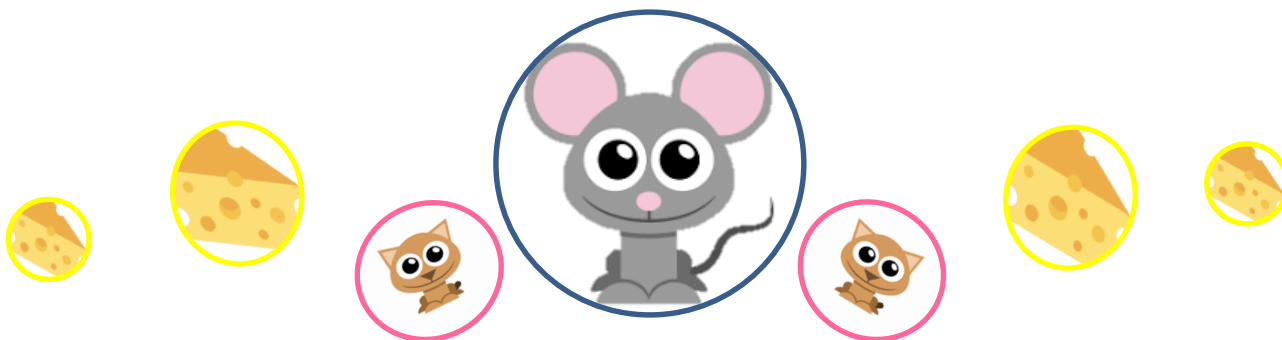




2ème Année Master Informatique, Semestre 1
Option : Systèmes informatiques Intelligents (SII)

Module : Vision Par Ordinateur



MINI JEU

Monôme :
- OUHOCINE Sarah

Professeur :
- M^r L.ABADA

Groupe :
- 3

SOMMAIRE

Introduction générale	
1 – Vision par ordinateur	
1.1 Principe de la vision par ordinateur	
1.2 Les domaines d'applications de la vision par ordinateur	
1.3 La détection et le suivi d'objet dans la vision par ordinateur	
1.4 Quelques méthodes pour la détection et de suivi d'objet	
1.4.1 La couleur	
1.4.2 Le mouvement	
1.4.3 L'histogramme	
1.4.4 La comparaison de mouvement	
2 – Description du projet	
3 – Réalisation du projet	
3.1 Méthode choisie pour la détection et le suivi de l'objet	
3.2 Environnements matériels	
3.3 Environnement logiciels	
3.3.1 Langage de programmation utilisé	
3.3.2 Bibliothèque utilisée pour le traitement d'image	
3.3.3 Bibliothèque utilisée pour la réalisation de l'interface graphique	
3.4 Améliorations	
3.5 Fonctions utilisées avec explications	
4 – Interface	
5 – Fonctionnement du jeu	
5.1 Configuration du jeu	
5.2 Déroulement du jeu	
5.3 Fin du jeu	
Conclusion générale	

INTRODUCTION GENERALE

La vision par ordinateur (aussi appelée vision artificielle ou vision numérique ou plus récemment vision cognitive) est une branche de l'intelligence artificielle dont le principal but est de permettre à une machine d'analyser, traiter et comprendre une ou plusieurs images prises par un système d'acquisition telle que la caméra ...

Plus précisément, le but est de permettre à une machine de comprendre ce qu'elle voit, lorsqu'on la connecte à une ou plusieurs caméras. Elle peut servir entre autre à la reconnaissance de formes, de couleurs, de distances, ou de mouvements qui consiste à reconnaître une de ces dernières dans une image après l'avoir enregistrée.

Dans ce projet, nous allons suivre sur une vidéo un objet en utilisant les couleurs, i.e. Nous allons utiliser la reconnaissance de la couleur afin de pouvoir suivre un objet en mouvement.

1 – VISION PAR ORDINATEUR

1.1 Principe de la vision par ordinateur

Le principe de la vision par ordinateur consiste à prendre une ou plusieurs images d'un même objet à l'aide d'une caméra. La ou les images numérisées sont ensuite traitées par un logiciel spécifique, permettant de détecter la conformité de l'objet par rapport à l'objet attendu, de prendre une décision par rapport à la couleur, à la forme, ...

1.2 Les domaines d'applications de la vision par ordinateur

En tant que discipline technologique, la vision par ordinateur cherche à appliquer ses théories et ses modèles à différents systèmes. Quelques exemples de systèmes d'application de la vision par ordinateur :

- ✚ Les bibliothèques numériques :
 - Acquisition (du papier/vidéo vers le numérique).



- ✚ Imagerie aérienne et spatiale :
 - Ressources naturelles et humaines.
 - Surveillance.
 - Météorologie.



- ✚ Industrie :
 - Contrôle non destructif.
 - Inspection et mesures automatiques.
 - Vision robotique.



- ✚ Médecine :
 - Cytologie.



- Tomographie.
- Echographie.
- ✚ Sciences :
 - Interventions en milieu confiné.
 - Astronomie, Robotique mobile.
 - Microscopie électronique, Biologie.
- ✚ Art, communication et Jeux vidéo :
 - Télévision, vidéo et jeux vidéo.
 - Photographie, Edition.
 - Transport information visuelle, archivage.
- ✚ Domaine militaire :
 - Surveillance.
 - Guidage automatique et poursuite d'engins.
 - Topographie



On considère comme sous-domaines de la vision par ordinateur **la reconstruction de scène, la détection d'événements, la détection d'intrusion, le match moving, l'estimation de mouvement, la reconnaissance et le suivi d'objets**, ainsi que certaines formes d'apprentissage automatique, d'indexation, de restauration d'image, etc...

