Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Électronique et d'Informatique

Département Informatique

Master 2 — SII — Groupe 2

Module : Vision par Ordinateur

Rapport du TP :

Création d'un petit jeu qui implémente la détection des couleurs en utilisant OpenCV

Réalisé par :

BENAMARA SOUFIAN PRZEMYSLAW (201500008989)

DERARDJA MOHAMED ELAMINE (201500008274)

06 / 01 / 2020

1 Introduction et problématique

1.1 Vision par ordinateur

La vision par ordinateur est l'ensemble de techniques permettant à un dispositif numérique de "voir" (tel que l'être humain le fait). La compréhension de ce terme relève les défis techniques reconcentrés pour réaliser une telle tâche, notamment, la logique derrière le processus de la vision (analyse des contours et couleurs [récupération d'une image brute] et les décisions/sentiments résultats]). Cependant, bien que complexe, ce domaine ouvre de nombreuses possibilités à mettre en œuvre dans la vie réelle (par exemple, détections des objets), d'où l'importance de ce domaine, notamment dans un monde submergé d'images (réseaux sociaux ou caméras de surveillance).

1.2 Problématique

Le sujet abordé consiste à utiliser les fonctions d'OpenCV pour la création d'un petit jeu. Ce dernier est caractérisé par le guidage d'un objet vers des points objectifs (les deux situés dans une interface 2D). Le déplacement de l'objet est fait suivant le sens d'une flèche décrite par deux couleurs : début et fin, ainsi, il s'agit de les détecter avec la direction désignée par ces couleurs. Ces derniers doivent-être, de préférence, sur un fond blanc (ou plus généralement, fond uniforme ne contenant pas les couleurs à détecter).

2 Implémentation

Ce chapitre décrit la résolution pratique (programmation) de la problématique posée : Les outils utilisés et les points phares des traitements effectués pour réaliser l'application.

2.1 Environnement de travail

La machine (PC) utilisée pour le travail est dotée des caractéristiques suivantes :

- Processeur : Intel Core i5-7200U @ 2.50 Ghz (2 cœurs physiques et 2 logiques).
- Ram : 8.00 Go (DDR3, 1600 Mhz).
- Système d'exploitation : Windows 10 Professionnel (Version 1809) x64 bits.

Le développement est fait avec le langage Python (version : 3.7), en utilisant les packages suivants :

- OpenCV (version : 4.1.2) : Bibliothèque pour le traitement d'images (une surcouche de la version originale sous C++).
- NumPy (version : 1.16.5) : Dépendance d'OpenCV. Utilisée pour le calcul scientifique en général, et pour stocker une image (matrice de pixels) en particulier. Les différentes méthodes sont implémentées en C/C++, ce qui offre, par conséquence, une rapidité de calcul.
- PyQt5 (version : 5.13) : Bibliothèque pour réaliser toutes sortes d'interfaces graphiques (une surcouche de la version originale sous C++).

Le code lui-même est écrit sous Spyder (un IDE pour Python).

2.2 Développement de la solution

Dans ce qui suit, les différentes phases de développement de la solution conçue seront présentées avec les explications détaillées.

2.2.1 La flèche guidante

La première phase consiste à définir la direction de la flèche qui permettra, par la suite, de guider la souris dans le jeu. Cette flèche est déterminée à partir de deux couleurs (début et fin de flèche) configurées au préalable et récupérées directement à partir de la caméra.

2.2.1.1 Détection des couleurs

La déterminaison des couleurs représentantes de la flèche commence par la récupération d'une image (trame, *en Anglais :* frame) à partir de la caméra du PC. L'image reçue est sauvegardée sous format *"en couleur"* standard d'OpenCV : BGR (Blue, Green, Red), en effet OpenCV inverse les deux chaines de couleurs (Blue à la place de Red, et inversement).

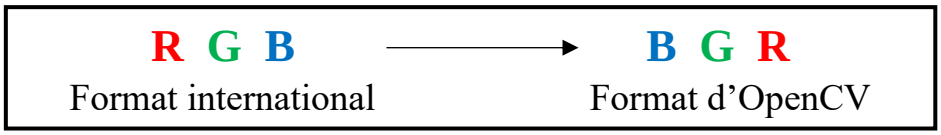

Figure qui montre la différence entre le format international et celui d'OpenCV

Après, la trame est convertie en HSV pour faciliter la détection des couleurs. En effet, le système HSV apporte une meilleure représentation de la perception des couleurs par l'œil humain en distinguant entre la teinte (hue), saturation et la valeur (value). Ainsi, ce système est le mieux adapté pour la problématique posée, puisqu'il s'agit de traiter des images réelles, issues directement de la caméra.

