Auteurs

Benjamin RENAULT, élève ingénieur, IMT Atlantique.
benjamin.renault@imt-atlantique.net
Mokhles BOUZAIEN, élève ingénieur, IMT Atlantique.
mokhles.bouzaien@imt-atlantique.net
Venceslas ROULLIER, élève ingénieur, IMT Atlantique.
venceslas.roullier@imt-atlantique.net

Destinataires
COPIL CODEV
Laurent LECORNU

Rapport de projet

Projet 04 : détection d'un échiquier et des mouvements des pièces d'un échiquier à partir d'un flux vidéo d'une webcam

Version 2.0 - 20/05/2019
Formation d'ingénieur
Année scolaire 2018-2019

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

RÉSUMÉ

Rédacteur : Benjamin RENAULT, relecteur : Venceslas ROULLIER.

Ce rapport présente le projet CODEV intitulé « Détection d'un échiquier et des mouvements des pièces d'échiquier à partir d'un flux vidéo d'une webcam », encadré par M. Laurent LECORNU. Le but du projet est de détecter un échiquier à partir d'un flux vidéo d'une webcam et de réussir à également détecter le mouvement d'un pièce en comparant le flux vidéo entre deux instants, ce qui permet alors de retranscrire automatiquement les mouvements effectués lors d'une partie d'échecs, tâche très longue à faire à la main.

Nous avons donc remis à jour une application Python utilisant la librairie *OpenCV* pour réaliser le projet, en respectant une structure du programme en différentes classes que nous avons pour la plupart reprises. Le produit final, testé au travers d'une phase de validation, est capable de retranscrire les coups effectués lors d'une partie d'échecs en découpant le plateau par cases et en sachant si une case est occupée ou non.

Mots clés : échiquier, détection, OpenCV, développement logiciel

SOMMAIRE

Index des illustrations

Index des tableaux

I. INTRODUCTION

Rédacteur : Venceslas ROULLIER, relecteur : Mokhles BOUZAIEN.

Transcrire les coups effectués lors d'une partie d'échecs est une tâche répétitive, ennuyeuse et assez longue. Une automatisation de ce processus par le traitement vidéo permettrait de faire gagner des heures à certains passionnés d'échecs. De nombreux produits existent déjà dans le commerce pour enregistrer des parties, comme des échiquiers électroniques détectant chaque mouvement. Ils sont par contre extrêmement coûteux (au moins 1000 € pour le plateau et les pièces). Il existe aussi des logiciels, tel que « Webcam Chess » [2], payant lui aussi (39 €) et dont l'interface est présentée dans Illustration 1. Son objectif est similaire à celui du produit précédent : détecter un échiquier, enregistrer et faire des parties « humain contre humain » ou « humain contre machine ». Un projet similaire au nôtre intitulé « ChessCam » a aussi été entamé [1]. L'objectif de celui-ci est de générer un module d'échecs qui détecte les mouvements des pièces et calcule le meilleur coup pour chaque position donnée. Il peut interagir avec le logiciel « Arena Chess GUI 3.0 » afin de permettre de faire des parties contre une IA avec la présence d'une interface graphique. Le logiciel disponible est libre et écrit avec Python en utilisant la librairie OpenCV [3] mais il ne semble pas fonctionnel.

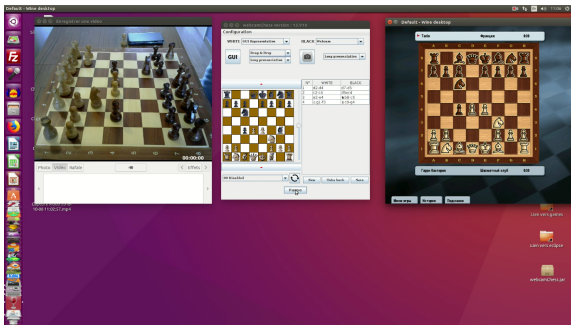


Illustration 1: L'interface du logiciel "Webcam Chess"

Notre projet consiste ainsi à développer un programme de détection d'un échiquier et des mouvements des pièces grâce au flux vidéo d'une webcam. L'objectif est d'être capable de détecter, à l'aide d'une simple webcam (Illustration 3), la présence d'un échiquier (Illustration 2) ainsi que les couleurs des pièces occupées par chaque case, pour en déduire les différents mouvements effectués. Le système doit pouvoir établir la liste des coups joués et potentiellement le faire en même temps que la partie se déroule. Il doit de plus pouvoir retranscrire les coups spéciaux. Durant le déroulement du projet, on se limite à l'utilisation d'un échiquier et d'une webcam associée à un ordinateur, ce qui limite le coût.

Illustration 2: L'échiquier utilisé

Illustration 3: La webcam utilisée

La première partie de ce rapport explique l'analyse plus détaillée du contexte du projet, ainsi que l'articulation des classes et des modules que nous proposons pour répondre aux besoins. En deuxième partie est expliqué avec précision le code développé, notamment le fonctionnement de chaque classe. Dans la partie finale, nous expliquons le fonctionnement des modules entre eux, ainsi que les scénarios validant les besoins. Nous détaillons à partir de ces scénarios un protocole de validation qui permet de savoir si le produit respecte le cahier des charges. Nous y analysons aussi les résultats, et les éventuels problèmes rencontrés lors de la mise en fonctionnement.

II. CONCEPTION DU PRODUIT

Rédacteur : Benjamin RENAULT, relecteur : Venceslas ROULLIER.

