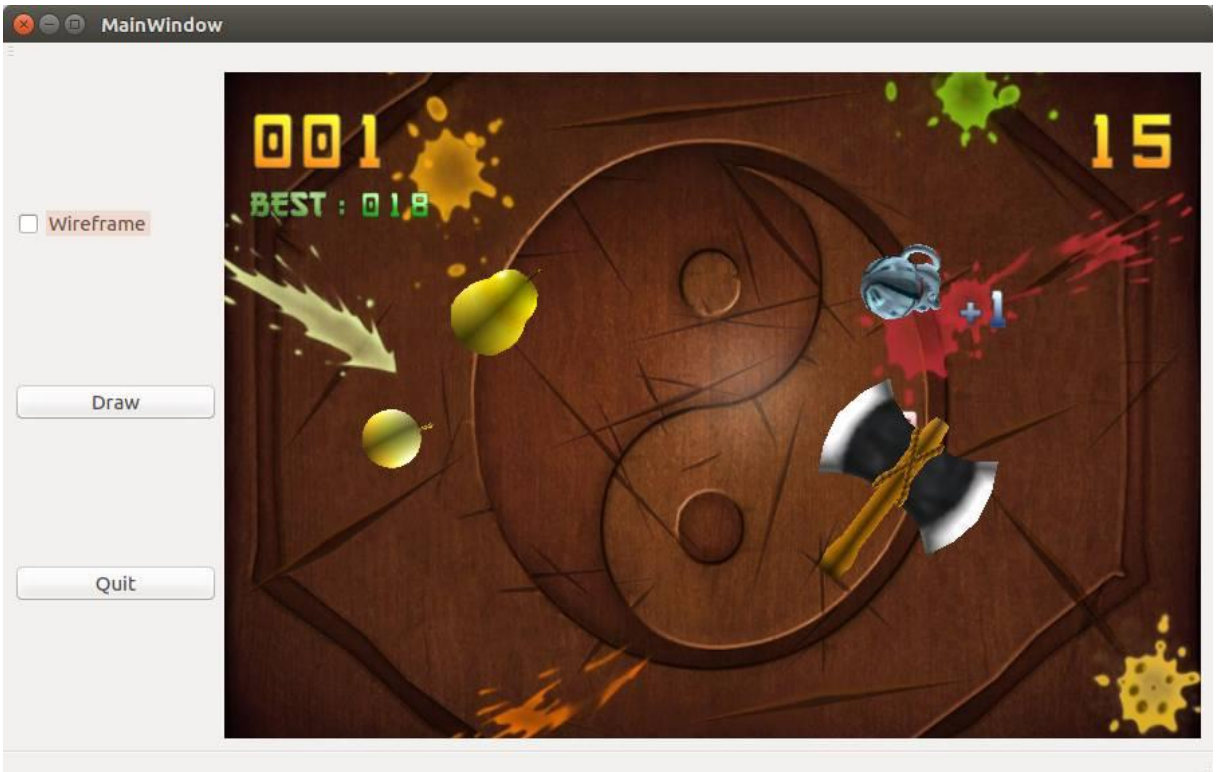
4ETI – Majeur IMI

FARINEAU Camille
MAIGRE Lucas
NGATTAI LAM Prince

Rapport de projet majeure IMI

Jeu "Fruit Samouraï"



Introduction

Ce projet nous permet de mettre en application toutes les connaissances acquises durant cette année en traitement et synthèse d'images. Il s'agit d'un sujet ouvert nous permettant de nous familiariser avec la bibliothèque OpenCV (Open Source Computer Vision), très répandue dans le milieu de l'imagerie, et spécialisée dans les applications de vision par ordinateur, détection de personnes, de formes, tracking. Notre pouvions faire un projet d'analyse d'image et de flux vidéo, un projet de réalité virtuelle ou encore un projet de réalité augmentée, tout en étant assez libre sur le contenu exact du projet.

Premièrement, nous pensions travailler sur un programme de réalité augmentée en incluant une scène 3D dans une scène réelle. Au cours d'une séance de brainstorming et de réflexion sur notre sujet nous nous sommes finalement fixés sur un projet contenant une scène 3D avec laquelle l'utilisateur interagit grâce à la caméra. Nous avons conçu un jeu développé en C++ et qui utilise les librairies OpenCV et OpenGL.

La raison principale qui nous a poussés à choisir ce sujet est justement qu'il nous permettait de prendre en main cette bibliothèque que nous n'avions jamais utilisée auparavant. De plus, étant un sujet plus ouvert que les autres, nous avions la liberté d'exprimer notre créativité.

Nous allons dans la suite tout d'abord présenter notre projet, puis nous expliquerons la méthodologie de travail adoptée durant tout le projet, puis nous rentrerons ensuite dans les détails plus techniques.