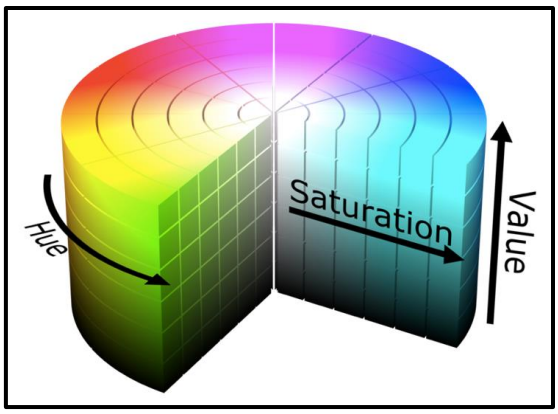

Figure qui montre la signification des composantes du systèmes HSV

Cependant, là aussi, OpenCV n'utilise pas la même plage de valeurs pour HSV que ceux de la norme internationale. Le tableau ci-dessous montre la différence entre les deux standards.

Composantes du HSV	Norme internationale		OpenCV	Conversion
	Intervalle	Unité		
Teinte (Hue)	[0 ; 360]	Degré (°)	[0 ; 180]	H_OpenCV = H_Inter / 2
Saturation	[0 ; 100]	Pourcentage (%)	[0 ; 255]	S_OpenCV = S_Inter * 2.55
Valeur (Value)	[0 ; 100]	Pourcentage (%)	[0 ; 255]	V_OpenCV = V_Inter * 2.55

Ainsi, des seuils minimums/maximums des valeurs HSV pour chaque couleur sont spécifiés. Ces seuils seront utilisés pour identifier les deux couleurs correspondantes au début/fin de la flèche.

Ensuite, l'image (représentée par une matrice 2D) est parcourue entièrement en récupérant les pixels (numéro de ligne et de colonne) satisfaisants les seuils définis précédemment. Le pseudo-algorithme ci-dessous montre le traitement effectué.

```
Algorithme récupérerCoordonnéesPixels;

Entrée : Une image (matrice de 'n' lignes, 'm' colonnes),
         seuilMinC1, seuilMaxC1, seuilMinC2, seuilMaxC2;

Sortie : Coordonnées des pixels de C1 et C2;

Var : i, j : entier;
      coordPixC1, coordPixC2 : tableau 2D de coordonnées (entiers : X et Y);

Début;
coordPixC1 ← [ ];   coordPixC2 ← [ ];
Pour i ← 0 à n Faire
  Pour j ← 0 à m Faire
    Si (image[i][j] >= seuilMixC1) et (image[i][j] <= seuilMaxC1)
      alors coordPixC1.ajouterElement(i, j);
      Sinon Si (image[i][j] >= seuilMixC2) et (image[i][j] <= seuilMaxC2)
        alors coordPixC2.ajouterElement(i, j);
    Fsi;
  Fait;
Fait;
retourner(coordPixC1, coordPixC2);
Fin;
```

Par la suite, sur la base des pixels récupérés précédemment, le *rectangle conteneur minimal* (appelé techniquement *AABB* [*Axis Aligned Bounding Box*]) est déduit. Cela consiste à définir l'air minimale du rectangle englobant tous les pixels d'une couleur donnée. Le calcul de cette surface nécessite deux paires de coordonnées : Le point minimal (MinimumX, MinimumY) et le point maximal (MaximumX, MaximumY). Les coordonnées d'un pixel (sur le repère orthonormé ayant comme centre le point haut, à gauche de l'écran) est la paire inverse de ses valeurs (le numéro de ligne désigne l'axe Y, la colonne désigne l'axe X).

Les données du rectangle sont calculées comme suit :

- MinimumX : Récupérer le numéro de la colonne du pixel dont la valeur est minimale (la colonne = l'axe X).
- MinimumY : Récupérer le numéro de la ligne du pixel dont la valeur est minimale (la ligne = l'axe Y).

- MaximumX : Récupérer le numéro de la colonne du pixel dont la valeur est maximale.
- MaximumY : Récupérer le numéro de la ligne du pixel dont la valeur est maximale.

Ainsi, ayant ces informations, la longueur et largeur du rectangle peuvent être définies et donc la forme dessinée. De plus, le centre de chacun des deux rectangles est calculé (il servira dans la déterminaison de la direction de la flèche) :

- CentreX = (MaximumX + MinimumX) / 2.
- CentreY = (MaximumY + MinimumY) / 2.

La figure ci-dessous illustre la totalité du processus.