II.1 Spécification du besoin

Le programme doit simplement dans un premier temps reconnaître les bordures du plateau la présence d'une pièce et sa couleur. La reconnaissance du type de pièce se déduit à partir de la position de départ et des mouvements possibles pour chaque pièce (par exemple, si on voit qu'une pièce disparaît d'une case et qu'une autre case auparavant vide est remplie et que la pièce disparue peut effectuer un mouvement vers la case nouvellement remplie, on en déduit que cette pièce a effectué ce mouvement). On suppose donc que tous les coups joués respectent les règles du jeu d'échecs : on pourrait aboutir à des incohérences dans le cas contraire. Dans un second temps, on met en mémoire les mouvements et on propose un affichage du déroulement de la partie filmée.

Le niveau attendu dans la reconnaissance des mouvements est seulement celui des coups classiques : nous excluons dans un premier temps les coups plus complexes (comme par exemple la prise en passant) et surtout la promotion (on ne peut pas deviner la pièce qui a été choisie sans reconnaître la forme de la pièce) qui requièrent une complexité supplémentaire.

II.2 Fonctions et contraintes

Nous avons défini les fonctions que notre produit devrait respecter. Celles-ci, décrites notamment dans un cahier des charges fonctionnel, sont reprises ici. Il y a des fonctions principales qui sont indispensables pour la validation du produit, et des fonctions complémentaires (secondaires) qui sont optionnelles et assez accessoires.

Fonction principale	Description
1. Détection de l'échiquier	L'application détecte les bords et les cases de l'échiquier.
2. Détection de la présence d'une pièce et de sa couleur	Le logiciel détecte la position et la couleur d'une pièce sur une case.
3. Représentation informatique de l'échiquier	L'application représente l'échiquier informatiquement par une matrice 8x8.
4. Détection d'un mouvement	Suite à un mouvement, l'application détecte le changement de configuration et en déduit quel mouvement a été réalisé.

Tableau 1: Liste des fonctions principales et leurs descriptions

Fonction secondaire	Description
Enregistrement des coups effectués	L'application enregistre chaque coup et peut retranscrire la partie en entier
Retranscription en direct	L'application peut afficher en direct l'état de l'échiquier

Tableau 2: Liste des fonctions secondaires et leurs descriptions

Nous avons également défini des contraintes : le positionnement de la caméra ne doit pas gêner les joueurs et l'application doit fonctionner pour une webcam placée dans une zone suffisamment grande pour être pratique. Nous avons aussi défini des contraintes au niveau du budget et du système d'exploitation, mais le fait de coder sur Python et avec OpenCV (contrainte imposée par l'encadrant) permet immédiatement de respecter celle-ci (+ : priorité normale, ++ : priorité élevée).

Fonction contrainte	Description	Priorité
1. Positionnement de la caméra	Pour ne pas gêner les joueurs, l'application doit fonctionner en plaçant la caméra dans la limite des zones rouges définies sur la figure 1 ci-dessous. De plus, la caméra doit être à une hauteur telle que l'angle entre la normale au plan de l'échiquier et le plan de l'échiquier soit entre 45 et 90°	++
2. Budget	Nul (caméra, plateau et pièces fournis)	+
3. Langage de programmation	Python 3 avec la librairie OpenCV	++
4. Système d'exploitation	L'application doit fonctionner sur tout système d'exploitation (Windows, MacOS, Linux)	+

Tableau 3: Liste des contraintes, leurs descriptions et leurs priorités

II.3 Modules

Des précédentes fonctions, nous avons divisé notre application en différents modules interagissant entre eux par des entrées et des sorties. Nous avons écrit les entrées et sorties au sens informatique, ce qui fait qu'un affichage n'est pas une sortie. Cependant, le module « Affichage » affiche en effet le plateau dans son état actuel, respectant la fonction secondaire 2. Les fonctions associées à chaque module sont précisées.

Module	Description	Entrée	Sortie	Fonction associée
Application	Le module principal (« main ») : permet de gérer l'instant n et n-1	-	-	*FP*3
Détection échiquier	Détecte les bords et les cases de l'échiquier à l'aide de la caméra	Image filmée	Position de l'échiquier sur l'image	FP1
Détection pièce	Détecte une pièce et sa couleur ainsi que sa position sur l'échiquier	Image filmée + Position de l'échiquier	Matrice qui représente l'état de l'échiquier	FP2
Détection mouvement	Détecte un changement de configuration de l'échiquier	Etat actuel de l'échiquier + Image filmée	Booléen qui représente la présence d'un mouvement	FP4
Traitement du mouvement	Déduit du changement de configuration le mouvement effectué	Booléen qui représente la présence d'un mouvement	Matrice qui représente le nouvel état	FP4
Affichage	Faire une interface qui affiche l'état de l'échiquier	Matrice qui représente l'état de l'échiquier	-	*FS*2

Tableau 4: Les modules utilisés et leurs différentes entrées/sorties

A partir de cette phase de définition du besoin, des fonctions et des différents modules à implémenter, nous avons structuré le programme dans le but de l'implémenter.

III. STRUCTURATION DU PROGRAMME

Rédacteur : Benjamin RENAULT, Mokhles BOUZAIEN, Venceslas ROULLIER, relecteur : Benjamin RENAULT, Mokhles BOUZAIEN, Venceslas ROULLIER.

III.1 Démarche

Ayant à disposition un code déjà existant écrit en Python 2 et dans une version antérieure d'OpenCV [1], notre démarche a été de reprendre ce code, de comprendre sa structure et son fonctionnement, et de mettre à jour les fonctions Python ou OpenCV qui ont changé de nom ou fonctionnent différemment. Nous avons en particulier repris les classes et

méthodes déjà proposées.

