DEPARTEMENT INFORMATIQUE
71 rue Peter Fink
01000 Bourg-en-Bresse
04 74 45 50 59

Diplôme Universitaire de Technologie

Informatique

Escalade en réalité augmentée

RAPPORT DE PROJET

MAADOUR Dalil, RICHAUD Guilhem, STUBLJAR Baptiste

Promotion 2020/2022

M. JAILLET

REMERCIEMENTS

Nous souhaitons tout d'abord remercier M. Jaillet, notre tuteur de projet, pour sa patience et le temps qu'il nous a accordé tout au long de ce projet.

Nous tenons également à témoigner notre reconnaissance à l'ensemble de l'équipe pédagogique de l'IUT. Ce projet n'aurait pas pu aboutir sans l'enseignement reçu durant ces deux années d'étude.

Nous remercions aussi M. Bachelet, professeur d'escalade, qui nous a permis d'effectuer nos tests en conditions réelles durant ses séances d'escalade.

Enfin nous remercions nos camarades de promotion pour leurs conseils et le temps accordé au test de notre projet.

TABLE DES MATIERES

I. INTRODUCTION

I.1 OBJECTIFS ET CONTEXTE DU PROJET

Dans le cadre de notre projet tuteuré réalisé au cours de notre deuxième année d'études à l'IUT Lyon 1 (site de Bourg-en-Bresse), nous avons dû réaliser un logiciel permettant de pratiquer de l'escalade en réalité augmentée. L'objectif était ainsi de proposer une expérience étendue de ce sport, au travers de jeux vidéo grandeur nature.

Certaines contraintes étaient à respecter : n'importe qui devait être en mesure d'utiliser le logiciel, ce qui impliquait l'utilisation de matériel simple : une caméra pour détecter la position des joueurs, un vidéoprojecteur pour diffuser les jeux sur le mur d'escalade, et un ordinateur pour les traitements, éventuellement remplacé par un Raspberry à terme. L'objectif était de proposer une interface simple permettant à l'administrateur de gérer les pistes et les grimpeurs, et de lancer différents jeux, facilement réalisables sur un mur d'escalade. Les jeux recommandés étaient :

- Le parcours : le but étant d'atteindre des points de passage dans l'ordre, et le plus vite possible. Dès qu'un objectif est touché, le joueur est prévenu par un signal visuel ou audio, puis sa performance est classée à la manière d'un jeu d'arcade.
- Pong : la balle est projetée sur le mur et deux joueurs doivent empêcher la balle de passer de leur côté à l'aide de leurs mains et leurs pieds.
- Astéroïdes : Un joueur se place sur le mur et doit esquiver les astéroïdes qui sont projetés sur le mur

Vous trouverez dans ce rapport, une présentation de l'équipe, puis les difficultés rencontrées durant tout le projet. Nous verrons ensuite les solutions apportées aux problèmes et enfin un aperçu du résultat.

I.2 PRESENTATION DE L'EQUIPE

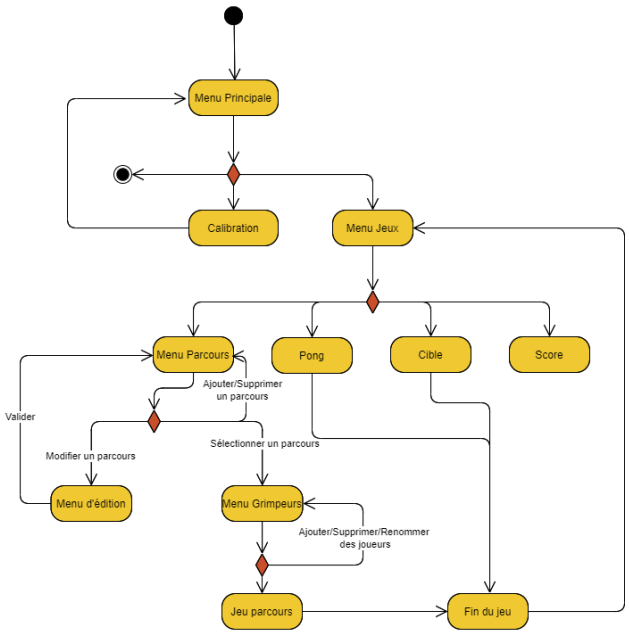
Pour la réalisation de ce projet, nous étions trois étudiants : MAADOUR Dalil, RICHAUD Guilhem et STUBLJAR Baptiste. Dalil s'est chargé de la partie jeux, et du début de l'interface. Guilhem a assuré le rôle de chef d'équipe, en déposant les rapports hebdomadaires et en concevant l'architecture du projet. Il a également été en charge de la partie modèle avec la base de données, de la détection de base des joueurs, et de la majorité de l'interface graphique sur la fin du projet. Enfin, Baptiste a mis en place les algorithmes de détection de la zone de jeux et des prises, ainsi que d'étalonnage.

II. REALISATION DU PROJET

II.1 PRESENTATION DES BESOINS

Plusieurs besoins ont été exprimés par notre tuteur en début de projet. Il fallait ainsi proposer les parties suivantes :

- Étalonnage, afin de corriger la perspective.
- Sélection de jeux, avec un catalogue de jeux au choix. Vous pouvez retrouver leurs règles dans la partie *I.1 Objectifs et contexte du projet.*
- Édition/configuration, permettant à l'administrateur de créer, d'éditer ou de supprimer des parcours composés de prises. La détection des prises d'un mur se fera automatiquement. Ces parcours seront ensuite sauvegardés pour être réutilisés.
- Gestion des joueurs, avec la possibilité de créer, d'éditer ou de supprimer des grimpeurs, ainsi que d'en sélectionner avant le lancement d'un jeu, afin de sauvegarder leurs scores (correspondant à la durée mise pour terminer un parcours) et réaliser des concours.
- Jeu, où le jeu sélectionné par administrateur sera lancé et projeté sur le mur, en capturant les mouvements du grimpeur.