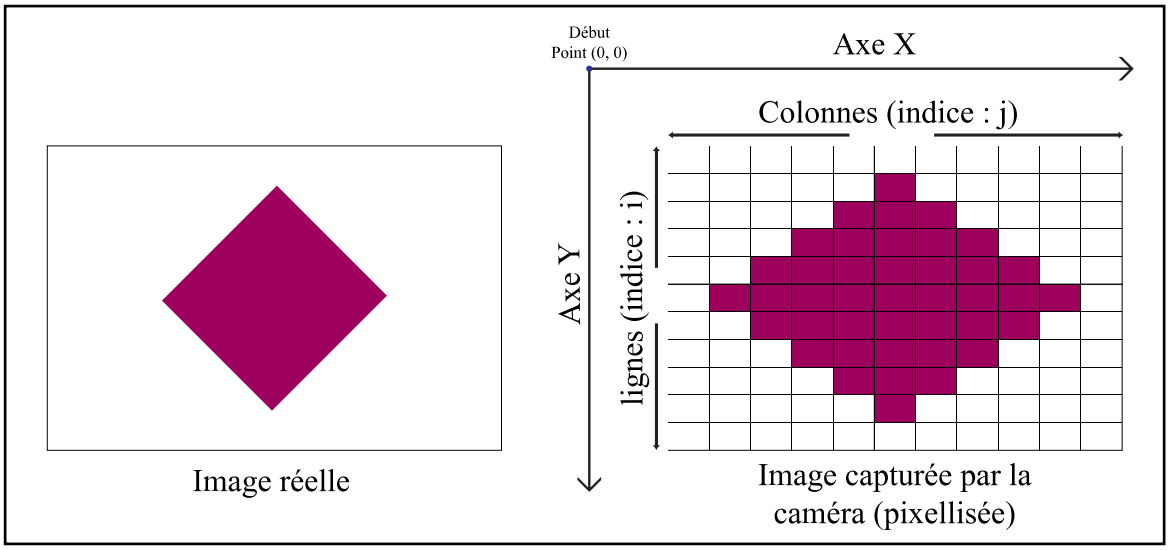

Figure qui montre l'image capturée par la caméra et la relation entre ses lignes/colonnes avec le repère orthonormé de l'écran (l'axe X et Y)

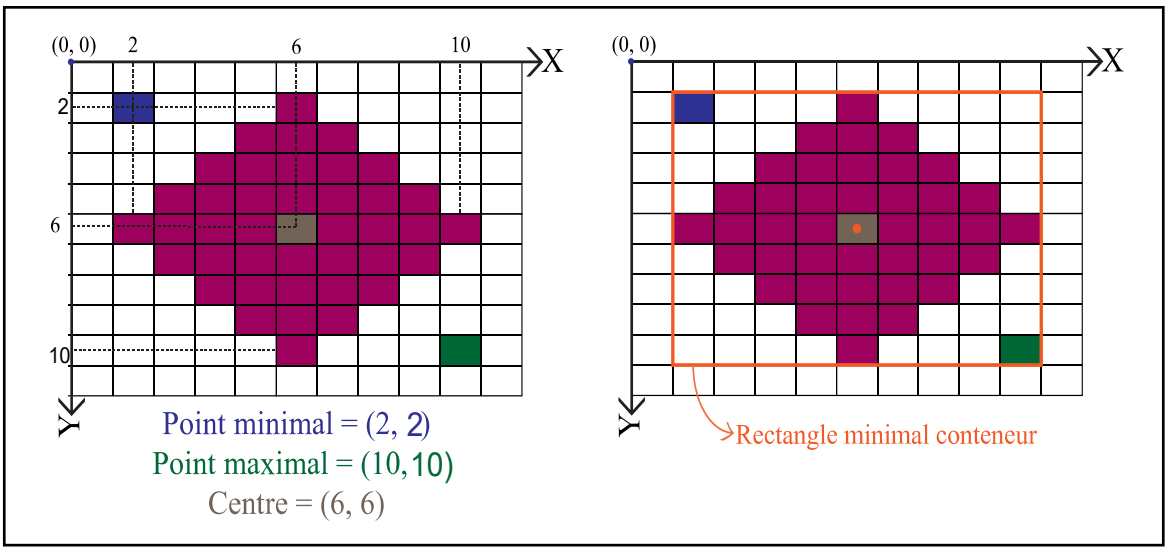

Figure qui montre la déterminaison du "rectangle conteneur minimal" d'une couleur à partir de la position de ses pixels

2.2.1.2 Direction de la flèche

La détection du sens de la flèche guidante débute par le calcul de l'angle absolu (sans signe) formé par les deux centres des deux rectangles déterminés précédemment. Pour cela, les propriétés d'un *triangle droit* sont utilisées en calculant le rapport entre le côté opposé sur le côté adjacent, le résultat est passé à la fonction tangente inverse ($\tan^{-1}$) pour récupérer l'angle (en valeur absolue). Le calcul est comme suit :

Centre de la couleur 1 (début de la flèche) : Coordonnées (X1, Y1)

Centre de la couleur 2 (fin de la flèche) : Coordonnées (X2, Y2)

Côté opposé (nominateur) = Y2 – Y1

Côté adjacent (dénominateur) = X2 – X1

Le rapport (la tangente) = Côté opposé / Côté adjacent

Angle = | $\tan^{-1}$ (le rapport) | (en degrés)

Les formules ci-dessus sont éclaircies par la figure ci-dessous.