I- Présentation

Pour ce projet de majeure nous avons voulu créer un jeu qui puisse interagir avec le joueur. Pour cela nous nous sommes inspirés d'un jeu mobile très connu qui existe depuis plusieurs années et auquel nous avons déjà joué : « fruit ninja ». Le but ici n'était pas l'innovation, nous nous sommes alors largement inspiré de ce jeu et nous avons donc créé un dérivé de ce jeu que nous avons intitulé : « Fruit Samouraï ».

Nous avons choisi ce jeu car ces règles sont très simples et qu'il nous paraissait réalisable à notre niveau avec les connaissances que nous avons acquises depuis ces dernières années à l'école.

Le but de « Fruit Samouraï » est de découper le plus de fruits qui apparaissent aléatoirement sur l'écran. Les fruits sont lancés depuis le bas de l'écran, traversent ce dernier puis en sortent. A l'aide d'une hache nous découpons les fruits qui apparaissent et chacun d'eux rapportent 1 point.

Sont mélangés aux fruits des bombes qui sont plus rares mais qui font office de malus dans le jeu, il faut donc éviter de les toucher en les esquivant avec la hache. En fonction du mode de jeu ils n'ont pas le même fonctionnement :

- Dans le mode de jeu « vie », le joueur débute avec trois vies. A chaque fois que le joueur touche une bombe elle lui en retire une. Le jeu se termine quand les trois vies ont été utilisées. Au fur et à mesure que le temps passe la vitesse des lancers des fruits et des bombes est accélérée ce qui rend plus difficile pour le joueur d'esquiver une bombe.
- Dans le mode de jeu « temps », le joueur n'a que 60 secondes pour jouer. A chaque fois que le joueur touche une bombe elle lui enlève 10 points du score courant. Le joueur ne peut pas obtenir de score négatif, il bloque automatiquement à 0 dans ce cas.

Le but de ce jeu est donc d'obtenir le meilleur score possible dans les deux modes de jeux présentés. Nous avons pensé à différentes méthodes pour que le joueur puisse interagir avec le jeu et nous nous sommes focalisé sur l'utilisation d'une manette comme hache.