Parmi les fonctionnalités qui n'ont pas pu être réalisées durant le projet, nous retrouvons :

- Le jeu "Astéroïde" : par manque de temps nous avons fait le choix de ne pas implémenter ce jeu, afin de nous concentrer sur les autres fonctionnalités.
- La sauvegarde de murs : Un mur aurait dû contenir toutes les prises présentes sur la surface de jeux ; et les parcours certaines prises présentes dans le mur auquel ils étaient liés. Cependant, nous avons fait le choix de n'implémenter que les parcours, plus simples pour l'utilisateur. En effet, cela aurait nécessité une couche supplémentaire de configuration, rendant l'utilisation du logiciel moins intuitive.
- La détection automatique des prises : l'algorithme de détection est fonctionnel mais n'a pas été implémenté dans l'interface car les jeux réalisés n'ont finalement pas nécessité cette détection.
- Nous n'avons pas essayé de porter notre projet sur Raspberry Pi, par contrainte de temps. Mais dans l'optique ou notre logiciel devait être utilisable par le plus grand nombre, il avait de tout manière plus vocation à être utilisé sur ordinateur portable.

II.2 SOLUTION APPORTEE

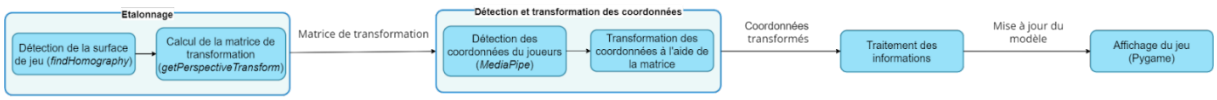
II.2.1 Solution technique mise en place

Durant la première partie du projet, nous avons pu déterminer la solution technique pouvant répondre aux besoins du projet. Il était nécessaire de séparer le projet en 4 grandes parties :

- L'étalonnage : comme il le sera expliqué dans la partie II.3.1 *L'étalonnage de la projection*, le logiciel a besoin d'une matrice de transformation pour pouvoir afficher correctement les grimpeurs.
- La détection des joueurs et la transformation des points récupérés.
- Le traitement de l'interaction entre le joueur et le jeu
- L'affichage du jeu et la gestion des parcours et des joueurs

Pour pouvoir organiser notre code, nous avons fait le choix d'utiliser le modèle MVC. Ce modèle était particulièrement adapté aux différentes parties du projet, en proposant une méthode structurée, nécessaire à des projets aussi importants que celui-ci.

Les 4 parties ont également été choisies pour pouvoir être développées séparément, nous permettant de commencer à programmer sans être contraint par l'avancement des autres. Cependant la partie jeux du projet allait être la plus longue étant donné les nombreuses possibilités de jeux à implémenter. Ainsi, deux d'entre nous ont commencé par l'étalonnage et la détection des joueurs pour ensuite rejoindre le plus rapidement possible le dernier développeur sur l'interface.



II.2.2 Outils techniques utilisés

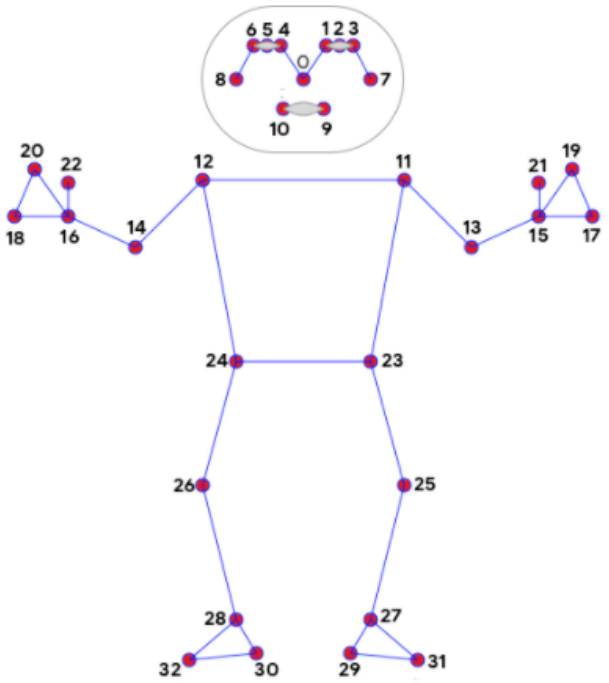
Aucune technologie ne nous a été imposée au cours de ce projet. C'est pourquoi nous avons recherché et sélectionné les langages et bibliothèques qui nous semblaient les plus adaptés.

II.2.2.1 OpenCV

OpenCV est une bibliothèque graphique libre, initialement développée par Intel, spécialisée dans le traitement d'images en temps réel. Elle nous permet ainsi de traiter le flux vidéo de la caméra, lors de l'étalonnage du jeu, mais aussi lors de la détection des prises sur le mur d'escalade.

II.2.2.2 Framework *MediaPipe*

MediaPipe est un framework développé par Google, basé sur des graphes pour la création de pipelines d'apprentissage automatique appliqués multimodaux (vidéo, audio et capteur). Entre autres, ils proposent de nombreuses solutions permettant la détection d'objets et personnes sur des images ou flux vidéo, ce qui répondait parfaitement à notre besoin de suivi des grimpeurs. En outre, MediaPipe propose plusieurs modes de détection en fonction des besoins, comme « Hands », pour se concentrer sur la détection des mains, ou encore « Iris », pour les yeux. Comme nous avions besoin de trouver la position des mains et des pieds du grimpeur, nous avons décidé d'utiliser le mode « Pose », permettant de détecter l'intégralité du corps.