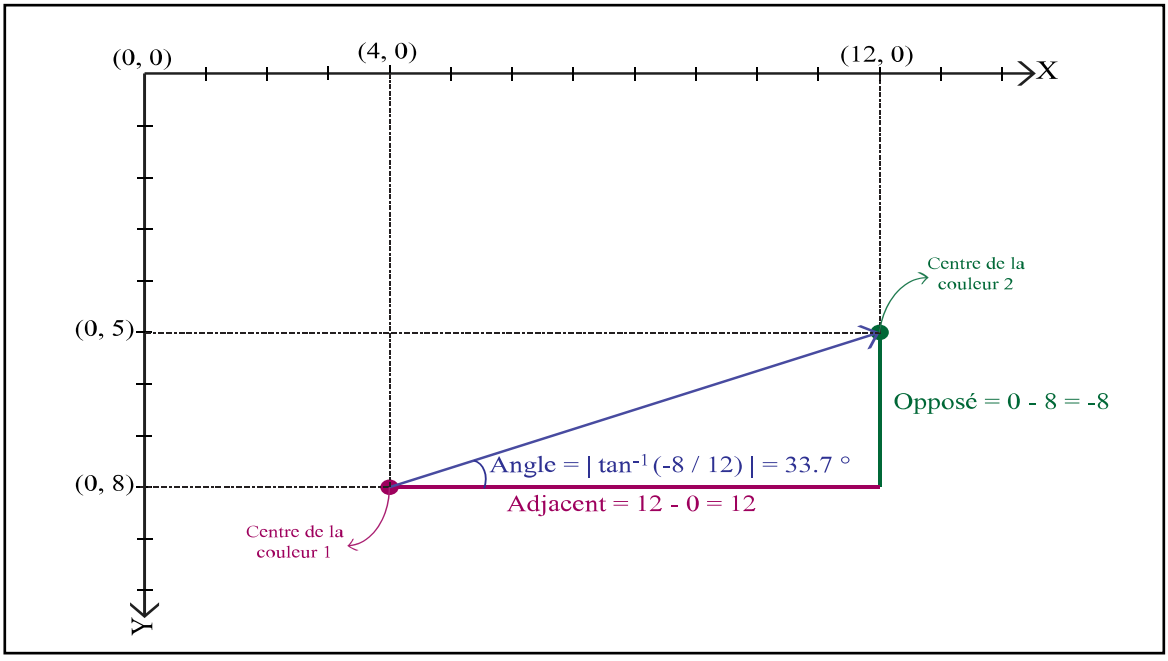

Figure qui montre la déterminaison de l'angle absolu formé par le centre du "rectangle conteneur minimal" de deux couleurs (dans l'axe orthonormé de l'écran)

Ensuite, l'angle réel sur le cercle trigonométrique est évalué. Pour le faire, un test des signes du nominateur et le dénominateur est fait pour se positionner dans le quart correct et effectuer les manipulations correspondantes. La figure ci-dessous explique ce traitement :

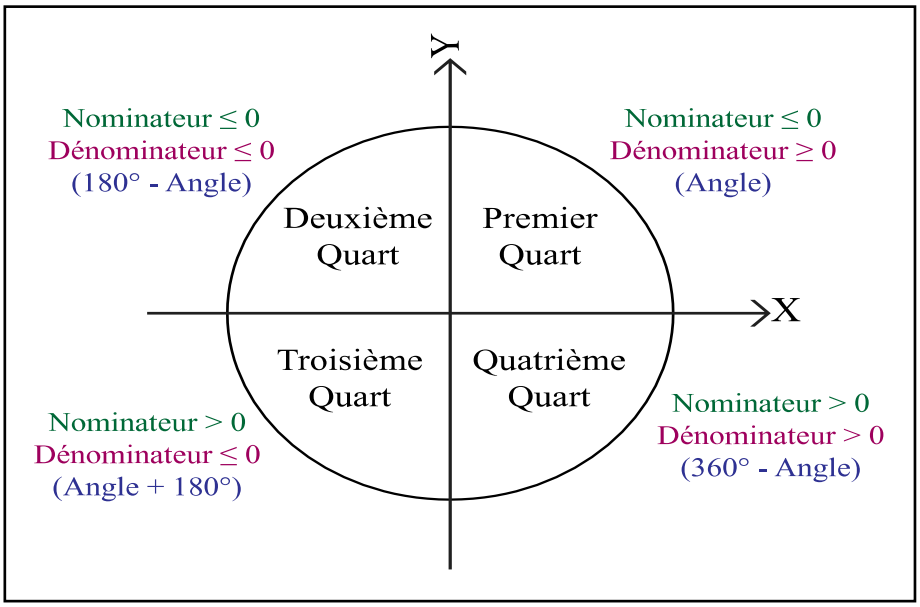

Figure qui montre le calcul de l'angle sur 360°

Ainsi, une fois l'angle réel déterminé il ne reste qu'à lui assigner la bonne direction (sur le système de huit directions, puisqu'un objet (la souris) peut se déplacer [dans une grille de pixels] dans 8 directions uniquement [au maximum, si aucune bordure n'est rencontrée]). De plus, un intervalle de manœuvre est attribué à l'angle (car, dans la pratique, l'orientation exacte vers un angle spécifique est difficile). Les détails ci-dessous.