Figure 1 : Screenshot du menu du jeu

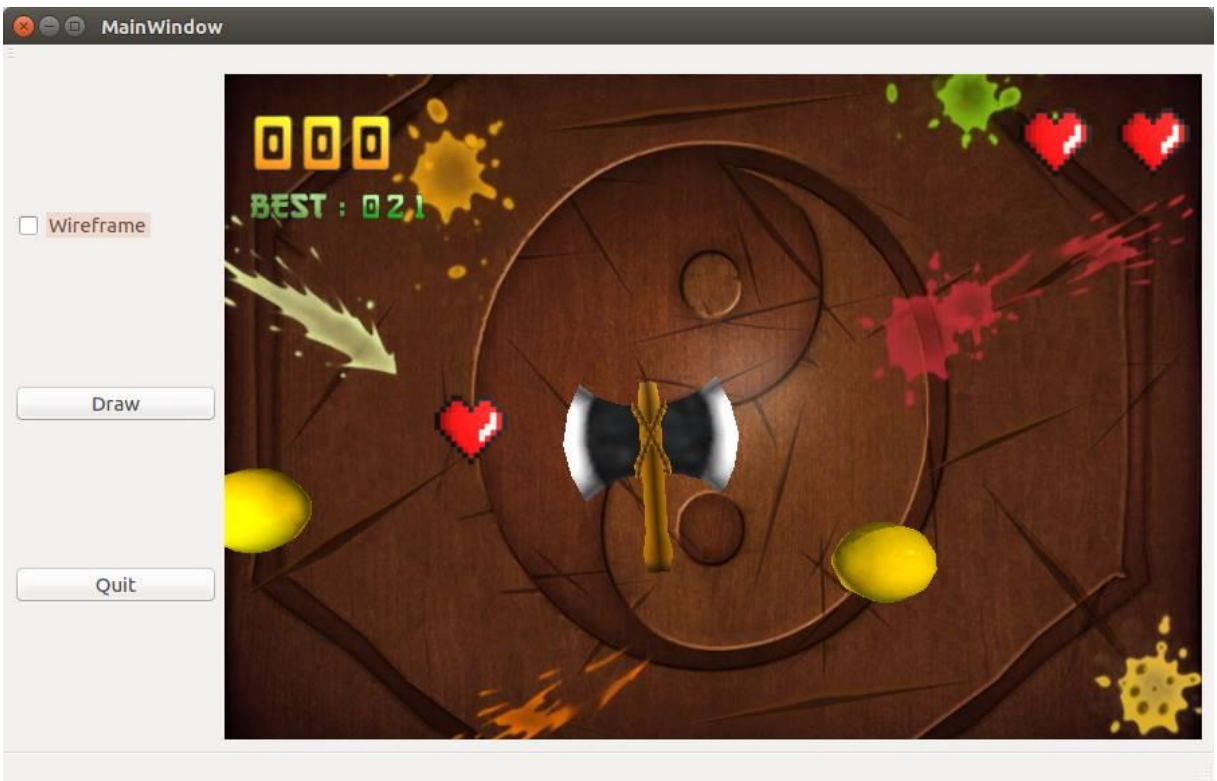


Figure 2 : Screenshot d'une partie avec des vies

Méthodologie

Afin de développer correctement ce jeu et de gérer correctement le projet, nous avons mis en place une méthodologie composée de plusieurs outils : planning, cahier des charges ainsi qu'un tableau de répartition des tâches.

Tout d'abord, après avoir décidé quel serait notre jeu, nous avons conçu un cahier des charges du projet. Nous avons décomposé notre jeu en modules et sous-modules.

Chaque module était alors catégorisé en tant que module obligatoire ou module optionnel afin de différencier les éléments indispensables au bon fonctionnement du jeu et les parties non nécessaire à la version de base du jeu mais qui viendraient l'améliorer. La Figure 1 synthétise ce cahier des charges.

La mise en place de ce cahier des charges nous a permis d'organiser notre travail. Nous nous sommes tout d'abord concentrés sur les modules obligatoires afin d'avoir une version livrable et fonctionnelle du jeu au plus vite. Ensuite nous avons développé les modules optionnels qui sont venus améliorer le jeu et l'expérience utilisateur.

Afin de tenir les délais imposé par le projet et d'avoir à la fin de celui-ci un jeu fonctionnel et agréable pour l'utilisateur, nous avons conçu au début de notre projet un planning détaillant les tâches à réalisées suivant les séances. L'annexe 1 présente ce planning.

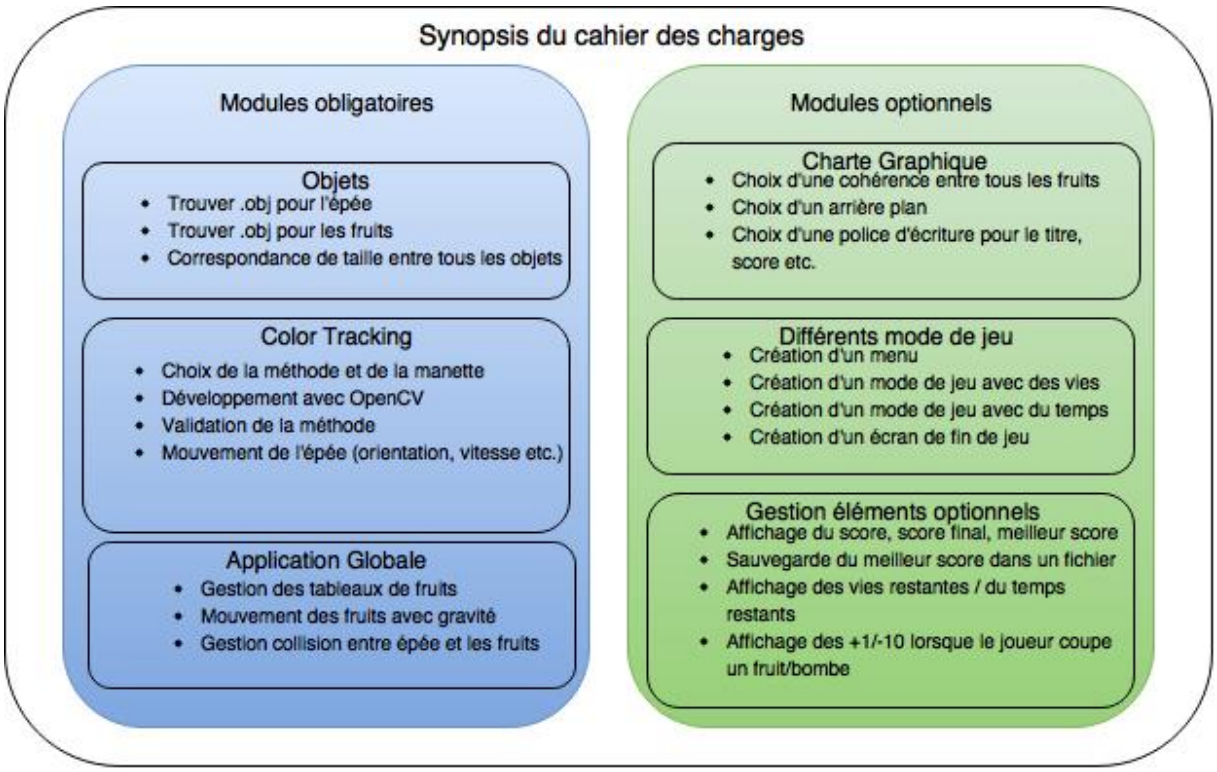


Figure 3 : Synthèse du cahier des charges

Nous nous sommes ainsi répartis les tâches au fur et à mesure de l'avancement du projet. Chaque fois que des modules étaient finis, nous avons effectué des tests unitaires sur celui-ci puis nous l'intégrions dans le projet global. Enfin nous testions le jeu avec le nouveau module.