Le code est alors divisé en différentes classes, qui correspondent chacune à la réalisation d'un ou plusieurs modules, les correspondances sont décrites ci-dessous :

Module	Classe(s) correspondante(s)
Application	ChessCam, inputManager
Détection échiquier	boardFinder
Détection pièce	stateDetector, Cell
Détection mouvement	movementDetector
Traitement du mouvement	state
Affichage	chessCam

Tableau 5: Les classes correspondantes à chaque module

Il existe également une classe Cell qui permet de représenter une cellule. Celle-ci ne correspond pas à un module car elle est utilisée dans des modules pour représenter les case de l'échiquier.

III.2 Fonctionnement général de l'application

Le fonctionnement de l'application repose sur une boucle principale dans la classe ChessCam ou se situe le main, qui crée une instance de ChessCam et cherche un nouveau mouvement tant que l'utilisateur ne quitte pas (en appuyant sur une touche). ChessCam regroupe dans ses attributs différentes instances des autres classes, notamment inputManager qui permet de récupérer le flux vidéo, boardFinder qui permet de traiter l'image pour délimiter le plateau d'échecs, et MovementDetector qui détecte et stocke les mouvements.

La boucle va alors appeler getNextMove qui est une méthode de ChessCam qui recherche un mouvement (elle boucle tant qu'il n'y a pas de mouvement). Cette méthode va alors récupérer une image du flux vidéo, faire le traitement d'image nécessaire dans boardFinder et l'envoyer à MovementDetector pour que celui-ci détecte si il y a eu un mouvement, en créant une instance de la classe StateDetector qui détecte les cases et les stocke dans un dictionnaire. L'image de chaque case est alors comparée avec l'image précédente, et si il y a un changement suffisamment grand et que cela est le cas pour deux cases on en déduit qu'il y a eu mouvement. La pièce qui a été bougée est connue car on connaît l'état précédent du plateau, on le met ensuite à jour si le mouvement est valide (cela est géré par la classe state).

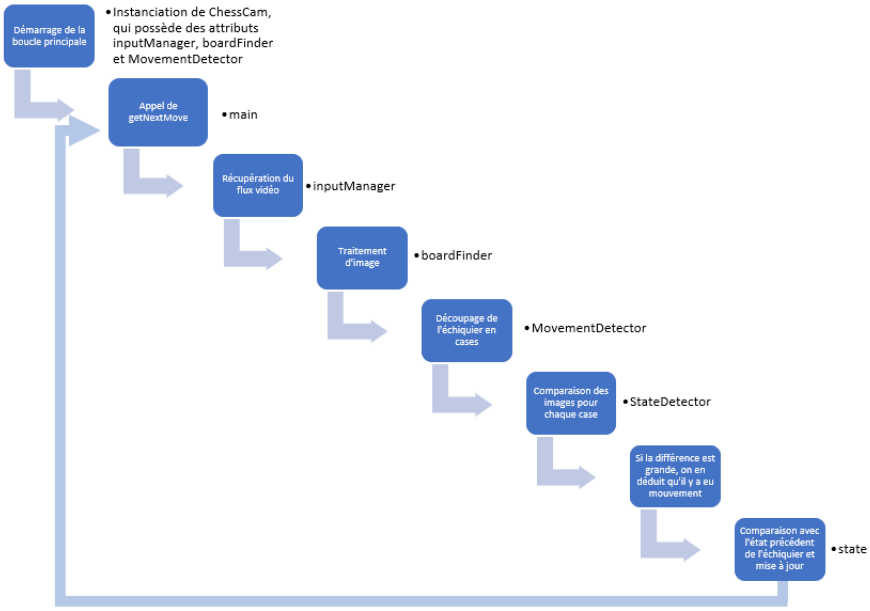


Illustration 4: Principe de fonctionnement du logiciel

III.3 Description des classes et des méthodes

On peut retrouver en annexe le diagramme de classes (Illustration 5)

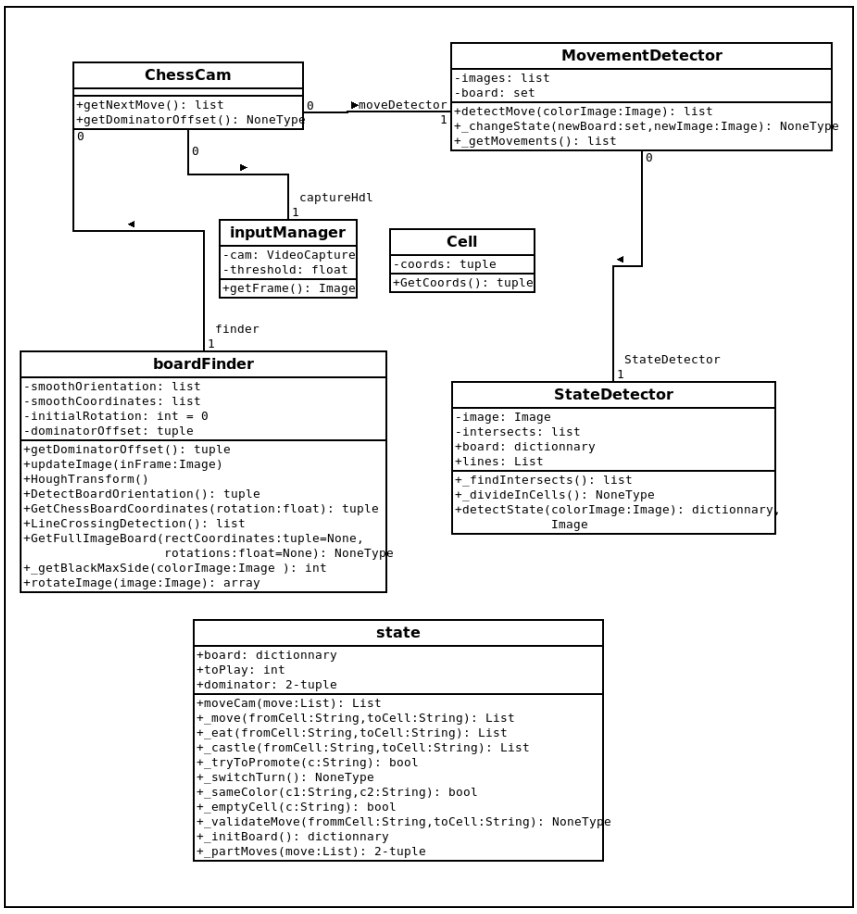


Illustration 5: Diagramme de classe

III.3.1 ChessCam

(a) Description générale :