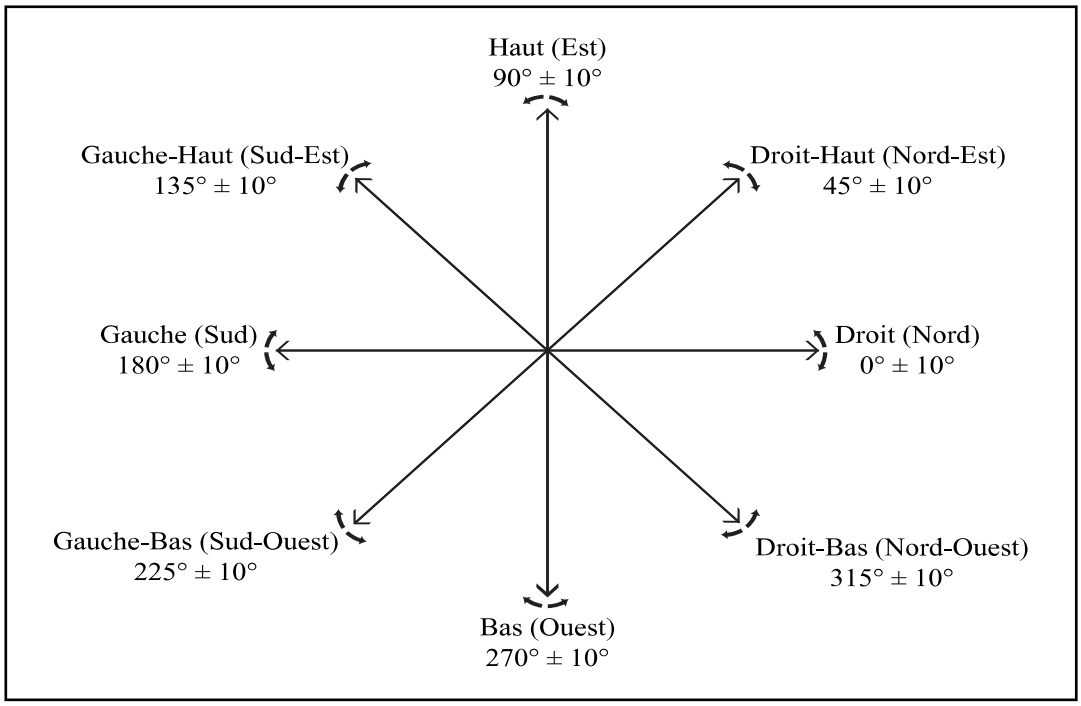

Figure qui montre la déterminaison de la direction de la flèche (sur les 8 positions géographiques)

2.2.2 Interface du jeu

La deuxième phase concerne le jeu lui-même. Les différents mécanismes seront présentés en détails : le but du jeu, le déplacement de l'objet et le principe de "collision".

2.2.2.1 Objectifs et contraintes

Le jeu proposé dans la problématique consiste à déplacer un objet (dans notre cas, une souris) dans une grille 2D (interface du jeu) et le diriger vers des points cibles (dans notre cas, des fromages), tout en évitant des obstacles (dans notre cas : des souricières). Les éléments du jeu sont présentés ci-dessous.

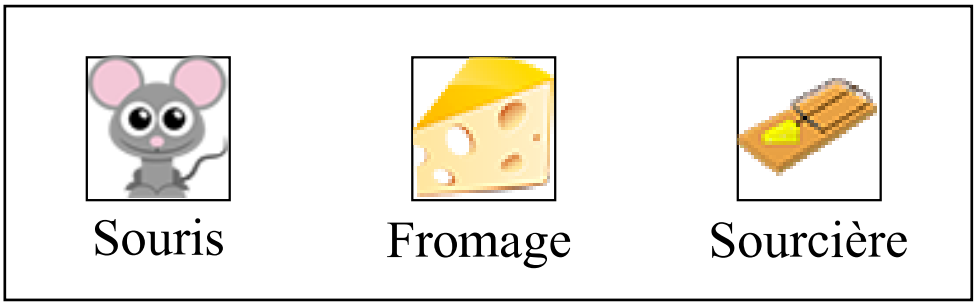

Figure qui montre les objets manipulés dans le jeu

Le sens de déplacement de la souris est récupéré suivant l'orientation de la flèche guidante (la section précédente). Ainsi, la position de la souris est modifiée (en incrémentant ou décrémentant sa coordonnée correspondante sur l'axe X ou Y dans l'écran [interface du jeu]) selon le sens détecté. Les détails sur la figure ci-dessous.