1.3 La détection/reconnaissance et le suivi d'objet dans la vision par ordinateur :

La détection et le suivi d'objets en temps réel est une problématique difficile qui se pose dans un grand nombre d'applications de traitement d'images comme l'interaction homme-machine, la surveillance civile et militaire, la réalité virtuelle, l'analyse de mouvement humain et objet, ou encore la compression d'images. Cette difficulté est accentuée dans les environnements sans contraintes où le système de suivi devra s'adapter à la variabilité importante des objets, aux variations de couleur, nuance et luminosité, aux occlusions (partielles ou totales) ainsi qu'aux problèmes de détection de mouvements, formes (géométriques ou autres), distances, histogrammes ...

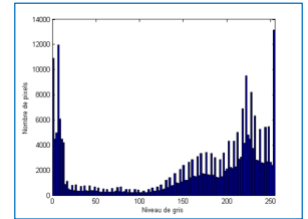
1.4 Quelques méthodes pour la détection et le suivi d'objet :

1.4.1 - **La couleur** : En évaluant les différentes couleurs RGB des pixels, si cette valeur est supérieure à une moyenne ou à une certaine valeur alors l'objet est détecté.



1.4.2 - **Le mouvement** : En comparant les valeurs des pixels entre le photogramme « background » et le photogramme actuel. Si la moyenne est supérieure alors le mouvement est détecté comme étant un objet.



1.4.3 - **L'histogramme** : En comparant l'histogramme de l'objet à reconnaître avec celui d'une particule. Si la moyenne est supérieure à un seuil alors la particule est détectée comme étant l'objet recherché



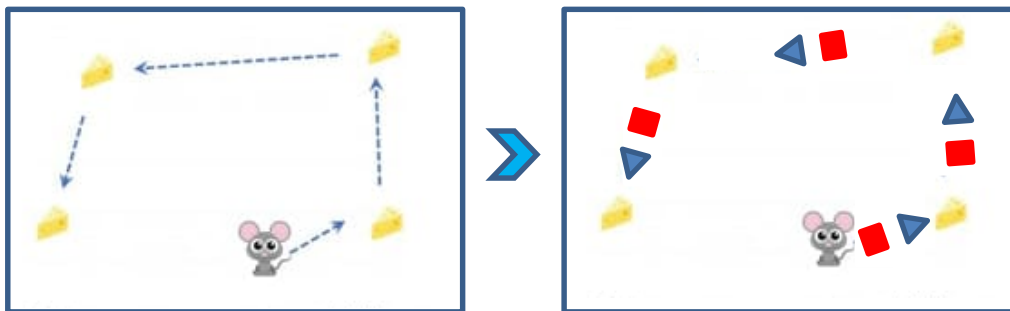
1.4.4 - **La comparaison de mouvement** : Il s'agit comme pour le mouvement de comparer le photogramme « background » avec le photogramme actuel. Puis de comparer les différences avec une image référence.

2 – DESCRIPTION DU PROJET

Création d'un petit jeu qui consiste à guider un objet vers des points objectifs générés de façon aléatoires, dans notre cas on a remplacé l'objet par une souris et les points objectifs par des fromages.

Le principe est de guider la souris vers tous les fromages avec une flèche dessinée sur une feuille blanche (pour garantir un fond blanc), la flèche a la forme   , la caméra donc à chaque fois aura la capacité de détecter les 2 couleurs différentes de la flèche, une fois la flèche détectée, la souris se déplace dans la direction correspondante tel que à chaque fois la position initiale de la souris c'est le début de la flèche qui est représenté par un **carré** (couleur rouge) et sa destination ou position cible c'est la tête de la flèche qui est représentée par un **triangle** (couleur bleue).


Une fois que la souris mange tous les fromages, la partie sera terminée.

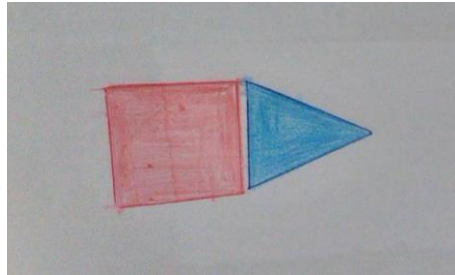


3 – REALISATION DU PROJET

3.1 Méthode choisie pour la détection et le suivi de l'objet :

Dans notre projet, Nous avons choisi pour comme méthode pour la détection et le suivi de la direction de la flèche : **la reconnaissance des couleurs**, i.e. les deux couleurs (couleur du **début** et de la **tête** de la flèche) et cela en évaluant les couleurs RGB des pixels, si ces dernières sont supérieures à une valeur seuil donnée alors la flèche sera détectée.

Objet à détecter : Notre objet à détecter c'est une flèche ayant la forme  dessinée sur une feuille blanche (pour garantir un fond blanc) et cela grâce à la caméra du PC (Vidéo en temps réel en mode niveau de gris) afin de faciliter et accélérer le temps de détection de la flèche.