ChessCam est la classe principale de l'application. Son fichier contient la fonction main, où l'on crée une instance de chessCam et de state puis on fait une boucle où l'on cherche un mouvement puis on valide (avec state) et affiche celui-ci.

Cette classe permet notamment de regrouper des instances des classes comme inputManager, boardFinder et MovementDetector qu'il met à jour dans sa méthode principale getNextMove qui récupère, analyse et renvoie le prochain mouvement effectué (qu'il soit valide ou non).

(b) Attributs :

captureHdl : inputManager

Instance d'inputManager qui gère le flux vidéo et permet notamment de recevoir une image de celui-ci grâce à la méthode getFrame

finder : boardFinder

Instance de boardFinder qui traite l'image pour détecter et ne garder que le plateau d'échecs.

moveDetector : movementDetector

Instance de MovementDetector qui stocke une image et les coordonnées des cellules sur le plateau pour pouvoir ensuite comparer à une nouvelle image et en déduire la présence d'un mouvement.

(c) Méthodes :

__init__()

Type de retour : *NoneType*

Initialise captureHdl , finder et moveDetector si la caméra est bien placée (sinon celui-ci boucle indéfiniment). Pour initialiser moveDetector, on traite une image récupérée de captureHdl grâce à finder.

getNextMove()

Type de retour : List

Récupère une image grace à captureHdl, la traite à l'aide de finder puis cherche si il y a eu un mouvement grâce à moveDetector. Si un mouvement est trouvé, il est renvoyé

getDominatorOffset() : 2-tuple

Définit de quel côté du plateau la caméra se trouve. Appelle tout simplement la méthode GetDominatorOffset de boardFinder (voir dans la section concernée).

III.3.2 MovementDetector

(a) Description générale :

Cette classe permet de stocker deux images et de les comparer cellule par cellule pour en déduire si il y a eu un mouvement. Elle stocke également les positions des cellules données par StateDetector.

(b) Attributs:

StateDetector : StateDetector

Instance de StateDetector permettant de traiter une image pour en déduire les coordonnées de chaque cellule.

images : List

Liste de deux images à comparer pour détecter la présence d'un éventuel mouvement

board : dictionnary

Dictionnaire ayant pour clé le nom de la case (une lettre puis un chiffre) et contenant des instances de Cell, soit les coordonnées des cellules correspondant à ces cases.

(c) Méthodes :

__init__(Image colorImage)

Type de retour : NoneType

Initialise une instance de StateDetector puis les attributs board et images à partir de la méthode StateDetector.detectState en envoyant l'image en paramètre du constructeur (qui est une image traitée par la classe boardFinder et représente seulement le plateau d'échecs). On initialise images avec deux fois la même image initiale car le constructeur est appelé lors de l'initialisation de l'application et il n'y a normalement pas de mouvement.

detectMove(Image colorImage)

Type de retour : List

Reçoit une nouvelle image traitée et appelle detectState de StateDetector pour recevoir le découpage de plateau correspondant et appelle la méthode _changeState qui met à jour les attributs permettant d'appeler et de retourner le résultat de _getMovements qui fait le traitement pour détecter la présence ou non d'un mouvement.

_changeState(dictionnnary newBoard, Image newImage) : NoneType

Met à jour les attributs board et images en remplaçant la deuxième image de images par la nouvelle image et en mettant à jour le board.

_getMovements() : List

Calcule la différence entre les deux images de l'attribut image en les découpant par cellule selon la division proposée par l'attribut board et remplit la liste mouvement si cette différence entre les deux images en une cellule est suffisamment grande. Le mouvement n'est renvoyé que si plus de deux cellules sont impactées. On met aussi à jour l'attribut images à la fin de la méthode pour ne pas détecter le mouvement deux fois (les deux images redeviennent les mêmes).

III.3.3 StateDetector

(a) Description générale :

Cette classe permet de traiter une image auparavant traitée par boardFinder, c'est-à-dire étant coupée pour ne représenter que le plateau d'échecs, pour découper le plateau en cellules représentées par la classe Cell.

(b) Attributs :

image : Image

Image représentant le plateau d'échecs

lines : List

Contient les lignes obtenues par la transformée de Hough [4] de l'image

intersects : List

Liste de coordonnées d'intersections des lignes de Hough obtenues par une transformée de Hough de l'image dans la méthode _findIntersects.

board : dictionnary

Dictionnaire ayant pour clé le nom de la case (une lettre puis un chiffre) et contenant des instances de Cell, soit les coordonnées des cellules correspondant à ces cases.

(c) Méthodes :

detectState(Image colorImage)

Type de retour : dictionnary, Image

Initialise image et intersects et appelle la méthode _divideInCells pour calculer l'attribut board, puis renvoie board et image. C'est la méthode qui est appelée par les autres modules pour obtenir les coordonnées des cases du plateau.

_findIntersects ()

Type de retour : List

Fait une transformée de Hough de l'attribut image et calcule les coordonnées des intersections des lignes obtenues.