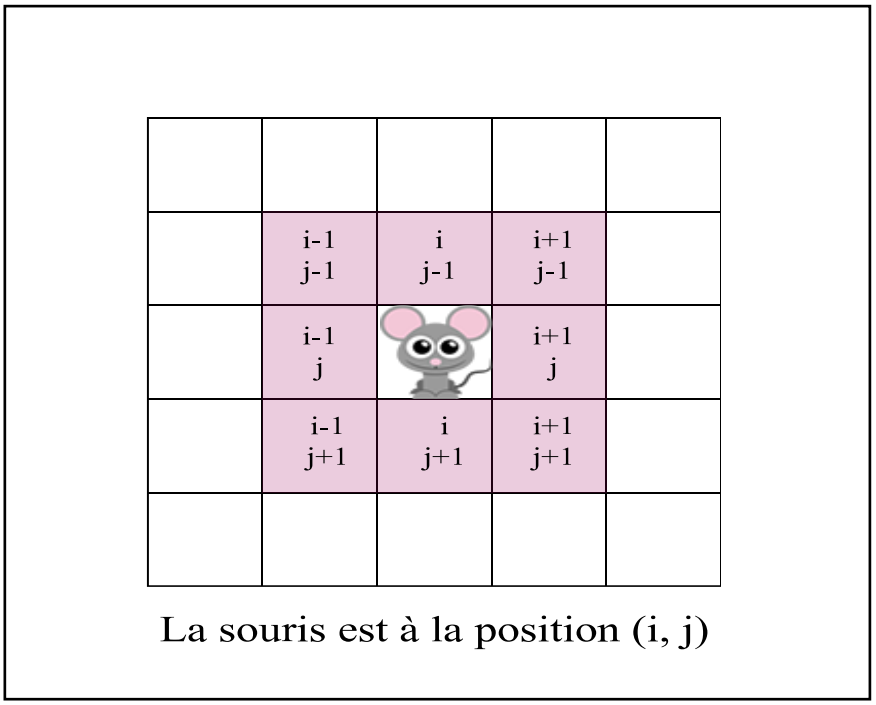

Figure qui montre les déplacements possibles de la souris (dans le cas parfait, aucune bordure n'est rencontrée)

Une partie est considérée "gagnée" si le joueur parvient à faire manger à la souris tous les fromages en évitant tous les obstacles. Dans le cas contraire (passer sur un obstacle), la partie est considérée "échouée".

2.2.2.2 Détection de collision

La détection du passage d'un objet sur un autre dans une surface 2D fait appel à la notion de "collision". Cette dernière consiste à déterminer s'il existe des pixels communs entre deux objets. La détection de collision passe par deux phases :

● Phase large : Cette phase permet de déterminer l'existence potentielle d'une collision réelle entre deux objets. Il s'agit de vérifier si une intersection (non-vide) existe entre les rectangles conteneurs minimaux de deux objets. Une telle vérification est simple à effectuer (peu coûteuse en temps de calcul).

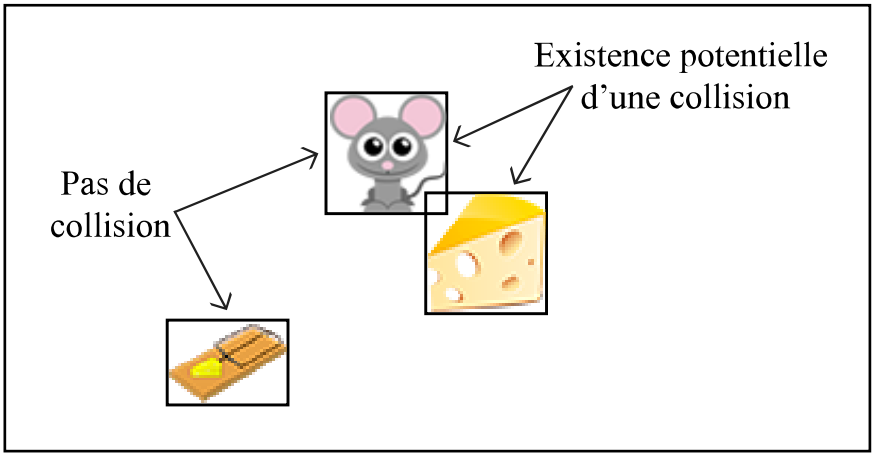

Figure qui montre la détection de collision dans la "phase large"

● Phase étroite : Successive à la phase large, une fois que cette dernière est vraie (existence d'une collision potentielle à cause du chevauchement des deux rectangles conteneurs minimaux), une vérification à pixel-près (*en Anglais :* pixel-perfect verification) est faite. Il s'agit de vérifier l'existence d'une intersection réelle entre deux objets (et non pas leurs pixels transparents uniquement). Cette vérification est coûteuse en temps de calcul (pour chaque pixel visible [non-transparent] de l'objet 1, vérifier s'il existe dans l'objet 2, et vice-versa), c'est pour cela qu'elle est précédée par la phase large (qui éliminera dès le début les cas où les objets sont loin les uns des autres).