0. nose
1. left_eye_inner
2. left_eye
3. left_eye_outer
4. right_eye_inner
5. right_eye
6. right_eye_outer
7. left_ear
8. right_ear
9. mouth_left
10. mouth_right
11. left_shoulder
12. right_shoulder
13. left_elbow
14. right_elbow
15. left_wrist
16. right_wrist
17. left_pinky
18. right_pinky
19. left_index
20. right_index
21. left_thumb
22. right_thumb
23. left_hip
24. right_hip
25. left_knee
26. right_knee
27. left_ankle
28. right_ankle
29. left_heel
30. right_heel
31. left_foot_index
32. right_foot_index

Figure 1 : Schéma des points détectés par le mode "Pose" de MediaPipe

MediaPipe utilise OpenCV pour traiter chaque image du flux de la caméra, et nous renvoyer un tableau de coordonnées en 3D. La profondeur renvoyée est déterminée de manière totalement artificielle, et est encore peu fiable, MediaPipe étant encore dans sa version béta, c'est pourquoi nous avons décidé de ne pas nous en servir et de considérer que le grimpeur sera toujours collé au mur.

Le framework est principalement fait pour être utilisé en Python ou C++, bien qu'il s'ouvre de plus en plus à d'autres langages. C'est cela qui nous orienté dans le choix du langage que nous allions utiliser pour le développement du reste du projet.

II.2.2.3 Python

Python proposant une utilisation plus aisée de MediaPipe, avec une documentation plus fournie sur le web, et des bibliothèques graphiques plus simples que SDL, nous avons finalement décidé de nous orienter sur ce langage. De plus, comme nous connaissions déjà le C++ nous avons considéré que le projet tuteuré serait l'opportunité d'apprendre un nouveau langage. Le projet a donc été entièrement développé en Python.

II.2.2.4 Pygame

Pygame est une bibliothèque libre multiplateforme qui facilite le développement de jeux vidéo temps réel en Python. Simple d'utilisation et basée sur SDL2, Pygame a répondu à nos besoins en matière d'interface. En outre, choisir une bibliothèque graphique utilisant le même langage de programmation que MediaPipe nous a permis de nous affranchir des contraintes liées à la communication entre deux langages différents, que nous aurions eu en utilisant des logiciels comme Unity, comme nous l'avions envisagé un temps, au début du projet.

II.2.2.5 Base de données

La base de données a pour objectif de stocker les parcours, les utilisateurs et l'historique des parties.

Plusieurs SGBD s'offraient à nous. Dans un premier temps, nous avons éliminé toutes les bases ne pouvant pas tourner en local. En effet, les utilisateurs ne disposant pas nécessairement d'une connexion dans le gymnase, cette solution n'aurait pas été adéquate. De plus, nous ne voulions pas nous occuper d'héberger la base, et de toute l'administration qu'il aurait alors été nécessaire de réaliser pour gérer les différents utilisateurs. C'est pourquoi l'utilisation d'une base sur la machine de ces dernier nous a paru être le choix le plus approprié. Nous pouvions alors soit utiliser une base MongoDB, soit une base SQLite. Nous avons dans un premier temps fait le choix de la base MongoDB, car le système de stockage sous format JSON nous parut plus intuitif pour, par exemple, stocker les prises présentes dans les parcours. Cependant, nous nous sommes assez vite rendu compte des limites de cette solution. En effet, MongoDB ne peut être utilisé en local seulement si l'utilisateur a déjà installé MongoDB sur sa propre machine. Or, nous ne voulions pas imposer cette étape supplémentaire à nos utilisateurs. Une solution aurait été d'exporter la base dans un fichier JSON à chaque fermeture du logiciel, et de la réimporter à chaque démarrage. Cependant, cette méthode aurait été très coûteuse en termes de temps et assez peu évidente à gérer pour les développeurs. C'est pour ces raisons que nous nous sommes finalement rabattus sur une solution plus traditionnelle, proposée par SQLite, permettant de stocker une base directement dans un fichier ".db", à l'intérieur du projet.

II.3 SOLUTION TECHNIQUE APPORTEE

Durant la réalisation du projet nous avons été confrontés à plusieurs problèmes techniques auxquels nous avons dû répondre, parmi lesquels l'étalonnage de la projection, l'affichage de l'interface, la gestion du modèle et de la base de données, et le fonctionnement des jeux vidéo, avec le suivi des joueurs.

II.3.1 L'étalonnage de la projection

Pour faciliter l'utilisation du logiciel, nous avons souhaité laisser à l'utilisateur la possibilité de placer comme il le souhaitait la caméra et le projecteur. En effet, il est déjà compliqué de pouvoir placer une caméra et un projecteur dans une salle d'escalade, il aurait été presque impossible que les

deux aient la même zone de projection/captation. Ainsi est apparue notre première contrainte : l'image perçue par la caméra n'allait pas être parfaitement alignée avec celle diffusée par le projecteur.

Pour faciliter la détection automatique, nous avons considéré que le projecteur était correctement réglé de sorte à projeter un vrai rectangle et non pas un trapèze. Cela nous a permis de ne pas faire de transformation entre l'affichage de l'interface sur l'ordinateur et avec le vidéoprojecteur.

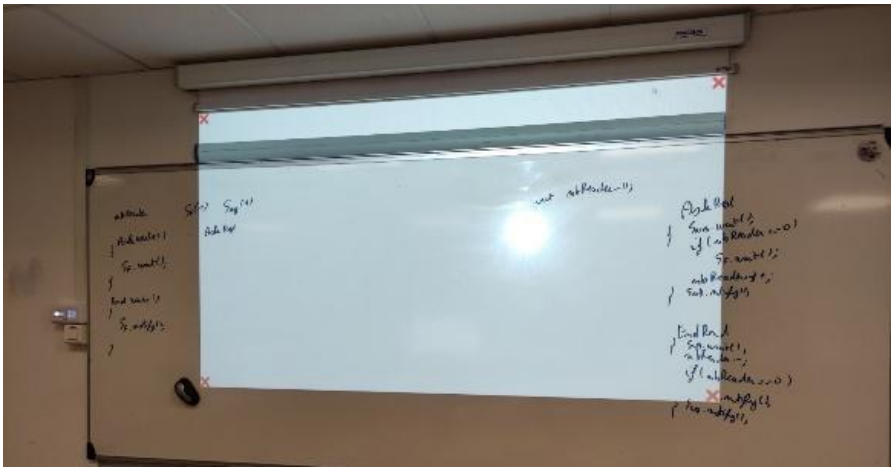


Figure 2 : Projection rectangulaire, mais trapézoïdale sur l'image

II.3.1.1 Création de la matrice de transformation

Lorsque MediaPipe détecte un grimpeur et nous renvoie ses coordonnées, ces dernières ne sont pas sur le même plan que le vidéoprojecteur. Il est donc nécessaire de faire une transformation sur les coordonnées reçues par MediaPipe, de telle sorte qu'elles correspondent à la projection. OpenCV permet de générer automatiquement une matrice de transformation applicable aux coordonnées. Pour cela nous avons utilisé la fonction *cv2.getPerspectiveTransform* qui prend en paramètre deux tableaux de 4 coordonnées, correspondant aux 4 coins de notre image et de notre projection, et renvoie la matrice de transformation correspondante.

