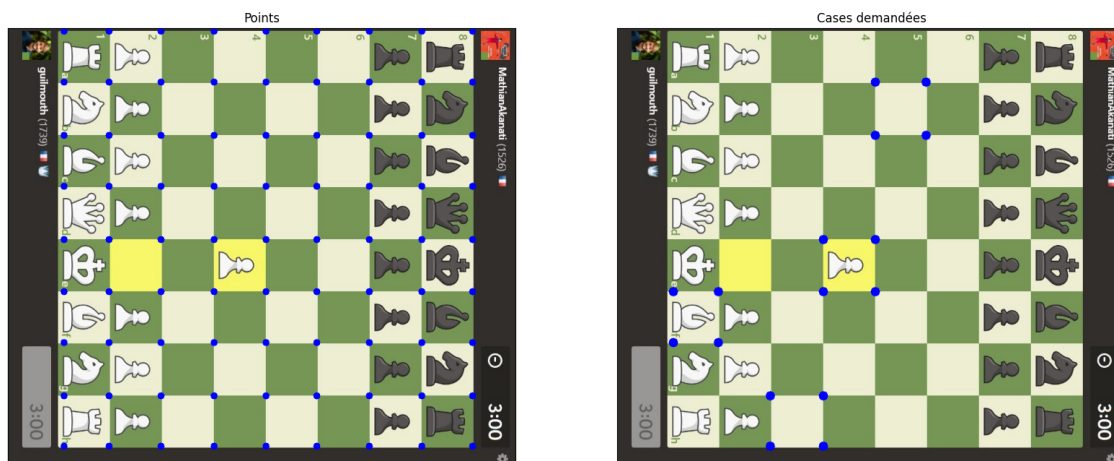


FIGURE 8 – Output du module

### 3.1.3 Difficultés rencontrées

Ce module contient beaucoup de paramètres : seuil pour le passage de l'image en binaire, seuil du détecteur de Hough, utilisation ou non de l'astuce de la médiane, les  $\epsilon$  des méthodes de clustering ... Et il faut régler tous ces paramètres à la main selon l'environnement de la photo, il y a notamment de grosses différences entre les captures d'écran prises sur chess.com et les photos de la réalité, et donc des réglages bien différents.

De plus, la quantité de manipulations à réaliser est importante, surtout pour les vrais échiquiers, pour éliminer les points qui ne nous intéressent pas et s'assurer qu'on a tous ceux qui nous intéressent. De plus, ce modèle nécessite beaucoup de réglages à la main selon l'échiquier et le positionnement de l'appareil photo. Notre outil n'est donc pas tout à fait automatisé. C'est pourquoi nous avons choisi de nous concentrer sur chess.com, où la détection des cases est plus simple et stable lors d'une partie entière.

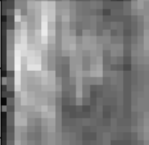
## 3.2 Détection d'une pièce et de sa couleur sur une case

La section précédente nous permet de collecter une image par case de chaque position. A partir de cette image, nous détectons la pièce et sa couleur. En connaissant la position initiale et le coup joué à chaque étape, il est possible de reconstituer la partie uniquement en détectant la couleur de la pièce sur la case (ie : case vide, ou pièce noire, ou pièce blanche).

Pour ce faire, nous disposons de 2 fonctions. Une fonction « is\_piece\_in\_square » et une fonction « is\_piece\_white ». La première se base sur l'étude du gradient de l'image. S'il est en moyenne au dessus d'un certain seuil (autour de 8) alors la case est considérée comme peuplée d'une pièce, sinon, la case est vide. En effet, une case vide est sensée être uniforme, d'où un gradient très faible. La fonction « is\_piece\_white », se basant aussi sur une fonction de seuillage, s'applique aux cases non vides, détectées par la méthode précédente. Pour cela, nous regardons le niveau de gris de la partie centrale de la case, là où est supposée se situer la pièce. Si la partie centrale de la case est assez claire, nous considérons que la pièce présente dans la case est blanche.



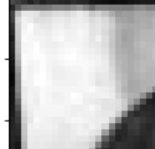
There isn't any piece in this square :



A 50x50 grayscale image showing a noisy, textured pattern. The image is labeled "There isn't any piece in this square :". The axes are labeled from 0 to 50.