Ceci nous a permis de procéder par étape et de gérer notre temps correctement. Après chaque intégration de nouveau module nous pouvions passer au suivant d'après le cahier des charges.

Avoir mis ces outils en place nous a permis de gérer le projet correctement à la fois en temps et en organisation. Chacun savait ce qu'il avait à faire et à quel moment. Nous faisions des points régulièrement afin de voir l'avancé de chacun.

À terme grâce à cette méthodologie nous avons réussi à concevoir un jeu fonctionnel, complet et agréable pour l'utilisateur.

II- Mise en œuvre du fonctionnement global du jeu

Nous allons maintenant expliquer quelques points techniques dans la mise en œuvre du fonctionnement global du jeu. Ces aspects techniques ne concernent que ce qui apparait à l'écran et non l'interaction avec le joueur et sa manette qui sera détaillée dans la partie suivante.

- **Les fruits et la bombe**

Pour faciliter la manipulation des fruits et leur interaction avec le jeu nous avons décidé de créer une classe « Fruit » comportant toutes les informations nécessaires : le type de fruit (nous avons 10 types différents de fruits plus une bombe), sa position dans l'espace, sa vitesse, son orientation et sa visibilité. De plus deux fruits sont utilisés pour l'écran d'accueil et qui représentent le choix d'un des deux modes de jeux, on a donc une variable pour savoir si l'une d'entre elle a été sélectionnée.

Pendant la partie plusieurs fruits se baladent sur l'écran, nous avons donc créé un tableau de 100 fruits qui s'affichent ou non dans le jeu. Ce nombre de fruits pouvant s'afficher sur l'écran en même temps est limité à 100 afin de ne pas surcharger le jeu. Cependant nous n'arriverons jamais à 100 fruits en même temps. Ce tableau est parcouru à tout instant du programme afin de voir quels fruits est à afficher ou non.

- **Initialisation du lancer des fruits**

Avant de lancer un fruit nous initialisons tout d'abord les différentes informations qui le compose et qui sont plus ou moins aléatoires : une position hors de l'écran, une vitesse qui est orientée vers l'écran, une visibilité (qui est enlevée lorsque le fruit rencontre avec la hache) et un axe de rotation qui sera le même durant toute la trajectoire du fruit.

Ces informations sont complétées dans le tableau dans la fonction « init_fruit » qui est appelée à intervalle de temps variable (entre une et quelques secondes). A chaque appel de cette fonction on initialise le lancer de un ou plusieurs fruits en même temps grâce à une autre variable aléatoire. Lorsqu'on sait le nombre de fruits qui vont être lancers, on remplit au même nombre de cases du tableau, les informations correspondantes de chaque fruit.

Au prochain appel de cette fonction on initialisera les cases suivantes du tableau de fruits. Ainsi, on remplit le tableau case par case avec différents fruits ayant chacun leur caractéristique au moment de leur lancer. Lorsqu'on atteint la fin du tableau (case 100), on retourne à la première case.

- **Parcours et trajectoire des fruits**

Pour simuler les lancers de fruits et pour qu'ils paraissent normaux (comme si quelqu'un les lançait en dehors de l'écran vers l'écran) nous avons discrétisé leur trajectoire en y ajoutant les contraintes de gravité et de frottement avec l'air (damping).

Au moment du lancer on initialise dans le tableau les informations de fruits qui ne sont pas encore éjectés et une fois qu'un fruit est lancé, sa position est périodiquement mise à jour, dans la fonction « update_pos », tout comme son orientation dans l'espace juste avant qu'on affiche celui-ci. Pour cela on applique la gravité au fruit et les frottements de l'air pour modifier la vitesse du fruit et ainsi nous pouvons obtenir la position future du fruit :

```
fruit.speed  = fruit.speed+fruit.dt*gravity;        //on applique la gravité
fruit.speed -= fruit.dt*damping*fruit.speed;        //on applique les frottements
fruit.pos = fruit.pos +fruit.dt*fruit.speed;        //on peut en déduire la position future du fruit
```

- **Affichage des fruits**

L'affichage des fruits s'effectue dans la fonction « draw_fruit », on parcourt le tableau et si la visibilité du fruit est activée (il n'a pas encore eu de collision avec la hache), alors on l'affiche.