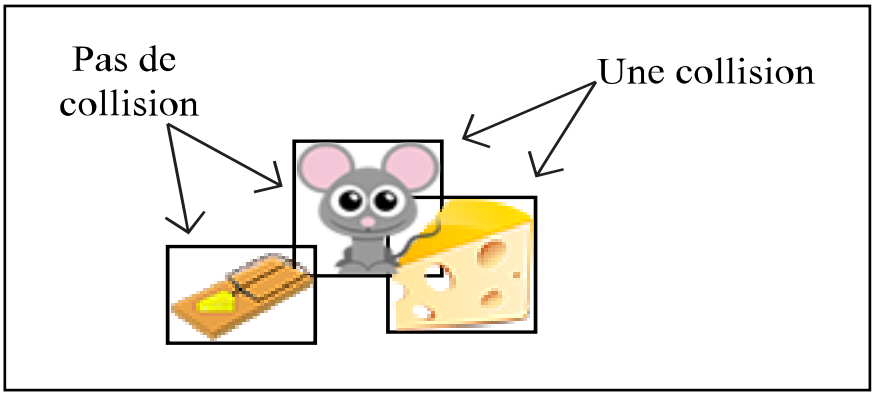

Figure qui montre la détection de collision dans la "phase étroite" (pixel-perfect, 2D collision detection)

3 Amélioration apportées

De plus des fonctionnalités de base, quelques améliorations ont été ajoutées au jeu pour le rendre plus intéressant. Elles se résument comme suit :

- Réglage des couleurs : La plage des couleurs au choix est extensible, un fichier XML modifiable est offert pour ajuster les seuils minimaux/maximaux (dans le système HSV d'OpenCV) des couleurs existantes (par défaut, il y'en a quatre) ou ajouter d'autres (suivant la syntaxe des balises du fichier).

- Réglage de la vitesse : La vitesse de déplacement de la souris peut être ajustée. Il s'agit de spécifier la valeur (de 1 à 5) du pas avec lequel la position actuelle sera modifiée. Ainsi, cette valeur sera utilisée (à la place du 1, par défaut) pour l'additionnement/retranchement au "i" ou "j" (coordonnées actuelles de la souris).

- Niveaux de difficulté : Au lieu d'une seule partie se répétant à chaque fois. Différentes parties dans plusieurs niveaux de difficultés sont proposées. Exactement, trois parties pour chaque difficulté (facile, moyen et difficile) ont été implémentées.

- Obstacles : Pour diversifier les parties du jeu, des obstacles ont été ajoutés. Il s'agit des sourcières que le joueur guidant la souris devra éviter.

4 Conclusion

À travers de cette problématique résolue, nous avons mis en pratique quelques aspects de la vision par ordinateur (précisément, la détection des couleurs) en utilisant une bibliothèque dédiée populaire : OpenCV. De plus, cela nous a permis de résoudre quelques problèmes algorithmiques en proposant des solutions adaptées (précisément, détection de la direction d'une flèche définie par deux points dans un repère orthonormé, et la collision 2D entre deux objets). Le résultat du travail est injecté dans un jeu simple permettant de mettre en valeur les notions abordées.

5 Annexe

5.1 Organisation du code-source

Le code-source de l'application développée est modulaire. Organisé comme suit :

- __main__.py : Point d'entrée de l'application (le fichier principal à exécuter).
- icons : Icônes du jeu (en format PNG).
- include : Corps de l'application (fonctions et classes).
  - colors.xml : Fichier de définition des noms, seuils minimums et maximums des couleurs.
  - detect.py : Fonction de détection du contour des couleurs et la direction de la flèche.
  - gamegroup.py : Classe de l'interface du jeu (état courant et déplacement des objets).
  - gameicon.py : Classe d'un objet du jeu (Emplacement et détection de collision).
  - importcolors.py : Fonction d'importation des couleurs définies dans "colors.xml".
  - mainwindow.py : Classe de la GUI principale de l'application (regroupement de l'interface du jeu, paramètres et la capture du caméra).

▪ paramsgroup.py : Classe des paramètres du jeu (couleurs, vitesse et niveaux de difficulté).

▪ placeobjects.py : Fonction de définition des niveaux de difficulté (placement des objets dans l'interface du jeu).

5.2 Installation de l'application

Pour exécuter l'application, à part l'installation du Python (version testée : 3.7), il faudra installer le package OpenCV : dans la ligne de commandes, entrez : **pip install opencv-python** (ceci installera automatiquement OpenCV (env. 50 Mo) et sa dépendance NumPy (env. 20 Mo)).