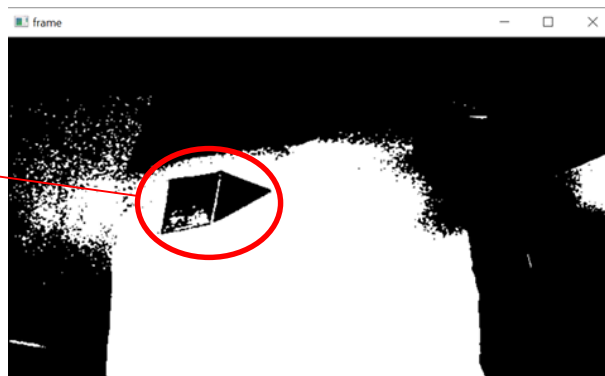
Fonctionnement de détection : lecture pixel par pixel de la flèche en (RGB) rouge vert et bleu. Puis test sur le pixel, suivant la couleur désirée. Le rouge, le vert et le bleu doivent être supérieurs ou inférieurs aux seuils imposés.

Test : condition de supériorité ou d'infériorité sur chaque couleur.

Paramètres : Les paramètres principaux sont les différents seuils des couleurs (RGB). Pour le rouge par exemple, les conditions sont $R > 120$, $G < 100$, $B < 100$, pour le bleu $R < 100$, $G < 100$, $B > 120$.

Résultat de détection : exemple d'un frame (capture prise lors du déroulement de la vidéo).

Flèche détectée



Dans ce cas la souris va se déplacer vers la droite selon la flèche détectée.

3.2 Environnements matériels :

Pour la réalisation de ce projet nous avons utilisé un pc **Dell** ayant les caractéristiques suivantes :

Un processeur : Intel(R) Core(TM) i5-8265 CPU @ 1.60GHz 1.80GHz.

RAM : 8 Go.

Type du système : Système d'exploitation 64 bits, processeur x64 (Windows 10 Professionnel).

WebCam : Ayant une qualité de vidéo 720p 16:9 30fps.



3.3 Environnement logiciels :

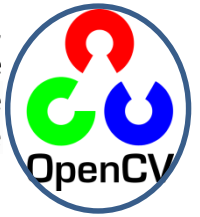
3.3.1 Langage de programmation utilisé :

Nous avons utilisé le langage **python** pour le développement de notre code car c'est un langage de programmation à usage général, qui est devenu très populaire surtout dans le traitement d'image en raison de sa simplicité et de la lisibilité du code. Il permet d'exprimer ses idées en moins de lignes de code sans réduire la lisibilité.



3.3.2 Bibliothèque utilisée pour le traitement d'image :

Nous avons opter pour **OpenCV** (Open Computer Vision) est une bibliothèque graphique libre, initialement développée par Intel, spécialisée dans le traitement d'images en temps réel. Cette bibliothèque est distribuée sous licence BSD, Actuellement, OpenCV prend en charge une grande variété de langages de programmation comme C ++, Python, Java, etc. et est disponible sur différentes plates-formes, notamment Windows, Linux, OS X, Android, iOS, etc.



Pour ce projet, nous avons configuré opencv pour python sous Windows et ceci en introduisant la commande suivante en cmd :

```
cmd Invite de commandes
C:\Users\Sara>pip install opencv-python_
```

3.3.3 Bibliothèque utilisée pour la réalisation de l'interface graphique :

Nous avons exploité la bibliothèque graphique **PyQt** : un module libre qui permet de lier le langage Python avec la bibliothèque **Qt** afin de créer des interfaces graphiques en Python.



Pour ce projet, nous avons configuré pyqt5 pour python sous Windows et ceci en introduisant la commande suivante en cmd :

```
cmd Invite de commandes
C:\Users\Sara>pip install pyqt5_
```

3.4 **Améliorations** : Nous avons fait 7 améliorations dans notre projet :

1 – **Ajout des obstacles** : nous avons ajouté des obstacles, ces derniers sont générés à des positions aléatoires tels que lorsque la souris rencontre un obstacle elle ne peut plus avancer, elle doit changer sa direction pour aller chercher des fromages.



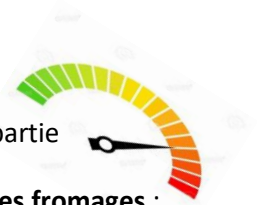
2 – **Ajouter un niveau de difficulté : (le chat)** : Nous avons ajouté un chat de plus que les obstacles, ce chat est généré à une position aléatoire tel que lorsque la souris rencontre un chat la partie est perdue, et on a un texte : **GAME_OVER !** qui s'affiche au joueur.

Notons aussi que lorsque la partie est gagnée i.e. lorsque la souris mange tous les fromages, et on a un texte : **CONGRATULATIONS !** qui s'affiche au joueur.



3 – Configuration automatique de la vitesse de la souris :

Nous avons donné la main au joueur de choisir la vitesse de la souris avant de lancer la partie



4 - Configuration automatique des dimensions de la souris, les obstacles ainsi que les fromages :

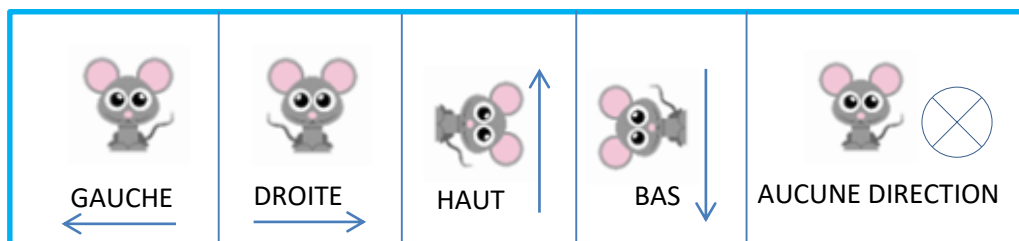
Nous avons donné la main au joueur de choisir la taille (Longueur et Largeur) des objets du jeu avant de lancer la partie (Souris, Obstacles, Fromages)

5 – Configuration automatique du nombre des obstacles et des fromages :

Nous avons donné la main au joueur de choisir le nombre d'obstacles et de fromages qu'il veut générer avant de lancer la partie

6 - **Automatisation du jeu** : Lorsque la souris ne détecte pas la flèche sur un fond blanc, elle se déplace dans des positions aléatoires (GAUCHE, DROITE, HAUT, BAS), des fois elle AUCUNE).