_divideInCells ()

Type de retour : NoneType

Fait un traitement d'image pour diviser celle-ci en cases définies par l'attribut board. Il utilise notamment les coordonnées des intersections des lignes de Hough pour définir la taille de celles-ci.

III.3.4 boardFinder

(a) Description générale :

Cette classe permet de détecter, à partir d'une image, la présence d'un échiquier. A l'aide de la méthode updateImage, cette classe permet d'appliquer la transformée de Hough à l'image en entrée et puis déterminer l'orientation de l'échiquier le côté de chaque couleur.

(b) Attributs :

smoothOrientation : *Deque*

smoothCoordinates : Deque

initialRotation : Int

Représente la rotation de l'échiquier détecté en degrés, et peut prendre les valeurs : 0, -90, 90 ou 180.

dominatorOffset : Tuple

frame : Image

C’est une instance de l’image à traiter.

BoardOrientation : Tuple

C’est un 2-tuple qui contient les deux angles dominants de la transformée de Hough.

boardCoordinates : tuple

C’est les coordonnées des 4 points qui détermine le rectangle de l’échiquier.

laplacianImage : Image

Le résultat de la transformée de Laplace de l’image afin de détecter les contours.

lines : List

Une liste des lignes détectées par la méthode HoughLines représentées par des couples [rho, theta]

HSV : Image

Une représentation “teinte, saturation, lumière” de l’image.

intersects : List

Les intersections des ligne de Hough

(c) MÉTHODES :

__init__(Image inImage)

Type de retour : NoneType

Le constructeur de la classe boardFinder qui utilise updateImage pour trouver l’échiquier à partir d’une image donnée.

getDominatorOffset()

Type de retour : Tuple

Donne la valeur de l'attribut dominatorOffset

updateImage(Image inFrame)

Type de retour : NoneType

Effectue la transformée de Hough ainsi que la conversion HSV pour calculer ensuite l’orientation et les coordonnées de l’échiquier.

HoughTransform()

Type de retour : NoneType

C’est la méthode qui effectue la transformée de Hough. Il s’agit d’une technique qui isole les traits d’une forme particulière présente dans l’image. Dans notre cas, ça sert à détecter les ligne de l’échiquier.

DetectBoardOrientation()

Type de retour : Tuple

Après avoir vérifié la détection des ligne de Hough, cette méthode détermine l'orientation de l’échiquier à travers l’angle maximal des lignes de Hough.

GetChessBoardCoordinates(Float rotation) :

Cette méthode détermine les coordonnées des quatre points qui délimitent l’échiquier.

LineCrossingDetection() :

Pour déterminer les intersections des ligne de Hough.

GetFullImageBoard(Tuple rectCoordinates, Float rotations)

Type de retour : List

Applique l'homographie nécessaire afin de représenter l'échiquier sur l'image toute entière (à partir des coordonnées des coins et l'angle de rotation)

_getBlackMaxSide(Image colorImage)

Type de retour : Int

Retourne un indice qui représente le côté des pièce noires au début du jeu.

0 : top, 1 : left, 2 : bottom, 3 : right

rotateImage(Image image)

Type de retour : Image

Cette méthode effectue une rotation de l'image afin d'avoir la case A1 en haut à gauche.

III.3.5 inputManager

(a) Description générale :

Cette classe représente le flux vidéo capturé à l'aide de la webcam.

(b) Attributs :

cam : Image

La capture d'image effectuée à l'aide de la webcam

threshold : Float

Un nombre réel qui représente le seuil à atteindre pour

(c) Méthodes :

__init__()

Type de retour : NoneType

getFrame()

Type de retour : Image

Effectue une comparaison entre deux images capturées consécutivement.

La différence entre les images est comparée à un seuil.

On refait les traitement jusqu'à ce que la différence soit au-dessous du seuil.

III.3.6 Cell

(a) Description générale :

Cette classe a pour but de définir une case du plateau de jeu. Chaque case est définie par ses coordonnées.

(b) Attributs :

coords : tuple

Coordonnées de la case concernée.

(c) Méthodes :

__init__(coords)

Type de retour : NoneType

Initialise la cellule

__repr__()

Type de retour : *str*

Renvoie les coordonnées de la cellule sous forme d'une chaîne de caractères

GetCoords()

Type de retour : Cell

Accesseur, renvoie les coordonnées

III.3.7 State

(a) Description générale :

Cette classe décrit l'état actuel du plateau, et donne le joueur dont c'est le tour. Elle vérifie de plus les coups joués sont possibles.

(b) Attributs :

board : dictionnary

Dictionnaire ayant pour clé le nom de la case (une lettre puis un chiffre) et contenant des instances de Cell, soit les coordonnées des cellules correspondant à ces cases.

toPlay : int

Variable qui indique le joueur dont c'est le tour.

(c) Méthodes :

__init__(dominatorOffset)

Type de retour : NoneType

Initialise une partie, avec le tour donné au blanc, le plateau étant mis en place grâce à _initBoard().

moveCam(move)

Type de retour : list

Met à jour la classe State. Elle prend en paramètre la liste des mouvements possibles et renvoie le mouvement s'il est possible. S'il est impossible, elle retourne une exception.

_move(fromCell, toCell)

Type de retour : list

Définit ce qu'est un mouvement simple, sans prise. En cas de promotion, transforme le pion en dame.

_eat(fromCell, toCell)

Type de retour : list

Définit les mouvements avec prise, en prenant en compte la possibilité de la prise en passant .En cas de promotion, transforme le pion en dame.