Le gradient moyen de l'image est : 28.0

There is a piece in this square :



### 3.3 Suivi de la partie

- Les tableaux concernant le coup qui a été joué. il s'agit d'un tableau de 64 cases avec des '0' (si absence de pièce), des '1' (si une pièce blanche est détectée) et des 2 (si une pièce noire est détectée). Ce tableau est obtenu à partir de l'image d'une position du jeu d'échecs et les méthodes précédemment citées.
- Les tableaux concernant la position. Ils nous seront utiles pour noter quel coup a été joué. Il représente le positionnement actuel des différentes pièces. Ce tableau est obtenu à partir de la position au coup précédent et du tableau du coup qui a été joué.

```
[[2 2 2 2 2 2 2 2]
 [2 2 2 2 2 2 2 2]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0]
 [0 0 0 0 0 0 0 0]
 [1 1 1 1 0 1 1 1]
 [1 1 1 1 1 1 1 1]]
```

```
[['R' 'N' 'B' 'Q' 'K' 'B' 'N' 'R']  
 ['P' 'P' 'P' 'P' 'P' 'P' 'P' 'P']  
 [' ' ' ' ' ' ' ' ' ' ' ']  
 [' ' ' ' ' ' ' ' ' ' ' ']  
 [' ' ' ' ' ' ' ' p' ' ' ' ' ' ']  
 [' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ']  
 ['p' 'p' 'p' 'p' ' ' 'p' 'p' 'p']  
 ['r' 'n' 'b' 'q' 'k' 'b' 'n' 'r']]
```

Ici, le tableau de droite a été obtenu à partir de la position initiale ( où tous les pions 'p' de blanc sont alignées) et du tableau de gauche représentant le coup ayant été joué. Notons que les

blancs sont représentés en minuscule et les noirs en majuscule.

### 3.4 Traitement des coups joués

#### 3.4.1 Introduction aux notations

Aux échecs, il existe une notation standard pour chacun des coups : la Portable Game Notation (PGN). L'idée est de noter quelle pièce a été jouée sur quelle case.

Dénomination des pièces :

Français	Anglais	Notation
Tour	Rook	R
Cavalier	Knight	N
Fou	Bishop	B
Dame	Queen	Q
Roi	King	K
Pion	Pawn	

Notons que le pion n'a pas de lettre attribuée et que le cavalier (knight) est représenté par la lettre 'N' car le 'K' est déjà utilisé pour représenter le roi (king).

Le principe de base est donc d'écrire 'Nf3' si le cavalier se déplace en f3 ou 'e4' si un pion se déplace en e4. Pour plus d'informations sur les notations, vous pouvez consulter ce site.

#### 3.4.2 Implémentation

Dans notre modélisation, nous nous servons des tableaux de coups et de position évoqués précédemment (Figure 11) Pour chaque case du tableau de position où il y a une pièce, nous regardons si un 0 est indiqué dans le tableau des coups. Si c'est le cas, cela signifie que la pièce de cette case a été déplacée. Inversement, si une case vide dans le tableau de position est peuplée sur le tableau des coups, cela signifie que la pièce s'est déplacée sur cette case. Dans le cas particulier d'une capture, on observe uniquement la disparition d'une pièce. Pour savoir quelle est la case sur laquelle la pièce s'est déplacée, il suffit de regarder quelle case a changé de couleur (ie : quelle case est passée de 1 à 2 ou de 2 à 1), cette case est la case sur laquelle la pièce ayant effectué la capture s'est déplacée.

## 4 Résultats

### 4.1 Sur Chess.com

Chess.com est l'application dédiée aux échecs avec le plus d'utilisateurs dans le monde. Nous avons commencé par analyser les images de cet outil. C'est un cas assez simple car les images sont semblables entre elles, sans imperfections, les pièces ont invariablement la même allure, pas d'ombres, elles sont centrées... Pour cet outil, les différents paramètres de notre modèle sont à fixer à :

- Seuil pour la transformation de l'image en binaire : 200
- Utilisation de l'astuce de la mediane pour les segments : False
- Seuil pour la détection de Hough probabiliste : 150
- Seuil pour « is\_piece\_in\_color\_square » : 8
- Seuil pour « is\_piece\_white » : 160

Il suffit de lancer le fichier `main.ipynb` pour voir un exemple d'application de l'algorithme. Cela indiquera une liste de coups pour la partie dont les images sont dans « `photos_test/chess.com_game` ». C'est une partie que nous avons joué l'un contre l'autre, qui s'est soldée par un match nul : temps insuffisant contre matériel insuffisant. Nous n'avons pas d'erreur pour cette partie, ainsi que sur toutes les parties testées issues de `chess.com`.