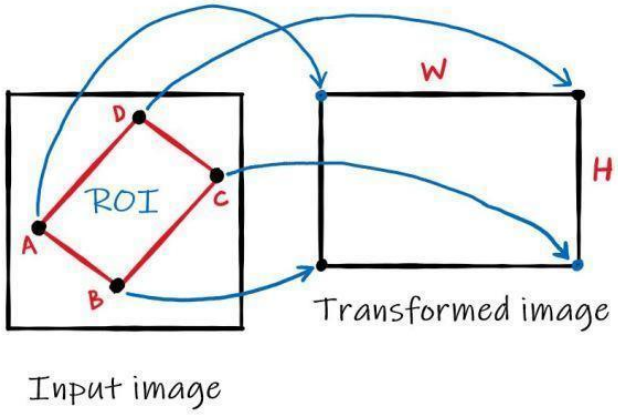


Figure 3

Sur l'image ci-dessus *(*Figure 3***Erreur ! Source du renvoi introuvable.)***,* on peut considérer les points A, B, C et D comme les 4 côtés de la projection et l'image d'entrée (input image) comme l'image transmise par la caméra.

Ainsi, lorsque MediaPipe nous fournit les coordonnées depuis le point de vue de la caméra, il nous suffit d'appliquer la matrice obtenue et pour obtenir les coordonnées des points sur le plan du mur d'escalade. Lorsque l'on projette ces points, ils s'affichent exactement là où se situe le grimpeur.

II.3.1.2 Détection des côtés de la projection

Comme nous l'avons vu dans la partie précédente, il est nécessaire de connaître les coordonnées des 4 coins de la projection. Afin de les détecter, la caméra va tout d'abord capturer une image puis va utiliser une méthode proposée par OpenCV appelé *cv2.findhomography*.

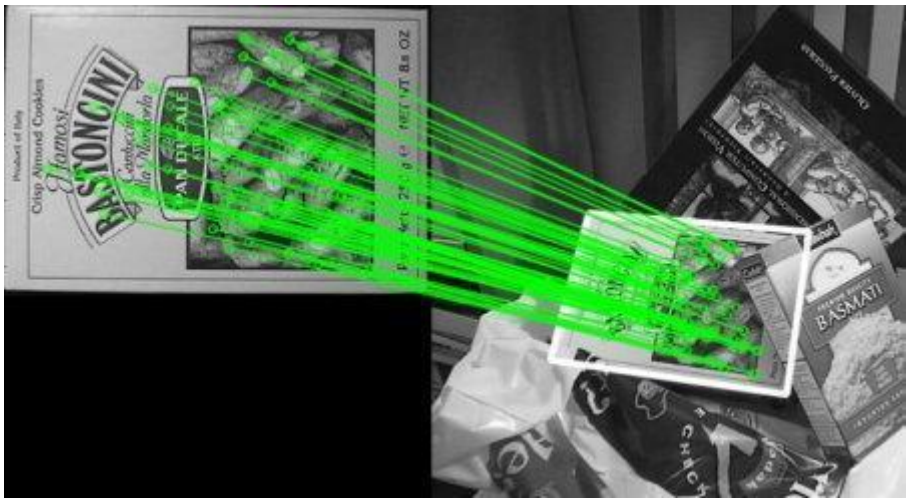


Figure 4 : Exemple de cv2.findhomography

Cette fonctionnalité nécessite une image « à plat » (image de gauche dans la Figure 4 : une boîte de cookies) de ce qu'OpenCV doit retrouver dans la seconde image (image de droite : un tas de boites). Elle va ensuite repérer des points clés sur les deux images et faire la correspondance entre les deux. Enfin, la fonction nous retourne les 4 coins de l'image de gauche dans l'image de droite. Il ne nous reste plus qu'à les trier dans l'ordre choisi et d'utiliser la fonction vue dans la partie II.3.1.1 Création de la matrice de transformation.

Pour ce projet nous avons choisi une image adaptée à la détection de points clés. Après quelques recherches sur internet, nous avons trouvé le "ChArUco board", qui est très utilisé dans la réalité augmentée, pour servir de base à la détection de surfaces. Le principal atout des ChArUco boards est leur rapidité à être détectés par OpenCV. De plus, comme la projection se fera sur un mur d'escalade souvent très coloré, cette image en noir et blanc sera plus facilement détectable.