Après, il suffira simplement d'exécuter le fichier principal "__main__.py".

5.3 Manuel d'utilisation

L'application développée est intuitive, simple à utiliser. Elle est composée de trois sections :

● Caméra : Affichage de l'image capturée par la caméra avec les informations associées (état de détection, encadrement des couleurs détectées, l'angle réel et la direction de la flèche).

● Jeu : La section principale, permettant de visualiser le déplacement de la souris, les collisions et l'étant courant (fromages restants, partie "gagnée" ou "perdue").

● Paramètres : Les améliorations apportées (choix des couleurs, vitesse et niveaux de difficulté).

L'utilisateur (le joueur) n'a qu'à choisir le niveau de difficulté et commencer la partie. Les autres paramètres peuvent être ajustés à tout moment.

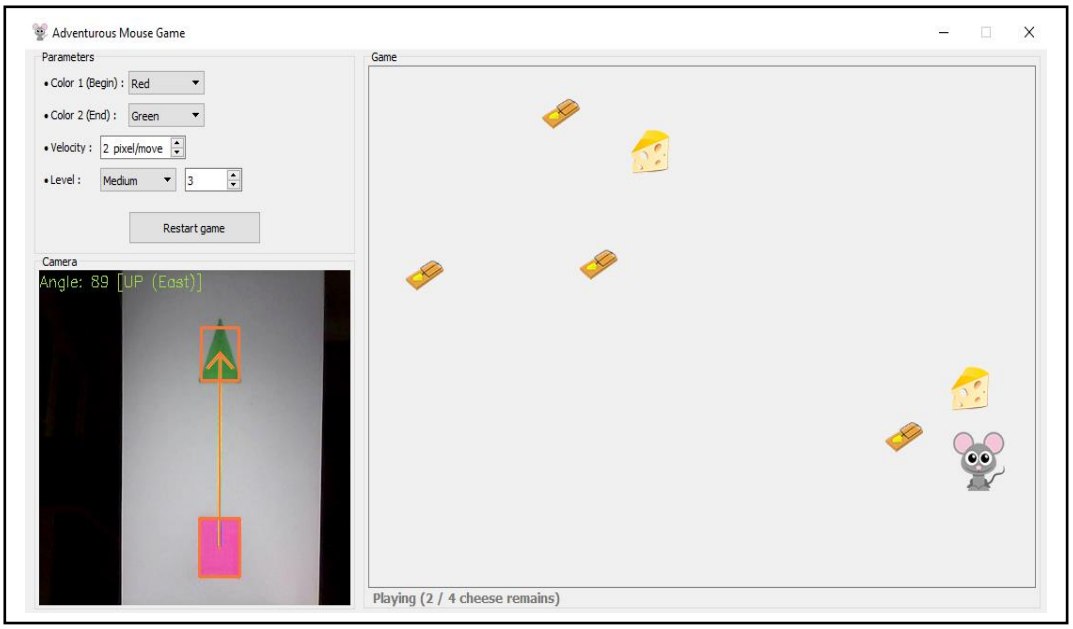

Figure qui montre une "partie en cours" du jeu développé

État courant : Partie "gagnée"

État courant : Partie "perdue"

5.4 Questions / Réponses

- Question : Quelles fonctions d'OpenCV avez-vous utilisé ?
- Réponse : Uniquement les fonctions vues en TP ont été utilisées. La liste ci-dessous :
  - Capture d'une image à partir de la caméra : VideoCapture.
  - Conversion entre les systèmes des couleurs : cvtColor, COLOR_BGR2HSV, COLOR_HSV2RGB.
  - Ecriture / dessin sur l'image : arrowedLine, circle, putText, rectangle.
  - Autres : resize, waitKey.

- Question : Pourquoi vous avez utilisé le langage Python au lieu du C++ ?
- Réponse : Nous n'avons jamais programmé en C++, que les notions de base nous sont connues. Or, nous avons déjà programmé en Python, donc ça sera plus simple et rapide d'implémenter l'application. De plus, le but du TP est de résoudre la problématique posée (idées algorithmiques), peu importe l'environnement utilisé.

- Question : Pourquoi l'exécutable direct de votre application n'est pas fourni avec le code-source ?
- Réponse : Pour générer un exécutable complètement fonctionnel (sans erreurs) sous Windows, la conversion du code Python en C++ est nécessaire, cela est fait avec "Cython". Ce dernier utilise les outils de développement Visual Studio qui dépend de la version du Python installée, pour notre cas, c'est la version 2017. Le problème est que cette dernière est très volumineuse (presque 15 Go), ainsi, cela ne vaut pas le cout de la télécharger juste pour compiler un petit bout de code. Notons que d'autres outils ont été essayés (cx_Freeze et py2exe), mais malheureusement sans soucis.