_castle(fromCell, toCell)

Type de retour : list

Définit le roc (castle en anglais)

_switchTurn()

Type de retour : NoneType

Fait passer le tour d'un joueur à l'autre.

_sameColor(c1, c2)

Type de retour : boolean

Vérifie que deux pièces de couleur c1 et c2 sont de la même couleur ou non.

_emptyCell(c)

Type de retour : boolean

Vérifie qu'une case est vide.

_initBoard()

Type de retour : *Dict*

Initialise et renvoie le plateau, et y place chaque pièce de jeu à sa position initiale.

Il existe aussi deux librairies, mathUtils et chessUtils, dont les buts sont respectivement de faciliter les calculs mathématiques pour la reconnaissance du plateau de jeu et de faciliter la vérification de la possibilité d'un coup.

IV. VALIDATION

Rédacteur : Mokhles BOUZAIEN, relecteur : Benjamin RENAULT.

Les fonctions définies dans le cahier des charges fonctionnel sont bien respectées. En effet, le flux vidéo est bien capté et les outils d'analyse d'image fonctionnent pour détecter les lignes sur l'image et donc les cases présentes sur l'échiquier (FP1). Ensuite, le module de détection de mouvement des pièces fonctionne par comparaison de l'image actuelle du flux vidéo à celle de l'état précédent détecté de l'échiquier (FP4). Le logiciel déduit alors la présence ou non-présence d'une pièce, ainsi que sa couleur (FP2). Enfin, la déduction de la pièce ayant subi un mouvement et la représentation de l'échiquier fonctionnent (FP3). On peut alors retranscrire une partie d'échecs en captant le flux vidéo de celle-ci, c'est-à-dire le besoin à l'origine du projet.

Afin de quantifier les objectifs de l'application développée, une fiche de test contenant les différents scénarios d'utilisation, le résultat attendu et le bilan pour chaque scénario a été créée :

N° étape	*MO*	*RA*	*RO*
1	l'utilisateur lance l'application sur l'un des systèmes d'exploitations (Windows, Mac, Linux)	l'application se lance normalement	OK
2	l'utilisateur place la caméra dans la zone définie par la *FC*1	l'application détecte l'échiquier avec un message de confirmation	Détection des lignes et donc des cases de l'échiquier
3	les joueurs disposent les pièces dans la configuration initiale d'une partie d'échecs	l'application renvoie un dictionnaire représentant l'état de l'échiquier d'une manière exacte	OK
4	un joueur déplace une pièce	l'application renvoie un dictionnaire représentant le nouvel état de l'échiquier	OK

Tableau 6: Fiche de tests

V. CONCLUSION

Rédacteur : Venceslas ROULLIER, relecteur : Mokhles BOUZAIEN.

Le programme permet de détecter l'échiquier à l'aide de la webcam, ainsi que la présence de pièces et leur couleur. Le système déduit les mouvements effectués grâce à ce qui est filmé, et peut établir une liste des coups, sans aucune difficulté pour les coups spéciaux (roc et prise en passant). Le programme renvoie aussi une représentation de l'échiquier sous forme d'une matrice, permettant de visualiser les différents coups de manière plus aisée. De plus, nous sommes bien restés dans les contraintes imposées en utilisant Python et la librairie OpenCV, et en utilisant seulement la webcam fournie. La retranscription d'une partie peut ainsi se faire aisément, sans autre travail supplémentaire que le placement de la caméra, et ainsi sans perte de temps. Une amélioration à plus court terme serait la création d'une interface pour une meilleure prise en main de tout le système, permettant l'utilisation sans formation ou explications préalables.

Références Bibliographiques

Rédacteur : Mokhles BOUZAIEN, relecteur : Benjamin RENAULT.

[1] OLIVIER DUGAS. *Play Chess With a Webcam* [en ligne]. Disponible sur : <http://blogdugas.net/blog/2015/05/18/play-chess-with-a-webcam/> (Consulté le 20.05.2019).

[2] TANIO NEVED. *Transformez votre échiquier en bois en échiquier électronique et jouez contre tous les moteurs d'échecs UCI à l'aide d'une webcam* [en ligne]. Disponible sur <http://webcamchess.fr/> (Consulté le 20.05.2019).

[3] OpenCV [en ligne]. Disponible sur <https://opencv.org/> (Consulté le 20.05.2019).

[4] *Transformée de Hough* [en ligne]. Disponible sur : <https://fr.wikipedia.org/wiki/Transform%C3%A9e_de_Hough> (Consulté le 12.06.2019).

GLOSSAIRE

Rédacteur : Mokhles BOUZAIEN, relecteur : *Venceslas ROULLIER.*

Deque : un type de variable en Python comme les listes mais plus simple et rapide à manipuler.

Dict (dictionnaire) : un type de données très utile en Python. À la différence des séquences, qui sont indexées par des nombres, les dictionnaires sont indexés par des clés, qui peuvent être de n'importe quel type immuable ce qui facilite l'accès aux valeurs correspondantes.

FC (Fonction Contrainte) : D'après la norme AFNOR X50-151 : « Une contrainte c'est une limitation à la liberté de choix du concepteur-réalisateur d'un produit ».

FP (Fonction Principale) : C'est la fonction qui satisfait le besoin. Elle assure la prestation du service rendu. C'est la raison pour laquelle le produit a été créé.

FS (Fonction Secondaire) : Fonction qui facilite, améliore, ou complète le service rendu. Ce type de fonction ne résulte pas de la demande explicite du client, et n'est pas non plus une contrainte. Il s'agit de proposer au client des améliorations pour son produit et la qualité.

HSV (hue, saturation, lightness) : désigne des modèles de description des couleurs utilisés en graphisme informatique et en infographie, qui adaptent ces paramètres (teinte, saturation, luminosité).