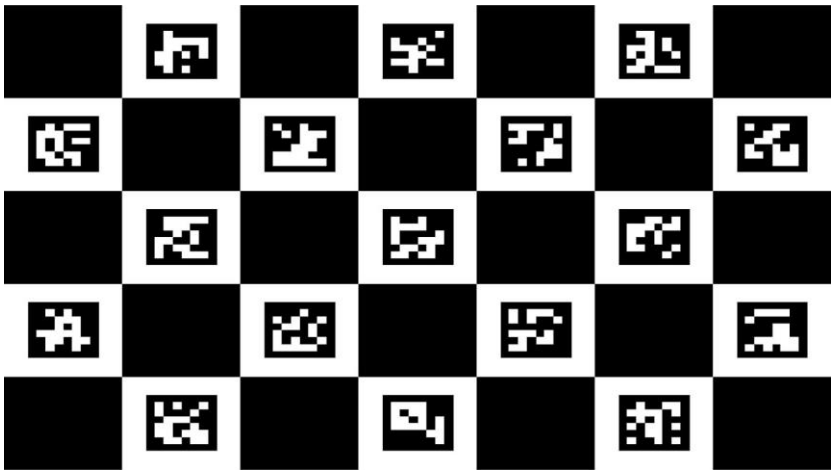


Figure 5 : Image ChArUco board

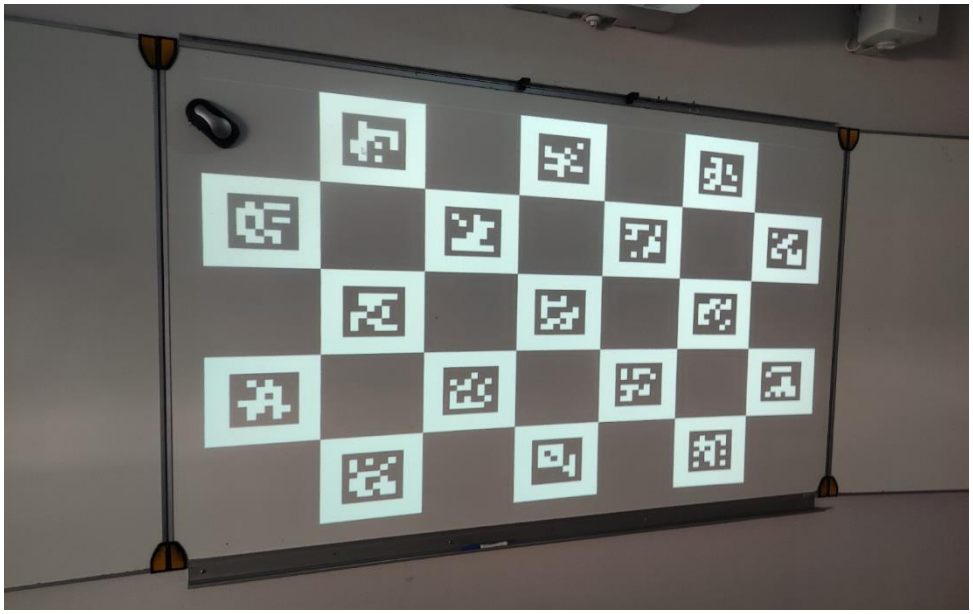


Figure 6 : Utilisation du ChArUco board durant l'étalonnage

II.3.2 L'interface

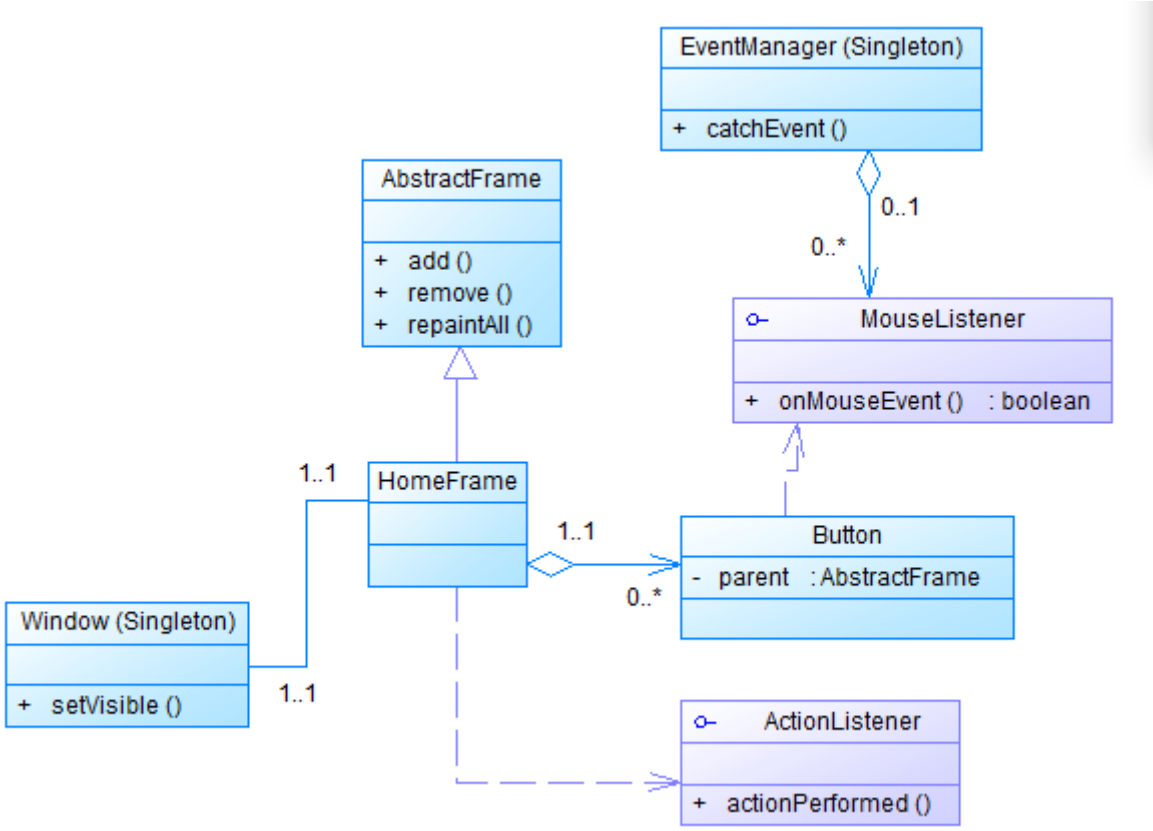


Figure 7 : Diagramme UML simplifié de l'interface

Bien évidemment, afin que les utilisateurs puissent utiliser notre logiciel, il fallait doter ce dernier d'une interface graphique. Comme nous l'avons déjà précisé, nous avons fait le choix de la bibliothèque Pygame. Pygame permet de réaliser des actions de base, comme la gestion d'événements, et le dessin de formes géométriques ou encore l'affichage d'images dans une fenêtre.