La sortie sera :

```
1. e4 e5 2. Nf3 Nc6 3. Bb5 a6 4. Ba4 Nf6 5. d3 b5 6. Bb3 d6 7. Bg5 Be6 8. Nc3 h6
9. Bh4 Be7 10. Ke2 Ng4 11. Bxe7 Qxe7 12. h3 Kd7 13. hxg4 Bxg4 14. Re1 Nd4 15. Kf1
Nxf3 16. gxf3 Bh5 17. Nd5 Qh4 18. Ne3 Qh1 19. Ke2 Bxf3 20. Kd2 Qh3 21. Rh1 Bxh1
22. Qxh1 Qxh1 23. Rxh1 h5 24. Bxf7 h4 25. Ke2 Rf8 26. Bg6 Rf6 27. Bf5 Ke7 28. Nd5
Kf7 29. Nxf6 Kxf6 30. Kf3 g5 31. Kg4 c6 32. Kh3 d5 33. Rg1 Rd8 34. Kg4 dxe4 35. Bxe4
Rh8 36. Bxc6 Rh6 37. Be4 a5 38. Rh1 b4 39. Rh3 Ke7 40. Kf5 Kd6 41. Kxg5 Rh8 42. Rxh4
Rg8 43. Kf6 Rf8 44. Kg7 Rxf2 45. Rh6 Kc5 46. Rc6 Kd4 47. Rc4 Ke3 48. Kg6 Kd2 49. Kg5
Kc1 50. a3 Rxc2 51. Rxc2 Kxc2 52. axb4 axb4 53. Kf5 Kxb2 54. Bd5 Kc3 55. Ke4 b3
56. Bxb3 Kxb3 57. Kxe5 Kc3 58. d4 Kc4 59. d5 Kc5 60. d6 Kc6 61. Ke6 Kb7
```

Pour tester le modèle sur une autre partie, il suffit de mettre des images de parties dans un dossier puis d'indiquer le lien du dossier dans `main.ipynb`. Il faut indiquer le chemin des images de partie à ce niveau :

```
game_images_path = '../photos_test/chess.com_game/'
```

## 4.2 Sur échiquier réel

### 4.2.1 Reconnaissance et Découpage des cases

Pour les photos de vrais échiquiers, ce module est très sensible aux conditions de la photo : la moindre ombre, surexposition ou désaxage entre la caméra et la direction des lignes de l'échiquier peut faire manquer une ligne ou une colonne au détecteur de Hough. Ce problème aurait pu être réglé en rajoutant une ligne fictive au milieu de deux autres, mais d'une part nous considérons cela trop approximatif, d'autre part le temps a manqué. Alors, des photos provenant de la même partie peuvent nécessiter différents réglages, ainsi ce module devient inefficace.

Le module fonctionne pour les 6 photos du dossier 'guilhem\_board', mais nous n'avons pas essayé ce dernier sur une partie entière. Pour voir les résultats sur un échiquier réel, exécutez le fichier 'box\_detection.py' avec les paramètres :

```
binary_threshold = 128
use_median = True
hough_lines_threshold = 28
```

### 4.2.2 Détection d'une pièce et de sa couleur sur une case

Concernant la détection de pièces, nous rencontrons également des problèmes. En effet, certaines cases vides ont un gradient plus élevé que certaines cases peuplées, que ce soit en nuance de gris ou en gradient tridimensionnel. Cela est principalement dû aux jeux d'ombres et aux rainures sur les cases vides.

Ainsi, des techniques aussi simples qu'une observation du gradient sont insuffisantes malgré plusieurs tentatives (gradient en nuance de gris, tridimensionnel, s'intéresser uniquement à la partie centrale de la case...). D'autres méthodes pourraient être envisagées comme la détection de contour ou