- **La collision des fruits avec les bords de l'écran et la hache**

Lorsqu'un fruit heurte avec un des bords de l'écran (sauf celui du bas), le fruit rebondi pour changer de direction.

Pour ce faire au moment de la mise à jour de la position du fruit on vérifie si celui-ci ne sort pas de l'écran, si c'est le cas on le replace sur le bord de l'écran et on lui applique une nouvelle vitesse qui est opposée à la précédente. De plus on ajoute un coefficient de rebond afin que le fruit perde de la vitesse lors d'une collision.

Pour la collision avec la hache, qui permet ensuite une interaction (soit un bonus pour le fruit, soit un malus pour la bombe), nous vérifions en permanence la position de la hache en fonction de chaque fruit visible. Si un des fruits visibles, lors du parcours du tableau, se trouve dans la zone de découpe de la hache alors il y a collision et nous désactivons sa visibilité. Pour ne pas faciliter le jeu nous avons enlevé la possibilité de découper un fruit avec une vitesse proche de nulle. Pour cela nous stockons dans une liste les dernières positions de la hache, nous en extrayons une vitesse approximative et non vérifions qu'elle est supérieure à un seuil, que nous fixé, lors d'une collision.

- **L'orientation de la hache**

Pour plus de réalisme, et pour que notre utilisateur soit plus immergé dans le jeu, nous avons également implémenté l'orientation de notre hache. Ainsi, si l'utilisateur tourne la manette, la hache tourne aussi à l'écran. Pour réaliser cela nous avons tout d'abord besoin de deux points, d'où la nécessité de deux boules sur notre manette. La détection de la première balle nous donnera la position de la hache, et la position de la seconde balle quant à elle nous permettra par des formules trigonométriques de déterminer l'angle de rotation à appliquer. Si l'on appelle α l'angle à appliquer, et $P_1(x_1; y_1)$ et $P_2(x_2; y_2)$ les deux points détectés. On a :

$$\alpha = \mathrm{Arctan}(\frac{x_2 - x_1}{y_2 - y_1})$$

On gère à part le cas où $y_2 = y_1$ et nous effectuons la rotation nécessaire dans ce cas.

- **Eléments graphiques à l'aide d'OpenGL**

Pour les différents éléments représentés en 3D tels que les fruits ou les bombes, nous avons téléchargés des « models » sur internet que nous avons ensuite implanté dans notre jeu à l'aide d'OpenGL.

Pour tous les autres éléments graphiques tel que les textes ou les affichages des scores nous avons réalisés des images de ces éléments sur *Microsoft Word* et nous les avons appliqués en tant que texture sur des « mesh » planes rectangulaires.

Pour l'affichage et la gestion du meilleur score nous l'enregistrons en fin de partie, si le précèdent meilleur score a été battu. Il est ensuite lu, sur ce même fichier, en début de partie suivante pour que l'on puisse le stocker dans une variable et que cette dernière puisse être utilisée dans le jeu.

III- Détection de la manette

La détection de la manette fut la principale problématique de ce projet. En effet, pour que l'utilisateur puisse interagir avec notre jeu, nous avons décidé de concevoir une manette formée d'une tige en bois ayant deux balles colorées, de couleurs différentes (une rouge et une verte, ou une jaune et une bleue) aux extrémités. Tout le problème étant de détecter cette manette de façon assez robuste pour ne pas être trop influencé par l'environnement, tout en gardant une certaine réactivité et une certaine fluidité dont notre jeu a besoin. Nous avons abordé différentes approche pour que la détection réponde à ces critères.