II.3.2.1 Cycle de vie

Le cycle de vie global de l'interface est donc le suivant : Dans le thread principal (classe Window, Singleton), on vient récupérer tous les évènements disponibles (classe EventManager, Singleton), puis on met à jour la fenêtre actuelle.

II.3.2.2 Gestion de l'affichage

Chaque fenêtre dispose d'une liste d'items, qui implémentent tous une méthode *draw()*, qui, lorsqu'elle est appelée, vient les peindre sur la fenêtre. Ainsi, lorsque la fenêtre est mise à jour, elle vide la vue, en affichant l'image de fond, puis appelle simplement la méthode draw() de ses items.

II.3.2.3 Gestion des évènements

Toute classe souhaitant gérer des événements doit s'inscrire auprès de l'EventManager, (d'où l'intérêt que ce soit un Singleton), grâce à un design pattern observer, afin qu'elle soit appelée lorsqu'un évènement approprié est reçu. En effet, les classes peuvent choisir de recevoir des évènements liés au clique, au mouvement ou à la molette de la souris, et au clavier, en implémentant respectivement les interfaces MouseListener, MotionListener, WheelListener et KeyboardListener.

Par exemple, la classe *Button* implémente *MouseListener*. Ainsi, lorsque les boutons reçoivent un évènement par le biais de l'*EventManager*, ils vérifient si la position de la souris correspond à leur propre position, puis retournent *True* ou *False* en fonction, de telle sorte que l'EventManager sache que l'évènement a été utilisé, et arrête son appel. Ensuite, le design pattern entre un seconde fois en jeu. En effet, des éléments, à l'instar des fenêtres peuvent venir écouter les boutons qu'elles contiennent, en implémentant l'interface *ActionListener*, de telle sorte à être prévenus lorsque le bouton est cliqué.

II.3.2.4 Changement de fenêtre

La classe *SwitchFrameController* s'occupe d'effectuer les changements de fenêtres. En lui passant en paramètre la classe de la fenêtre à instancier, et elle va se charger de vider l'*EventManager* de ses listeners, de créer la fenêtre, et de l'inscrire en tant que fenêtre courante, dans la classe *Window*.

Dans le cas des jeux, un appel supplémentaire à la fonction *execute()* de la classe StartGameController permettra de lancer le jeu en tâche de fond.

II.3.2.5 Conclusion

La plus grosse difficulté ici a été de devoir recréer tout un système fonctionnel, avec la gestion d'évènements et l'affichage de fenêtres complètes avec des boutons, listes et zones de texte. Peut-être la bibliothèque Pygame n'a-t-elle finalement pas été la plus appropriée, car proposant des fonctionnalités encore trop bas niveau, comparé à ce que Swing en Java peut proposer. Après des recherches post-projet, PySide ou PyQt4 auraient pu être plus appropriées.

II.3.3 La gestion du modèle et de la base de données

Afin de gérer la persistance des données à chaque lancement de l'application, nous avons utilisé la bibliothèque sqlite3.

II.3.3.1 Connexion à la base de données

Sqlite3nous a permis de facilement communiquer avec la base de données, une fois mis à part quelques problèmes rencontrés. En effet, certains paramètres étaient à passer lors de l'initialisation afin de correspondre à notre utilisation, et nous avons parfois mis du temps avant de trouver que l'origine des problèmes venaient de ceux-ci. Voici à quoi ressemble finalement la phase de connexion à la base de données :

```
self.con = sqlite3.connect("database.db", isolation_level=None, check_same_thread=False)

self.con.execute("PRAGMA foreign_keys = ON")
```

Le paramètre *isolation_level=None* permet de préciser que l'on souhaite exécuter nos requêtes dès l'appelle de la fonction *execute()*. Sans ce paramètre, aucune de nos requêtes ne prenait effet.

Le paramètre *check_same_thread=False* permet d'exécuter des requêtes sur des threads différents du thread principal, ce qui est le cas lors de la sauvegarde des scores à la fin d'une partie. Sans ce paramètre, l'application levait une exception.

Il était nécessaire d'exécuter la requête *PRAGMA foreign_keys = ON* pour que les clés secondaires prennent effet. En effet, sans cette requête, les ON DELETE CASCADE n'étaient pas déclenchés, ce qui compromettait l'intégrité de notre base lors de la suppression d'éléments qui dépendaient d'autres éléments (par exemple, lors de la suppression d'un parcours, il est nécessaire que les prises qui lui sont liées soient également supprimées).

II.3.3.2 Communication avec la base de données

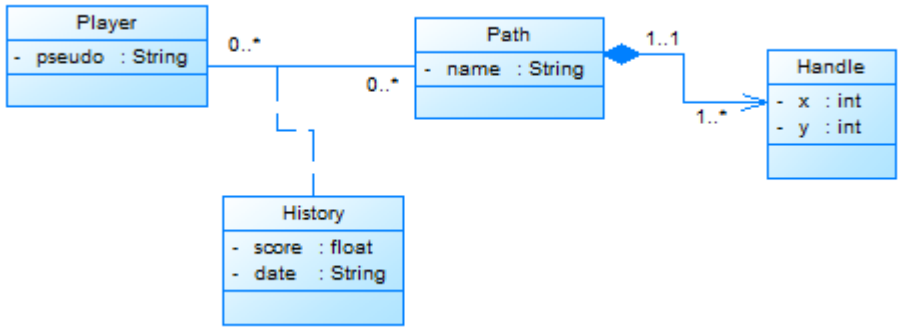


Figure 8 : Diagramme de classe de la base de données

Afin que le reste du logiciel puisse communiquer avec la base de données, nous avons créé une classe Database (Singleton), ayant des fonctions permettant d'enregistrer en base les éléments du modèle, prenant directement en charge les opérations CRUD (Create, Read, Update, Delete). Par exemple, la fonction *setHandlesInPath()* prend en paramètre un parcours (déjà en base) et une liste de prises. Ensuite, elle vient comparer la liste de prises avec les prises dans la base de données : si des prises ne sont pas présentes en base, elles les ajoutes, si elles l'étaient déjà, elle les met à jour. En revanche si des prises présentent en base ne sont pas dans la liste, elle les retire de la base. Des fonctions similaires existent pour récupérer et écrire les joueurs et scores en base.