7 – **Dynamisme de la souris** : Lorsque la souris se déplace son positionnement s'adapte selon sa direction, nous avons 4 directions possibles (GAUCHE, DROITE, HAUT, BAS) :



3.5 Fonctions utilisées :

Dans cette partie on va expliquer les fonctions de notre code :

Fonction 1

Lit l'image du fromage et elle l'ajoute au tableau des fromages, puis elle génère une position aléatoire pour chaque fromage, en prenant en considération que la génération des positions des fromages ne seront pas égaux ou très proches (collées) des positions des obstacles, et cela en comparant la distance entre chaque fromage et les obstacles du jeu

```
def generateCheesses():
    for i in range(0, CHEESE_NUM):
        cheese = cv2.imread("cheese.png")
        cheese = cv2.resize(cheese, (CHEESE_WIDTH, CHEESE_HEIGHT))
        cheesses.append(cheese)
        randXPos = random.randint(0, BACKGROUND_WIDTH - CHEESE_WIDTH - 1)
        randYPos = random.randint(0, BACKGROUND_HEIGHT - CHEESE_HEIGHT - 1)
        j=0
        # La Contrainte pour que le fromage ne soit pas collé ou chevauché à un obstacle
        while j < OBSTACLE_NUM:
            if abs(obstaclePositions[j][0] - randXPos) < 31 or abs(obstaclePositions[j][1] - randYPos) < 31 :
                randXPos = random.randint(0, BACKGROUND_WIDTH - CHEESE_WIDTH - 1)
                randYPos = random.randint(0, BACKGROUND_HEIGHT - CHEESE_HEIGHT - 1)
                j+=1
            else:
                j+=1
        cheesePositions.append( (randXPos, randYPos) )
```


Fonction 2

Lit l'image de l'obstacle et elle l'ajoute au tableau des obstacles, puis elle génère une position aléatoire pour chaque obstacle

```
def generateObstacles():
    for i in range(0, OBSTACLE_NUM):
        obstacle = cv2.imread("obstacle.png")
        obstacle = cv2.resize(obstacle, (OBSTACLE_WIDTH, OBSTACLE_HEIGHT))
        obstacles.append(obstacle)
        randXPos = random.randint(0, BACKGROUND_WIDTH - OBSTACLE_WIDTH - 1)
        randYPos = random.randint(0, BACKGROUND_HEIGHT - OBSTACLE_HEIGHT - 1)
        obstaclePositions.append( (randXPos, randYPos) )
```

Fonction 3

Fonction qui prend en entrée la direction de la souris, et la positionne selon cette direction :

- #1 si la direction est à GAUCHE ou AUCUNE_DIRECTION laisse l'icône de la souris telle qu'elle est la tête de la souris à gauche et sa queue à droite
- #2 si la direction est à DROITE inverser l'icône de la souris de telle sorte que la tête de la souris sera à droite et sa queue sera à gauche (image miroir)
- #3 si la direction est en BAS Rotation de l'icône 90 degrés dans le sens contraire des aiguilles de la montre de telle sorte que la tête de la souris sera en bas et sa queue sera en haut
- #4 si la direction est en HAUT Rotation de l'icône 90 degrés dans le même sens des aiguilles de la montre de telle sorte que la tête de la souris sera en haut et sa queue sera en bas.

```
def changeDirection(direction):
    global RAT_HEIGHT, RAT_WIDTH, RAT_DIRECTION, actualRat, defaultRat

    #si GAUCHE ou PAS_DIRECTION : Laisser l'icone de la souris telle qu'elle est.
    if direction == RatDirection.LEFT:
        RAT_DIRECTION = RatDirection.LEFT
        RAT_WIDTH = len(actualRat[0])
        RAT_HEIGHT = len(actualRat)
        actualRat = defaultRat
        return actualRat
    elif direction == RatDirection.NO_DIRECTION:
        RAT_DIRECTION = RatDirection.NO_DIRECTION
        actualRat = defaultRat
        return actualRat
    #si HAUT : Rotation de l'icone 90 degrés dans le meme sens des aiguilles de la montre .
    elif direction == RatDirection.UP:
        RAT_DIRECTION = RatDirection.UP
        newRat = cv2.rotate(defaultRat, cv2.ROTATE_90_CLOCKWISE)
        RAT_WIDTH = len(newRat[0])
        RAT_HEIGHT = len(newRat)
        return newRat
    #si DROITE : inverser l'icone de la souris (image miroire)
    elif direction == RatDirection.RIGHT:
        RAT_DIRECTION = RatDirection.RIGHT
        newRat = cv2.flip(defaultRat, 1)
        RAT_WIDTH = len(newRat[0])
        RAT_HEIGHT = len(newRat)
        return newRat
    #si BAS : Rotation de l'icone 90 degrés dans le sens contraire des aiguilles de la montre
    elif direction == RatDirection.DOWN:
        RAT_DIRECTION = RatDirection.DOWN
        newRat = cv2.rotate(defaultRat, cv2.ROTATE_90_COUNTERCLOCKWISE)
        RAT_WIDTH = len(newRat[0])
        RAT_HEIGHT = len(newRat)
        return newRat
    else:
        raise Exception('INVALID RAT_DIRECTION ')
```