MO : Mode Opératoire

NoneType : l'absence de sortie pour les méthodes en Python.

OpenCV : Open Source Computer Vision Library

RA : Résultat Attendu

RO : Résultat Obtenu

str (string) : type de données en Python qui représente des séquences immuables de points de code Unicode.

ANNEXES

Rédacteur : *Venceslas ROULLIER*, relecteur : Mokhles BOUZAIEN.

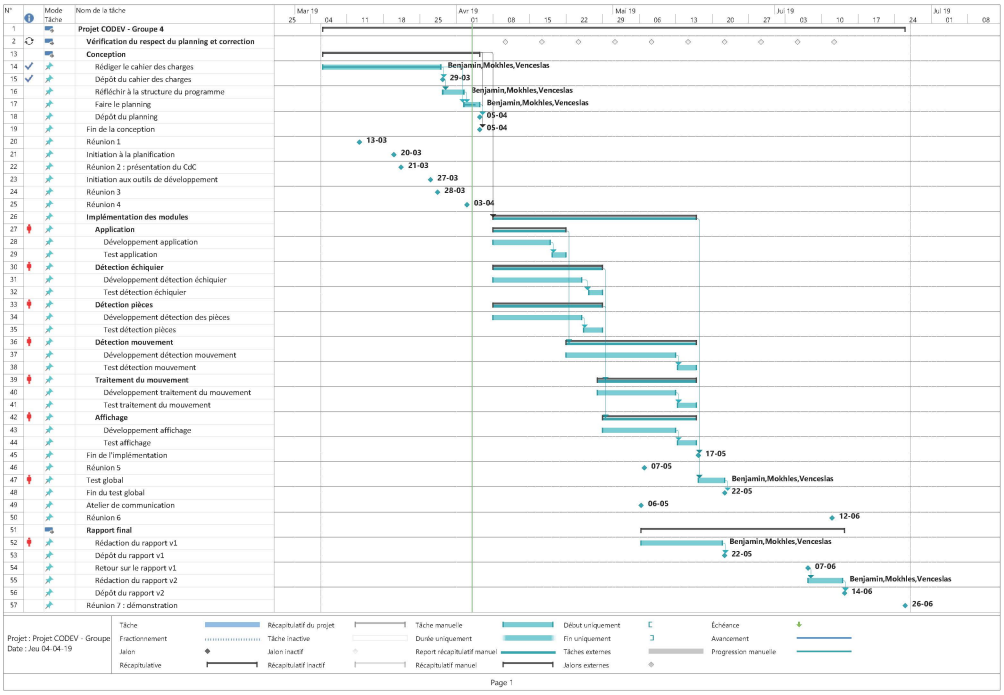
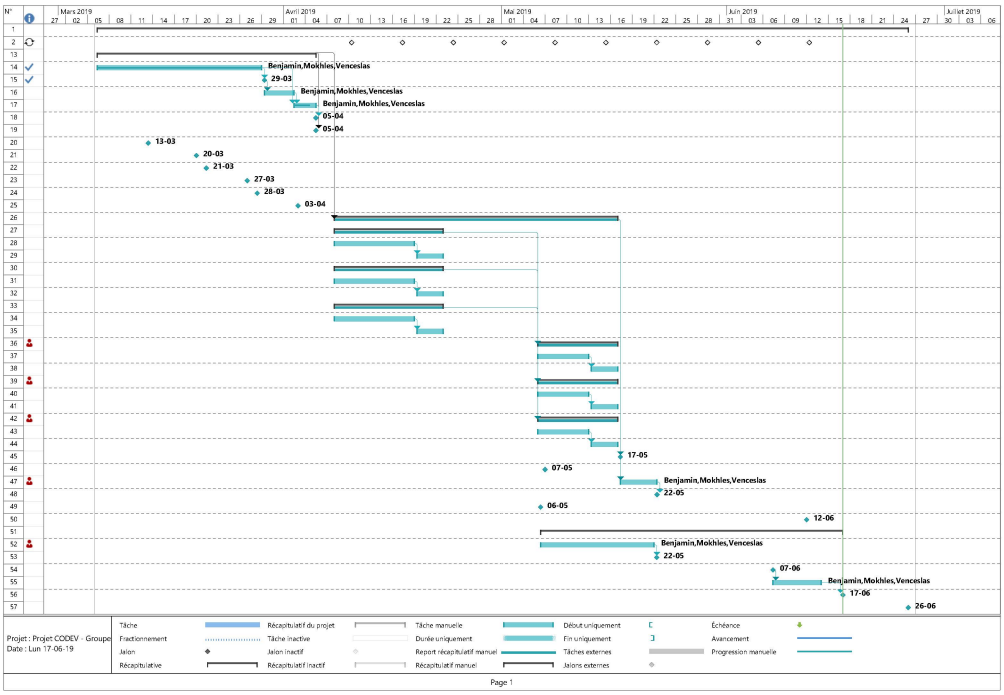