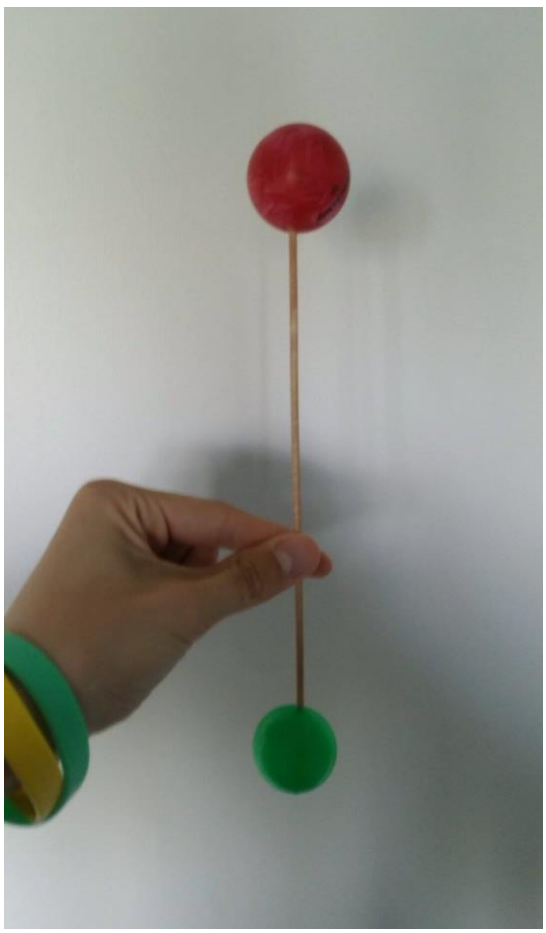


Figure 4 : Manette utilisée lors du projet

1- 1ere Approche - Détection par couleur unique

Il s'agit de la détection par couleur unique qui a été implémenté dans le code initial qui nous a été confié. Cette méthode consiste à parcourir tous les pixels de l'image filmée par la webcam, tout en récupérant leurs composantes RGB. On vérifie ensuite si la couleur du pixel vérifie le critère de sélection correspondant à la couleur que l'on veut détecter, et si c'est le cas on retient sa position dans un tableau. On fait ainsi pour toute l'image et en faisant une moyenne du tableau on obtient la position du milieu de l'objet de la couleur à détecter.

Nous avons testé cette méthode avec la couleur rouge, puis avec la couleur bleu. Les résultats étaient plutôt positifs, nous avions une bonne reconnaissance de la couleur et un bon positionnement du centre. Néanmoins en essayant d'appliquer cette méthode à d'autres couleurs qui ne sont ni le rouge, ni le vert, ni le bleu, par exemple le jaune, nous avons eu de grandes difficultés dans la mise en place du critère de sélection. Soit notre critère était trop précis et donc notre objet n'était pas détecté, soit il ne l'était pas assez et tout aux alentours (tel que les chaises, les tables, les personnes) était détecté.

Nous avons compris que l'un des défauts principal de cette méthode était le fait qu'elle utilise un critère fixe pour détecter la couleur, or en fonction de la luminosité, de l'environnement, de la texture, la couleur changeait plus ou moins. Notre méthode n'aurait donc jamais été réellement robuste et applicable en temps réel. Nous avons donc décidé de la rendre plus robuste en y ajoutant la détection de cercle.

2- 2e Approche : Détection de cercle

Notre manette est composé de deux balles, ce qui en 2D nous donne deux cercles. L'idée était de superposer la méthode précédente avec une détection de cercle, de sorte à ce que nous détection non seulement une couleur donnée mais dans une forme donnée. Ainsi nous détecterions uniquement des cercles d'une couleur donnée correspondant à notre manette. Dans le principe cela permettrait de rendre plus robuste notre méthode.

Pour effectuer cette détection de cercle, nous profitons du fait d'utiliser les bibliothèques OpenCV, et nous nous servons de la fonction HoughCircles()[1]. Cette fonction prend en paramètre une image, un tableau de vecteurs de dimensions 3, ainsi que des paramètres dont un correspond à la détection des contours et l'autre est un critère de précision (ou nombre de votes). Cette fonction analyse l'image envoyée en paramètre conformément à la méthode de Hough, et remplit le tableau avec les coordonnées (x,y) des centres des cercles détectés, ainsi que leurs rayons.

Nous avons testé cette méthode toute seule, sans la détection des couleurs, et elle ne fonctionnait pas comme nous le voulions. Pour un verre ou un gobelet par exemple, la méthode détectait bien un cercle, mais pour notre balle ce n'était pas aussi bien. Nous avions le classique problème avec le critère de sélection, soit il était trop faible et nous détections tout et n'importe quoi soit il était trop grand et nous ne détection pas notre cercle. Le problème est qu'en utilisant cette fonction d'OpenCv, nous utilisons une boite noire c'est-à-dire que nous n'avons pas accès au code qui se cache derrière et donc il est très difficile de résoudre les problèmes, puisque tout ce que nous tentions (à savoir modifié les paramètres, effectuer un prétraitement de l'image à analyser) était à l'aveugle.

Nous avons par exemple tenté de supprimer le fond et de ne garder à l'image que les objets en mouvement, à savoir donc la main et la manette, croyant que le problème venait de

[1] Documentation OpenCV :
http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_circle/hough_circle.html

l'environnement qui perturbait trop notre détection. Pour ce faire nous utilisons la méthode « *Background Substractor* » [2]Et nous essayions d'appliquer la détection de cercle sur cette image sans fond. Le résultat ne fut pas glorieux, il fut même pire.

Finalement, nous sommes parvenus à trouver sur internet une sorte de réplique du code de cette fonction. Nous avons ainsi pu afficher l'image sortant du filtre de Sobel que la fonction utilise pour détecter les contours.