Fonction 4

Cette fonction assigne tous les objets (souris, fromages, obstacles, chat) qui sont des sous-matrices au background qui est la grande matrice (matrice principal) selon leurs positions et dimensions.

```
def assignObjectsToBackground():
    global game, RAT_HEIGHT, RAT_WIDTH, actualRat

    #copier le back_ground (pixels blancs) dans une variable globale appelee game
    game = np.copy(background)

    #Si tous les fromages ont Ã©tÃ© mangÃ©s,afficher l'image de CONGRATULATIONS et sortir
    if CHEESE_NUM == 0:
        game = youWon
        return

    #Si perdu (rencontre du chat),afficher l'image de GAME OVER et sortir
    if loose:
        game = youLose
        return

    #Poser tous les objets du jeux (souris, fromages, obstacles, chat) selon leurs positions dans la variables game
    try:
        game[ actualRatPos[1]: actualRatPos[1] + RAT_HEIGHT , actualRatPos[0]: actualRatPos[0] + RAT_WIDTH ] = actualRat
    except NameError:
        print(NameError)

    try:
        game[ actualCatPos[1]: actualCatPos[1] + CAT_HEIGHT , actualCatPos[0]: actualCatPos[0] + CAT_WIDTH ] = defaultCat
    except NameError:
        print(NameError)

    for i in range(0,OBSTACLE_NUM):
        try:
            game[ obstaclePositions[i][1] : obstaclePositions[i][1] + OBSTACLE_HEIGHT , obstaclePositions[i][0]: obstaclePositions[i][0] + OBSTACLE_WIDTH ] = obstacle
        except NameError:
            print(NameError)

    for i in range(0,CHEESE_NUM):
        try:
            game[ cheesePositions[i][1] : cheesePositions[i][1] + CHEESE_HEIGHT , cheesePositions[i][0]: cheesePositions[i][0] + CHEESE_WIDTH ] = cheeses[i]
        except NameError:
            print(NameError)
```

Fonction 5

#Cette fonction reconnaît les 2 couleurs de la flèche en "Mode_RGB", en effectuant un test sur les pixels (B,G,R) suivant la couleur qu'on veut reconnaître (Exemple de cas où la tête de la flèche est BLEUE et le début de la flèche est ROUGE) telle que ces pixels doivent être supérieures ou inférieures au seuil imposé.

#Puis calcule la somme des pixels pour les 2 couleurs, si les 2 sommes sont supérieures à un certain seuil la flèche sera détectée.

#Puis elle calcule la sommes des positions des pixels des deux couleurs pour calculer ensuite la moyenne de leurs positions en "divisant la sommes des positions sur la somme des pixels" afin de savoir la position de l'une des couleurs par rapport à l'autre afin de décider de la direction de déplacement de la souris.

```
def traitImage():

    global imageCopiedFlag ,actualRat

    height = len(frameCopy)
    width = len(frameCopy[0])

    #tant que l'image n'a pas encore Ã©tÃ© copiÃ©e dans le main : boucler
    while True:

        if not imageCopiedFlag:
            continue

        listOfRedPixels = []                #liste des pixels rouges
        listOfBluePixels = []               #liste des pixels bleus
        redPixelsPositionsSum = [0, 0]      #initialiser la somme des position des pixels rouges
        bleuPixelsPositionsSum = [0, 0]     #initialiser le somme des position des pixels bleus
```

```

#frameCopy[y, x, 0] --> Bleu |#frameCopy[y, x, 1] --> Vert| #frameCopy[y, x, 2] --> Rouge
for y in range(0, height):
    for x in range(0, width):
        #detection de la couleur rouge : R>120, B<100, G<100
        if frameCopy[y, x, 2] > 120 and frameCopy[y, x, 0] < 100 and frameCopy[y, x, 1] < 100:
            #Si oui, sommer les positions des pixels rouges pour calculer ensuite la moyenne de ces positions
            redPixelsPositionsSum[0] += x
            redPixelsPositionsSum[1] += y
            listOfRedPixels.append(frameCopy[y, x])

        #detection de la couleur bleue : R<100, B>120, G<100
        if frameCopy[y, x, 0] > 120 and frameCopy[y, x, 1] < 100 and frameCopy[y, x, 2] < 100:
            #Si oui, sommer les positions des pixels bleus pour calculer ensuite la moyenne de ces positions
            bleuPixelsPositionsSum[0] += x
            bleuPixelsPositionsSum[1] += y
            listOfBluePixels.append(frameCopy[y, x])

#si on trouve au moins 20 pixels rouges et 20 pixels bleus donc on a une flèche --> calcul moy
if len(listOfBluePixels) > 20 and len(listOfRedPixels) > 20 :
    redMoy = [0, 0] #initialiser la moyenne des positions des pixels rouges
    #calculer la moy en divisant la somme des positions des pixels rouges sur le nombre des pixels rouges
    redMoy[0] = redPixelsPositionsSum[0] / len(listOfRedPixels)
    redMoy[1] = redPixelsPositionsSum[1] / len(listOfRedPixels)

    bleuMoy = [0, 0] #initialiser la moyenne des positions des pixels bleus
    #calculer la moy en divisant la somme des positions des pixels bleus sur le nombre des pixels bleus
    bleuMoy[0] = bleuPixelsPositionsSum[0] / len(listOfBluePixels)
    bleuMoy[1] = bleuPixelsPositionsSum[1] / len(listOfBluePixels)

    #AFFICHAGE
    print("FLECHE DETECTEE !!!")
    print(" ROUGE : " + str(redMoy[0]) + " " + str(redMoy[1]))
    print(" BLEU : " + str(bleuMoy[0]) + " " + str(bleuMoy[1]))

    #Prise de décision de la direction

    #calculer la distance horizontale (diff entre les X) entre les moyennes de positions des pixels rouges et bleus
    xDef = abs(bleuMoy[0] - redMoy[0])
    #calculer la distance verticale (diff entre les Y) entre les moyennes de positions des pixels rouges et bleus
    yDef = abs(bleuMoy[1] - redMoy[1])

    #si la distance horizontale > la distance verticale Alors aller soit à gauche soit à droite (les Y sont très proches entre eux)
    #car les pixels sont à peu près sur la même ligne
    if (xDef > yDef): # RIGHT OR LEFT
        #si la moyenne des positions des pixels bleus > la moyenne des positions des pixels rouges (les X)--> ALLER A DROITE
        if (bleuMoy[0] > redMoy[0]):
            print(" ALLER A DROITE !!")
            actualRat = changeDirection(RatDirection.RIGHT)
        else:
            print(" ALLER A GAUCHE !!")
            actualRat = changeDirection(RatDirection.LEFT)
    #si la distance horizontale < la distance verticale Alors aller soit en haut soit en bas (les X sont très proches entre eux)
    #car les pixels sont à peu près sur la même colonne
    else: # UP OR DOWN
        #si la moyenne des positions des pixels bleus > la moyenne des positions des pixels rouges (les Y)--> ALLER EN BAS
        if (bleuMoy[1] > redMoy[1]):
            print(" ALLER EN BAS !!")
            actualRat = changeDirection(RatDirection.DOWN)
        else:
            print(" ALLER EN HAUT !!")
            actualRat = changeDirection(RatDirection.UP)
    else:
        print(" FLECHE NON DETECTEE, AUCUNE DIRECTION !! ")
        actualRat = changeDirection(RatDirection.NO_DIRECTION)
        #cv2.imshow('frameCopy3', frameCopy)
    imageCopiedFlag = False