Illustration 6: Planning prévisionnel du projet



N°	Mode Tâche	Nom de la tâche	Durée	Début	Fin	Prédécesseurs	Noms ressources
1		**Projet CODEV - Groupe 4**	**70 jours?**	**Mer 06-03-19**	**Mer 26-06-19**		
2		**Vérification du respect du planning et correction**	**39.38 jours**	**Mer 10-04-19**	**Mer 12-06-19**		
13		**Conception**	**19.5 jours**	**Mer 06-03-19**	**Ven 05-04-19**		
14		Rédiger le cahier des charges	14.88 jours	Mer 06-03-19	Jeu 28-03-19		Benjamin,Mokhles,Venceslas
15		Dépôt du cahier des charges	0 jour	Ven 29-03-19	Ven 29-03-19	14	Mokhles
16		Réfléchir à la structure du programme	1.88 jours	Ven 29-03-19	Mar 02-04-19	15	Benjamin,Mokhles,Venceslas
17		Faire le planning	2.75 jours	Mar 02-04-19	Ven 05-04-19	16,14	Benjamin,Mokhles,Venceslas
18		Dépôt du planning	0 jour	Ven 05-04-19	Ven 05-04-19	17	
19		Fin de la conception	0 jour	Ven 05-04-19	Ven 05-04-19	13	
20		Réunion 1	0 jour	Mer 13-03-19	Mer 13-03-19		Benjamin,Mokhles,Tuteur,Venceslas
21		Initiation à la planification	0 jour	Mer 20-03-19	Mer 20-03-19		Benjamin
22		Réunion 2 : présentation du CdC	0 jour	Jeu 21-03-19	Jeu 21-03-19		Benjamin,Mokhles,Tuteur,Venceslas
23		Initiation aux outils de développement	0 jour	Mer 27-03-19	Mer 27-03-19		Benjamin,Mokhles,Venceslas
24		Réunion 3	0 jour	Jeu 28-03-19	Jeu 28-03-19		Benjamin,Mokhles,Tuteur,Venceslas
25		Réunion 4	0 jour	Mer 03-04-19	Mer 03-04-19		Benjamin,Mokhles,Tuteur,Venceslas
26		**Implémentation des modules**	**25.5 jours?**	**Lun 08-04-19**	**Jeu 16-05-19**	**13**	
27		**Application**	**9.75 jours**	**Lun 08-04-19**	**Lun 22-04-19**		**Benjamin**
28		Développement application	8 jours	Lun 08-04-19	Jeu 18-04-19		
29		Test application	1.75 jours	Ven 19-04-19	Lun 22-04-19	28	
30		**Détection échiquier**	**9.75 jours**	**Lun 08-04-19**	**Lun 22-04-19**		**Mokhles**
31		Développement détection échiquier	8 jours	Lun 08-04-19	Jeu 18-04-19		
32		Test détection échiquier	1.75 jours	Ven 19-04-19	Lun 22-04-19	31	
33		**Détection pièces**	**9.75 jours**	**Lun 08-04-19**	**Lun 22-04-19**		**Venceslas**
34		Développement détection des pièces	8 jours	Lun 08-04-19	Jeu 18-04-19		
35		Test détection pièces	1.75 jours	Ven 19-04-19	Lun 22-04-19	34	
36		**Détection mouvement**	**8 jours**	**Lun 06-05-19**	**Jeu 16-05-19**	**27**	**Benjamin**
37		Développement détection mouvement	5.38 jours	Lun 06-05-19	Dim 12-05-19		
38		Test détection mouvement	3.5 jours	Lun 13-05-19	Jeu 16-05-19	37	
39		**Traitement du mouvement**	**8 jours**	**Lun 06-05-19**	**Jeu 16-05-19**	**30**	**Mokhles**
40		Développement traitement du mouvement	5.38 jours	Lun 06-05-19	Dim 12-05-19		
41		Test traitement du mouvement	3.5 jours	Lun 13-05-19	Jeu 16-05-19	40	
42		**Affichage**	**8 jours**	**Lun 06-05-19**	**Jeu 16-05-19**	**33**	**Venceslas**
43		Développement affichage	5.38 jours	Lun 06-05-19	Dim 12-05-19		
44		Test affichage	3.5 jours	Lun 13-05-19	Jeu 16-05-19	43	
45		Fin de l'implémentation	0 jour	Ven 17-05-19	Ven 17-05-19	26	
46		Réunion 5	0 jour	Mar 07-05-19	Mar 07-05-19		Benjamin,Mokhles,Tuteur,Venceslas
47		Test global	2.75 jours	Ven 17-05-19	Mer 22-05-19	26	Benjamin,Mokhles,Venceslas
48		Fin du test global	0 jour	Mer 22-05-19	Mer 22-05-19	47	
49		Atelier de communication	0 jour	Lun 06-05-19	Lun 06-05-19		Benjamin,Mokhles,Venceslas
50		Réunion 6	0 jour	Mer 12-06-19	Mer 12-06-19		Benjamin,Mokhles,Tuteur,Venceslas
51		**Rapport final**	**26.25 jours**	**Lun 06-05-19**	**Lun 17-06-19**		
52		Rédaction du rapport v1	10.5 jours	Lun 06-05-19	Mar 21-05-19		Benjamin,Mokhles,Venceslas
53		Dépôt du rapport v1	0 jour	Mer 22-05-19	Mer 22-05-19	52	Mokhles
54		Retour sur le rapport v1	0 jour	Ven 07-06-19	Ven 07-06-19		
55		Rédaction du rapport v2	4.38 jours	Ven 07-06-19	Jeu 13-06-19	54	Benjamin,Mokhles,Venceslas
56		Dépôt du rapport v2	0 jour	Lun 17-06-19	Lun 17-06-19	55	Mokhles
57		Réunion 7 : démonstration	0 jour	Mer 26-06-19	Mer 26-06-19		Benjamin,Mokhles,Tuteur,Venceslas

Page 2

Illustration 7: Planning réel du projet

Ci-dessus le planning prévisionnel et le planning réel du déroulement du projet.

Globalement, nous avons pu respecter notre planning, malgré un facteur perturbant. En effet, durant les deux semaines passées à Nantes, nous n'avions pas réussi à trouver les bonnes conditions de travail permettant d'avancer le projet ce qui traduit le gap entre le 22/04 et le 05/05.