II.3.4 Fonctionnement des jeux vidéo et le suivi des joueurs

Pour le moteur de jeu, nous avons choisi Pygame car il permet de régler le problème de l'interface et du moteur de jeu en même temps. Pygame utilisant le même repère de coordonnées qu'OpenCV, la communication entre MediaPipe et nos jeux s'en est trouvée grandement simplifiée.

Pour avoir une détection des joueurs ainsi que l'affichage du jeu en simultané, l'utilisation de la programmation concurrente était nécessaire pour garder une fluidité. Nous avons 3 jeux distincts : Cible, Parcours et Pong.

II.3.4.1 Jeux à 1 joueur

Pour les jeux à 1 joueur, nous avons une classe, *GameSinglePlayer* qui traite les données. Elle est appelée dans le thread principal, et appelle 2 nouveaux threads :

- *startSingleMediapipePose* qui démarre la caméra et MediaPipe
- *setPlayerPosition* qui récupère les coordonnées données par MediaPipe, les transforme, et les stocke.

II.3.4.1.1 Cible

Pour le jeu Cible, nous avons créé une classe du même nom, qui hérite de la classe *GameSinglePlayer.* Aléatoirement entre 2 et 5 secondes, une cible apparaît à un endroit aussi aléatoire. Le joueur a 5 secondes pour passer devant la cible.

II.3.4.1.2 Parcours

Pour le jeu Parcours, héritant aussi de *GameSinglePlayer,* Il est d'abord nécessaire que l'utilisateur créé et sélectionne un parcours grâce à la page correspondante, et fasse ensuite de même pour le joueur. Les interface de gestion des parcours et des joueurs réalisent de nombreux appels à la classe *Database*. Une fois fait, le jeu commence, et consiste pour le grimpeur à toucher les prises dans l'ordre, et ce, le plus rapidement possible. A la fin, son score et la date de sa performance sont enregistrés en base.

II.3.4.2 Jeux à 2 joueurs

Pour les jeux à 2 joueurs, nous avons la classe *GameMultiPlayer*. Elle est aussi appelée dans le thread principal, et appelle 4 nouveaux threads :

- *startResult* qui est appelé 2 fois (un pour chaque joueur). MediaPipe étant de base configuré pour ne traiter qu'une seule personne à la fois. La solution que nous avons trouvée pour pallier ce problème consiste à copier l'image reçue de la caméra pour avoir 2 images. Sur chaque image, nous mettons un masque noir pour cacher le joueur de droite ou de gauche, pour simuler le fait d'avoir 2 joueurs. Une fois les masques posés, nous traitons chaque image avec MediaPipe.
- *setPlayer1Position* et *setPlayer2Position* qui récupèrent les coordonnées données avec MediaPipe pour chaque joueurs, les transforment et les stockent

II.3.4.2.1 Pong

Pour le jeu Pong, nous faisons hériter la classe de *GameMultiPlayer*. Pour le calcul de la trajectoire de la balle après une collision, nous prenons le vecteur entre le centre de la position du membre du joueur, et le centre de la balle. La norme est définie par une vitesse fixée au préalable. Pour rendre le jeu plus fiable, la balle part toujours vers le camp adverse, peu importe où et comment on la touche

(voir Figure 9). De plus, la vitesse de la balle est calculée en fonction du temps, ce qui veut dire que la puissance d'un ordinateur n'agit pas sur celle-ci.

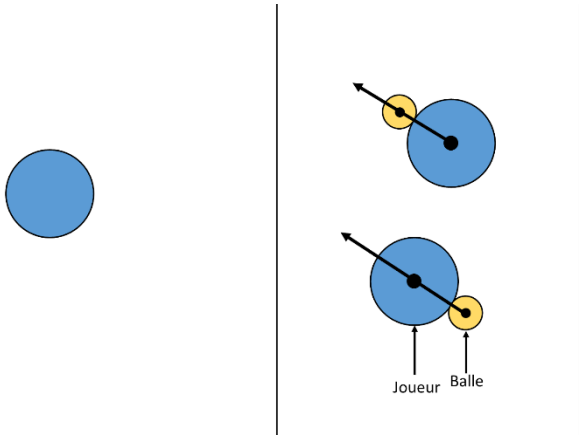


Figure 9 : Schéma de collision de la balle avec la main

III. RESULTATS

Le résultat final du projet est un logiciel, fonctionnel. Il permet donc de jouer à plusieurs jeux : cible, parcours, et pong. Il est possible de sauvegarder des parcours, des joueurs et des scores. Les jeux sont fluides et agréables à jouer. L'utilisation du logiciel est très simple et épurée.