```

Fonction 6

Cette fonction vérifie si la souris est collée (chevauchement des position de la souris et du fromage) si oui, il la supprime et décrémente le nombre de fromages

```

def eatCheeses():
    global cheesePositions, actualRatPos, RAT_WIDTH, RAT_HEIGHT, actualRat, CHEESE_NUM, RAT_DIRECTION

    #pour chaque position d'un fromage, calculer la distance entre ce dernier et la position de la souris
    for pos in cheesePositions:
        xDef = pos[0] - actualRatPos[0]
        yDef = pos[1] - actualRatPos[1]

        dist = math.sqrt(xDef * xDef + yDef * yDef)

        if dist < RAT_WIDTH:
            #suppression du fromage et décrémentation de son nombre
            cheesePositions.remove(pos)
            CHEESE_NUM -= 1

```

Fonction 7

Cette fonction vérifie si la souris est collée au chat (chevauchement des positions de la souris et du Chat) si oui, la partie est perdu.

```
def checkCat():
    global loose , actualCatPos, actualRatPos, RAT_WIDTH, RAT_HEIGHT, actualRat, CHEESE_NUM, RAT_DIRECTION

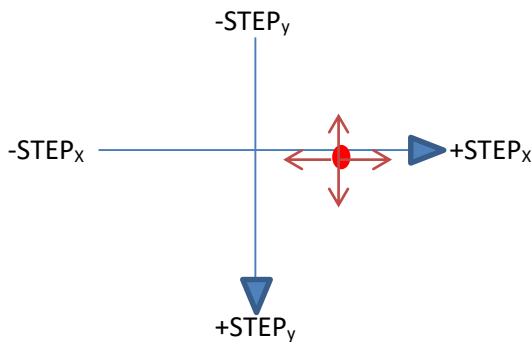
    xDef = actualCatPos[0] - actualRatPos[0]
    yDef = actualCatPos[1] - actualRatPos[1]

    dist = math.sqrt( xDef * xDef + yDef*yDef)

    if dist < RAT_WIDTH:
        loose = True
```

Fonction 8

cette fonction déplace la souris dans la direction détectée et cela en mettant à jour a chaque fois le vecteur de positions de la souris en tenant en compte que lorsque on croise des obstacle ou les extrémités (périmètre du jeu) la souris ne se déplace plus (stagnation) ainsi de la vitesse de la souris qui est enfaite une variable (**STEP**) contenant un entier qui est en fait le nombre de pixel avec lequel la souris doit se déplacer.