Nous nous sommes rendu compte que la luminosité influençait grandement nos résultats. Et également, lors de la détection des contours, un verre ou un gobelet donne un cercle plutôt épais ce qui augmente les chances d'être détecté. Par contre notre balle nous donne un cercle très fin avec des imperfections. De plus en fonction de la texture de la balle et de la luminosité, le contour pourra être déformé, on se retrouvera donc avec des discontinuités dans le contour, ou alors des contours qui ne sont pas vraiment circulaires. Ces imperfections sont incompatibles à la mise en place d'une méthode robuste et fluide. L'idéal aurait été de coder nous-même notre propre fonction de détection de cercle, nous aurions pu ainsi en contrôler les paramètres et agir en conséquence. Néanmoins, nous avions déjà perdu assez de temps avec cette méthode qui au final ne nous a rien donné, nous avons donc décidé de nous y prendre autrement.

3- 3e Approche : Détection par color tracking

La troisième approche est similaire à la première dans le sens où nous revenons à une détection par couleur. Nous avons vu de nombreux exemples de color tracking sur internet, particulièrement un dont nous nous sommes inspirés[3]. Nous avons dû faire pas mal de modification dans ce code pour l'adapter au projet, mais nous avons gardé le même principe de fonctionnement. Le principe change quelque peu de la première méthode. En effet, cette fois ci on choisit directement sur la fenêtre la couleur que l'on souhaite détecter. Le principe est le suivant : Tout d'abord nous récupérons l'image RGB filmée par la webcam et nous la convertissons en image HSV. Cette étape est importante, car en base HSV, les composantes d'un pixel sont sa teinte (H-Hue), sa saturation (S-Saturation) et sa luminosité (V-Value). En cliquant sur un pixel de l'image nous recueillons donc ses composantes HSV, qui deviendront les composantes de références à détecter. Ensuite nous procédons comme la méthode une en parcourant notre image et en moyennant les positions des pixels vérifiant le critère. Le critère de sélection en question ne se fera que sur la composante H et sur la composante S, de sorte à ce que l'on ait une grande indépendance vis-à-vis de la luminosité, qui peut perturber la détection de couleur. Pour définir ce critère nous devons bien sûr définir un seuil de tolérance qui doit être adapté à l'environnement et à la situation.

Cette méthode à l'avantage d'être plus robuste que la première méthode. En effet, on s'affranchit des paramètres fixes implémentés dans le code en faveur d'un choix des couleurs en temps réel. Nous sommes donc moins limités et nous avons un plus vaste choix de couleur utilisable. De plus, le passage en HSV, nous permet de réduire l'influence de la luminosité qui nous posait quelques problèmes. Enfin, cette phase de calibrage nous permet de savoir en temps réel ce qui est détecté, et l'on peut corriger la détection avant de commencer le jeu. Néanmoins, elle présente quelques soucis elle aussi pour détecter le jaune en raison de sa trop grande présence dans l'environnement. Et bien que l'on ait rendu cette méthode plus robuste, nous avons tout de même des contraintes car il s'agit toujours d'une détection par couleur, il suffit que l'environnement ne soit pas adapté et notre jeu est inutilisable.

[2] Background Substractor Method :
http://docs.opencv.org/master/d1/dc5/tutorial_background_subtraction.html#gsc.tab=0
[3] Detection de Couleur : http://www.geckogeek.fr/tutorial-opencv-isoler-et-traquer-une-couleur.html

En somme nous sommes arrivés à un résultat très satisfaisant en terme de réactivité et de fluidité, et moyennement satisfaisant en terme de robustesse car pouvons faire mieux, par exemple en implémentant une détection de cercle fonctionnel et en superposant à notre méthode de color tracking.

IV- Lancement du jeu

Notre jeu commencera par une phase de calibrage de la manette.