Le gradient moyen de l'image est : 6.7  
empty

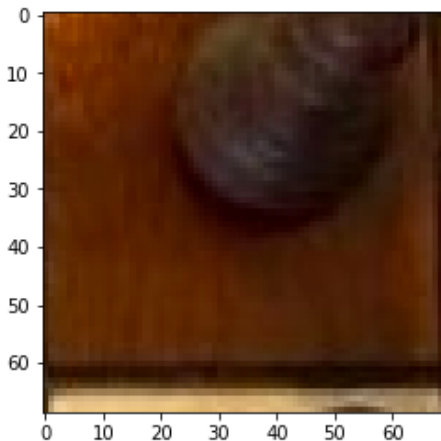


FIGURE 13 – Case peuplée avec un faible gradient

Le gradient moyen de l'image est : 7.3  
empty

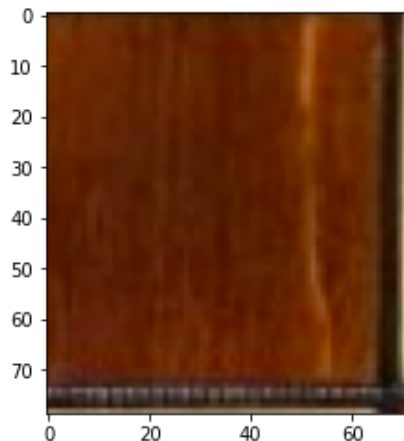


FIGURE 14 – Case vide avec un gradient élevé

l'entraînement d'un réseau de neurone. Une limite que l'on peut imposer pourrait être de s'intéresser uniquement à des échiquiers en plastique ou métal, supprimant les caractéristiques du bois qui est une matière vivante sur laquelle on peut observer des contrastes même sur une case vide. Cela pourrait être l'objet d'une étude ultérieure.

Nous observons toutefois une réussite partielle car la détection est correcte dans + de 95% des cas (en moyenne sur plus de 61 cases sur 64). Par ailleurs, la détection de couleur est correcte l'intégralité du temps avec un bon seuillage.

Meilleurs seuils pour une analyse d'échiquier réel :

- Seuil pour « is\_piece\_in\_color\_square » : 7.5
- Seuil pour « is\_piece\_white » : 100

## 5 Pistes d'amélioration

Plusieurs fonctionnalités et améliorations étaient envisagées, mais n'ont pas pu être réalisées par manque de temps. En voici une liste non exhaustive :

- Amélioration du module de détection des cases : le faire marcher sur plusieurs échiquiers, le rendre plus robuste aux différentes conditions, réduire le nombre de cas particuliers.
- Réduire le temps pris par le module de détection des cases. Solution possible : Imposer un échiquier et une caméra fixe, et ne faire tourner qu'une seule fois le module.
- Ajout des règles spéciales non gérées : le roque du roi, la prise en passant, sous-promotion du pion en un fou, cavalier, ou tour... Un module simple de deep learning serait alors nécessaire afin de reconnaître la pièce qui a été choisie pour la promotion.
- Améliorer la qualité des PGNs en output, en ajoutant des "+" lorsqu'il y a échec, et la spécification du caractère déterminant quand 2 mêmes pièces peuvent se rendre sur la même case (exemple : si deux tours, situées en a8 et h8 peuvent se rendre en e8, on notera Rae8 ou Rhe8, et non Re8).

- Création d'un module prenant une vidéo et retournant la liste de photos des coups que nous prenons actuellement en input.
- Éventuellement un algorithme de recalibrage qui met les photos dans le bon sens : pièces blanches à gauche, pièces noires à droite. Cela autoriserait donc les photos de n'importe quel angle de vue.
- Détection des chiffres sur l'horloge, pour ajouter l'information du temps restant aux joueurs à notre restitution de la partie.

## 6 Conclusion

En somme, l'étude que nous avons menée démontre qu'une application permettant de numériser des parties jouées sur un vrai échiquier est réalisable. Elle apporte déjà des résultats convaincant pour les cas les plus simples (parties jouées sur chess.com). Cependant, elle possède certaines limites (Voir 4.2, 3.1.3).

Ce projet présente des pistes d'amélioration intéressantes qui permettront de généraliser l'algorithme à d'autres type de parties.

Dans les tournois d'échecs, le problème de diffusion en direct des coups joués a été abordé différemment que par vision par ordinateur. Des échiquiers digitaux sont utilisés, dont les cases reconnaissent les pièces (contenant chacune une puce). Ce sont donc les échiquiers eux-mêmes qui détectent et transmettent les coups.

## 7 Bibliographie

### Références

- [1] M. Piškorec N. Antulov-Fantulin J. Ćurić O. Dragoljević V. Ivanac L. Karlović. Computer vision system for the chess game reconstruction, 2009/2010. [matijapiskorec.github.io](https://github.com/matijapiskorec).
- [2] Venceslas ROULLIER Benjamin RENAULT, Mokhles BOUZAIEN. Détection d'un échiquier et des mouvements des pièces d'un échiquier à partir d'un flux vidéo d'une webcam, 2018/2019. [docplayer.fr](https://docplayer.fr).
- [3] Fayolle Lucas Kulcsar Jeremy. Computer vision system for the chess game reconstruction, 2020. Anciens Élèves de l'école Centrale Paris, leur rapport est disponible sur [edunao](https://edunao.com).