A GAUCHE = **POSITION ACTUELLE** - STEP_x

A DROITE = **POSITION ACTUELLE** + STEP_x

EN HAUT = **POSITION ACTUELLE** - STEP_y

EN BAS = **POSITION ACTUELLE** + STEP_y

```
def moveRat():
    global actualRatPos, RAT_DIRECTION, STEP, RAT_WIDTH, BACKGROUND_WIDTH, RAT_HEIGHT, BACKGROUND_HEIGHT
    direction = RAT_DIRECTION

    if direction == RatDirection.NO_DIRECTION:
        return

    #-----#
    #Sur X si je vais a gauche alors je dois
    elif direction == RatDirection.LEFT:
        #controle a chaque fois si ya intersection entre la nouvelle position de la souris et celle de l'obstacle
        for i in range(0, OBSTACLE_NUM):
            # s'ils ont le mm Y (intersection avec obstacle) et (calcul nvl pos de la souris sur x) si intersection avec obstacle<0
            if abs(obstaclePositions[i][1] - actualRatPos[1]) < RAT_HEIGHT and abs(obstaclePositions[i][0] + OBSTACLE_WIDTH - (actualRatPos[0] - STEP)) < STEP:
                #Si oui la souris ne pourra pas bouger a gauche
                return
            #sinon on la calcule le nouvelle pos sur X en tenant compte que cette position ne dâpassera pas l'extrimitâ gauche du jeu
            if actualRatPos[0] - STEP >= 0:
                actualRatPos = (actualRatPos[0] - STEP, actualRatPos[1])
        #-----#

    #Sur Y si je vais en haut alors je dois
    elif direction == RatDirection.UP:
        #controle a chaque fois si ya intersection entre la nouvelle position de la souris et celle de l'obstacle
        for i in range(0, OBSTACLE_NUM):
            # s'ils ont le mm X (intersection avec obstacle) et (calcul nvl pos de la souris sur Y) si intersection avec obstacle<0
            if abs(obstaclePositions[i][0] - actualRatPos[0]) < RAT_HEIGHT and abs(obstaclePositions[i][1] + OBSTACLE_HEIGHT - (actualRatPos[1] - STEP)) < STEP:
                #Si oui la souris ne pourra pas bouger vers le haut
                return
            #sinon on la calcule la nouvelle pos sur Y en tenant compte que cette position ne dâpassera pas l'extrimitâ haute du jeu
            if actualRatPos[1] - STEP >= 0:
                actualRatPos = (actualRatPos[0], actualRatPos[1] - STEP)
        #-----#

    #Sur X si je vais a droite alors je dois
    elif direction == RatDirection.RIGHT:
        #controle a chaque fois si ya intersection entre la nouvelle position de la souris et celle de l'obstacle
        for i in range(0, OBSTACLE_NUM):
            # s'ils ont le mm Y (intersection avec obstacle) et (calcul nvl pos de la souris sur x) si intersection avec obstacle<0
            if abs(obstaclePositions[i][1] - actualRatPos[1]) < RAT_HEIGHT and abs(obstaclePositions[i][0] - (actualRatPos[0] + RAT_WIDTH + STEP)) < STEP:
                #Si oui la souris ne pourra pas bouger vers la droite
                return
            #sinon on la calcule la nouvelle pos sur X en tenant compte que cette position ne dâpassera pas l'extrimitâ droite du jeu
            if actualRatPos[0] + STEP + RAT_WIDTH < BACKGROUND_WIDTH:
                actualRatPos = (actualRatPos[0] + STEP, actualRatPos[1])
        #-----#

    #Sur Y si je vais en bas alors je dois
    elif direction == RatDirection.DOWN:
        for i in range(0, OBSTACLE_NUM):
            if abs(obstaclePositions[i][0] - actualRatPos[0]) < RAT_HEIGHT and abs(obstaclePositions[i][1] - (actualRatPos[1] + RAT_HEIGHT + STEP)) < STEP:
                return
            if actualRatPos[1] + STEP + RAT_HEIGHT < BACKGROUND_HEIGHT:
                actualRatPos = (actualRatPos[0], actualRatPos[1] + STEP)
        #-----#
```

Fonction 9

Pour enregistrer la vidéo à partir de la caméra en niveau de gris et faire le traitement d'image en temps réel que la vidéo, si on veut arrêter l'exécution de la vidéo ou du jeu il suffit de cliquer sur 'q'

```
cpt = 0

#Thread pour le traitement d'image
t = Thread(target=traitImage, args=())
t.start()

#Lire a partir de la caméra
while(True):
    # Capture frame-par-frame
    ret, frame = cap.read()

    #Pour lire la vidéo en niveau de gris
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    ret,th1 = cv2.threshold(gray,100,255,cv2.THRESH_BINARY)
    frameCopy = np.array(frame)
    cv2.imshow('frame',th1)
    cv2.imshow('game',game)




    #clique sur q pour quitter
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# quand tout est fait, realiser la capture
cap.release()
cv2.destroyAllWindows()
```

4 - INTERFACE

Ci-dessous l'interface graphique de notre Mini Jeu:

Jeu ...

Nombre d'obs

Dimensions de l'obstacle

Longueur

Largeur

Dimensions de la souris

Longueur

Largeur

Vitesse

Nombre de frc

Dimensions du fromage

Longueur

Largeur

Lancer la partie

De gauche à droite :

1- Configuration des obstacles

2- Configuration de la souris

3- Configuration des fromages :

5 – FONCTIONNEMENT DU JEU

Notre jeu consiste en 3 parties :

5.1 >> CONFIGURATION DU JEU :

Cette phase se fait en remplissant les champs de l'interface graphique :

1- Configuration des obstacles :

Le joueur doit d'abord choisir dans la liste déroulante le nombre d'obstacles qu'il veut générer dans sa partie, puis il saisit les dimensions (la longueur et la largeur) de ces obstacles.

2- Configuration de la souris :

Le joueur doit saisir les trois champs concernant la souris et qui sont : ses dimensions (la longueur et la largeur), sa vitesse (le nombre de pixels de déplacement dans la partie).