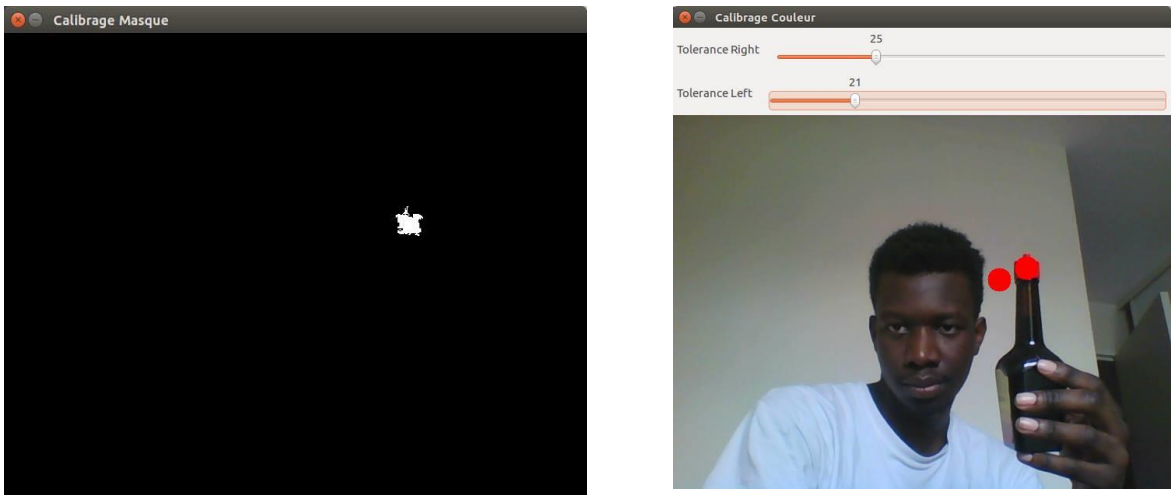


Figure 5: Phase de calibrage

Voici à quoi ressemblera le jeu au démarrage. Dans l'image à droite on clique sur la couleur à détecter, ici on clique sur le bouchon rouge de cette bouteille. On peut voir sur l'image de gauche une tache blanche, correspondant aux pixels qui sont détectés en fonction de la couleur sélectionnée. Nous avons ensuite deux curseurs en haut de l'image de droite qui nous permette de modifier la tolérance de sélection. Tolerance Right pour le clic droit et Tolerance Left pour le clic gauche. En effet, nous avons besoin de détecter deux couleurs, nous implémentons donc deux couleurs à détecter. Le point rouge à droite, nous donne l'indication du point se situant à la moyenne des positions de chaque pixel blanc. Concrètement la position du point rouge sera la position utilisée durant tout le jeu pour notre hache. On modifie la tolérance jusqu'à ce que la détection soit satisfaisante, autrement dit que le point rouge à gauche ne soit pas trop instable. On ne doit pas hésiter à bouger l'objet dans tous les sens et voir que le point rouge reste bien collé à la couleur. Si c'est n'est pas le cas il faut cliquer à nouveau ou modifier la tolérance. Cette étape est primordiale pour une bonne fluidité du jeu. Une fois le calibrage effectué, il suffit ensuite de cliquer sur 'q' ou 'Q' pour lancer le jeu.

Nous sommes ensuite face au menu principal ci-contre où nous devons choisir notre mode de jeu en tranchant un des deux fruits correspondants.

Le jeu démarre ensuite, et vous pouvez vous acharnez sur les fruits sans aucune pitié en essayant de réaliser le meilleur score possible.

Figure 6 : Menu d'accueil

Conclusion & Améliorations

À la fin de ce projet, nous avons atteint notre objectif de concevoir un jeu fonctionnel et agréable pour le joueur, qui possède plusieurs fonctionnalités tels deux modes de jeu, sauvegarde de meilleur score etc. Grâce à notre méthodologie, nous avons réussi à finir la conception du jeu dans les temps et implémenter le maximum de fonctionnalités optionnelles. Le résultat est très satisfaisant, la méthode de détection par color tracking assez robuste, le jeu est fluide et intéressant pour le joueur.

Nous avons réfléchi à plusieurs améliorations potentielles de ce projet afin d'augmenter l'expérience utilisateur et l'intérêt du jeu.

La première idée est d'augmenter la robustesse de détection de notre manette. Pour ce faire, nous souhaiterions implémenter une méthode détection des cercles, grâce à la transformée de Hough, afin de superposer les méthodes de détection et d'accroitre notre précision sur la localisation de la manette par la caméra.

Une deuxième idée est d'essayer de s'affranchir de toutes les contraintes liées à l'environnement de jeu. Un changement de luminosité oblige l'utilisateur à calibrer à nouveau sa manette. Cependant dans un environnement donné, où les conditions (luminosité, éclairage etc.) ne changent pas, le calibrage n'est nécessaire qu'une seule fois.

Ensuite une amélioration possible est le fait de pouvoir jouer à deux joueurs. Actuellement avec la calibration de la manette sur deux boules de couleurs différentes (rouge et verte) nous pouvons jouer avec un seul joueur. En détectant une deuxième manette, composée de boules bleue et jaune par exemple, cela permettrait de faire joueur un deuxième joueur et ainsi d'augmenter le challenge du jeu, les deux s'efforçant de faire le meilleur score possible. Avec ce mode deux joueurs, on peut penser à des améliorations qui permettraient à un joueur de lancer des bombes sur l'autre ou d'autre chose de ce type.

Enfin la dernière idée concerne la façon de jouer en tant que telle. Nous aimerions mettre en place la détection de mains et des bras afin de jouer directement avec le corps et de s'affranchir des manettes. Ainsi le joueur n'aurait qu'à se présenter devant la caméra, les mains bien visibles, afin de commencer le jeu, comme avec le système Kinect de Microsoft par exemple.

Annexe

Annexe 1 - Planning du projet