3- Configuration des fromages :

Le joueur doit d'abord choisir dans la liste déroulante le nombre de fromages qu'il veut générer dans sa partie, puis il saisit les dimensions (la longueur et la largeur) de ces fromages.



Nombre d'obs	Dimensions de l'obstacle
10	Longueur: 25, Largeur: 25

Dimensions de la souris
Longueur: 80, Largeur: 80, Vitesse: 15

Nombre de frc	Dimensions du fromage
2	Longueur: 40, Largeur: 40

Lancer la partie

Exemple de configuration

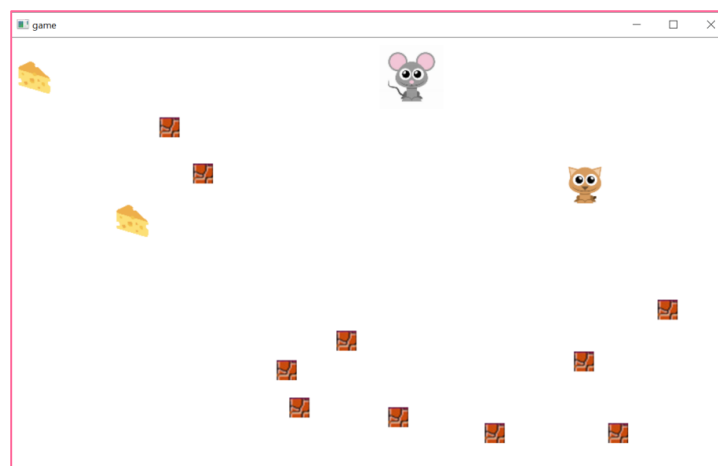
5.2 >> DEROULEMENT DU JEU :

Cette phase commence après avoir cliqué sur le bouton

Lancer la partie

Une fois que la partie est lancée, on obtiendra :

1 - La fenêtre principale du jeu : contenant tous nos objets configurés (obstacles, souris, fromages) ainsi que le chat qui est configuré automatiquement dans le code. (génération aléatoire).



Fenêtre principale du jeu pour la configuration précédente



Pour l'affichage des informations à propos de la détection et de la direction de la flèche.

Accédez aux paramètres pour activer Windows.

Console IPython Historique

Fins de ligne : CRLF Encodage : UTF-8 Ligne : 474 Colonne : 17 Mémoire : 73 %

Si la flèche est détectée dans la vidéo : elle affiche ➡ La direction de la flèche (GAUCHE, DROITE, HAUT, BAS) ainsi que la moyenne des positions (X, Y) des pixels

ROUGES et BLEUS

5.3 >> FIN DU JEU :

A la fin du jeu, la partie sera soit gagnée, soit perdue.

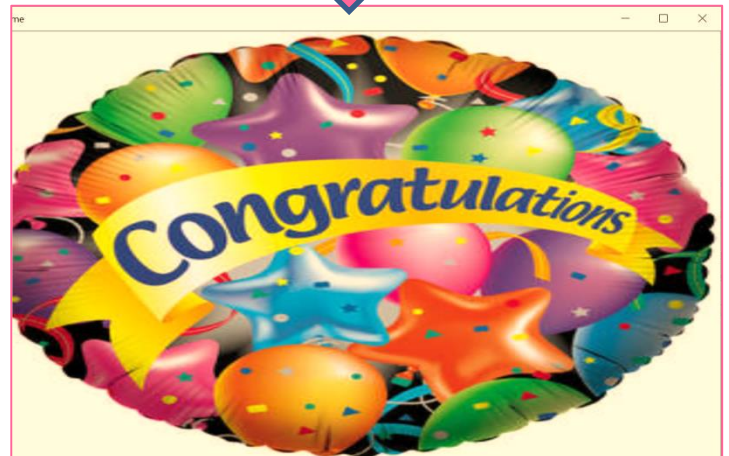
Partie Perdue : si la souris rencontre le chat avant qu'elle réussisse à manger tous les fromages générés.

On obtiendra ainsi cette fenêtre indiquant la **PERTE**.



Partie Gagnée : si la souris réussisse à manger tous les fromages générés sans avoir rencontré le chat.

On obtiendra ainsi cette fenêtre indiquant le **GAIN**.



Notons qu'on a pris en considération que les objets générés (obstacles, fromages, souris, chat) ne se chevauchent jamais quel que soit les nombres saisies dans la configuration.

CONCLUSION GENERALE

Ce Projet nous a permis de voir la simplicité de mettre en œuvre une méthode de détection et de suivi d'un objet à l'aide des couleurs, il suffit de reconnaître une couleur qui ressort par rapport au reste de l'image. En revanche cette méthode est limitée en raison de :

- il existe beaucoup de nuances de couleurs (256^3) il faut avoir alors une grande base de données puis choisir la couleur ou définir à chaque fois la couleur souhaitée ce qui sous-entend que ces valeurs soient connues.
- Si un autre objet possède la même couleur on ne peut pas les différencier, excepté s'il y a une grande différence de taille entre les 2 objets et que l'objet à reconnaître est le plus grand.
- Si notre objet change de couleur au cours de la vidéo, la couleur à reconnaître devra également être modifiée.

Mais aussi, le projet nous a permis d'appliquer nos connaissances théoriques dans le module de la vision par ordinateur et cela en exploitant la bibliothèque **OpenCV** qui a été créée à la base pour le traitement d'image.