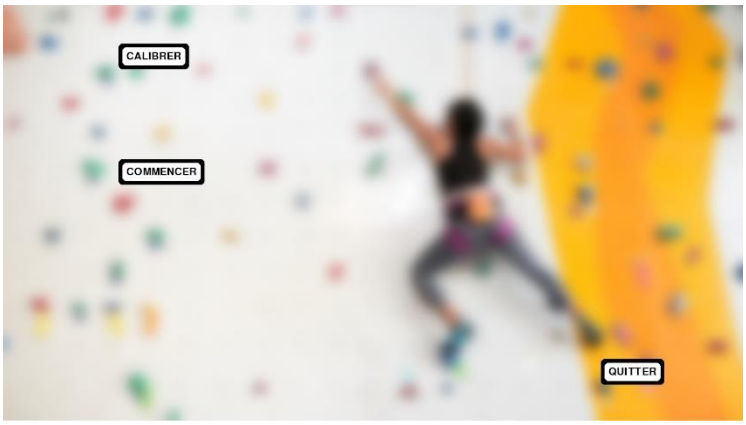


Figure 10 : Menu Principal

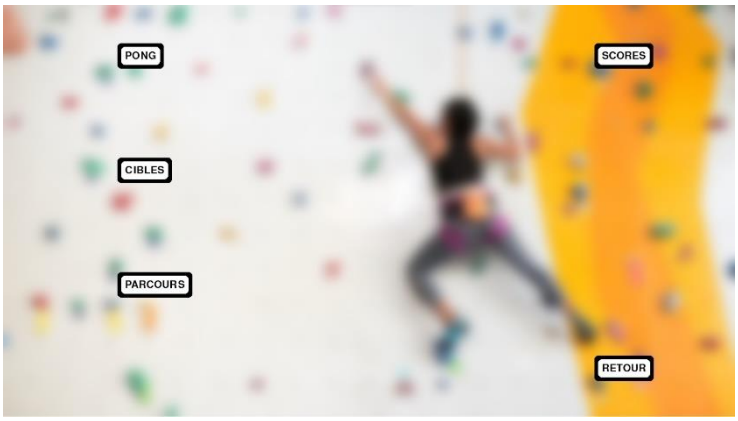


Figure 11 : Menu jeux

III.1 JEU PARCOURS

Figure 12 : Menu de sélection de parcours

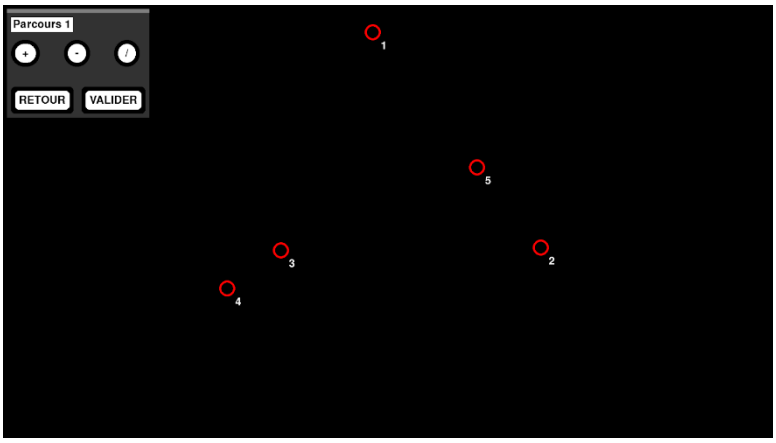


Figure 13 : Menu d'édition de parcours

En cliquant sur l'interface on peut rajouter de nouvelles prises au parcours. Il est possible de changer l'action du clique en sélectionnant le bouton « - » pour supprimer une prise ou le bouton « / » pour la déplacer. De plus, la fenêtre « boite à outils » est flottante pour permettre de placer des prises sur toute la surface de jeu. Enfin il est possible de renommer le parcours en tapant son nouveau nom avec le clavier.

Figure 14 : Menu de sélection du grimpeur

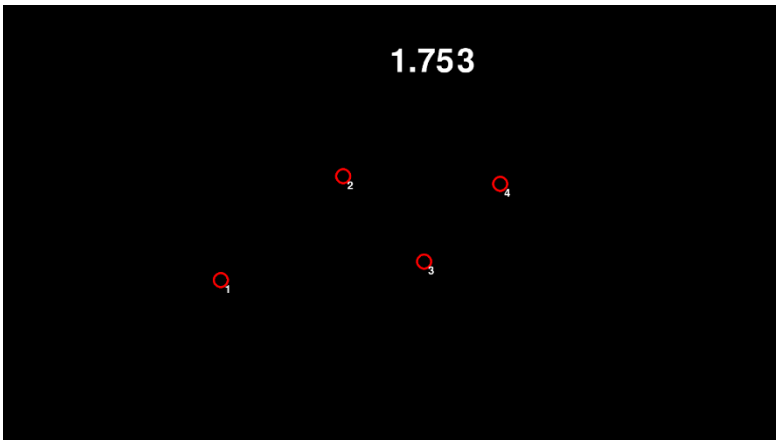


Figure 15 : Interface du jeu parcours

On peut observer le chronomètre en haut de l'interface, et les prises rouges sont celles qui ne sont pas encore validées. Elles deviennent vertes une fois que le joueur passe sa main ou son pied dessus. L'indice de la prise est affiché en bas à droite de celle-ci.

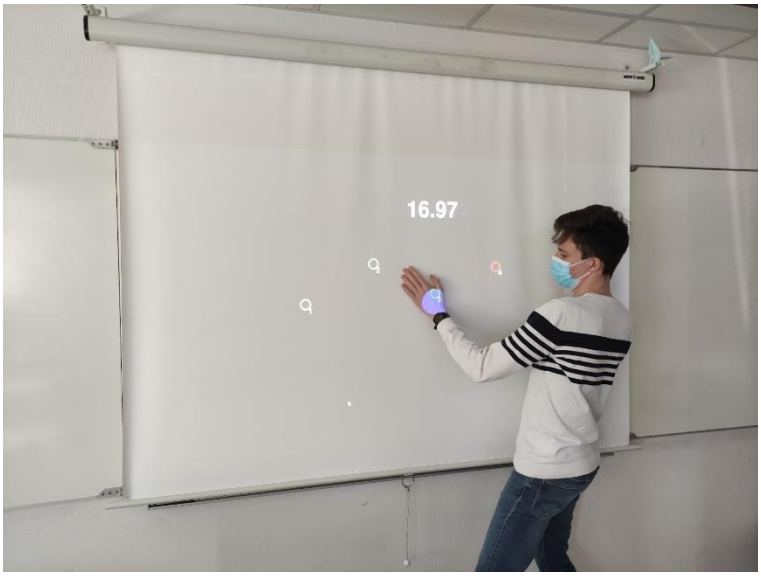


Figure 16 : Jeu parcours

III.2 JEU CIBLE

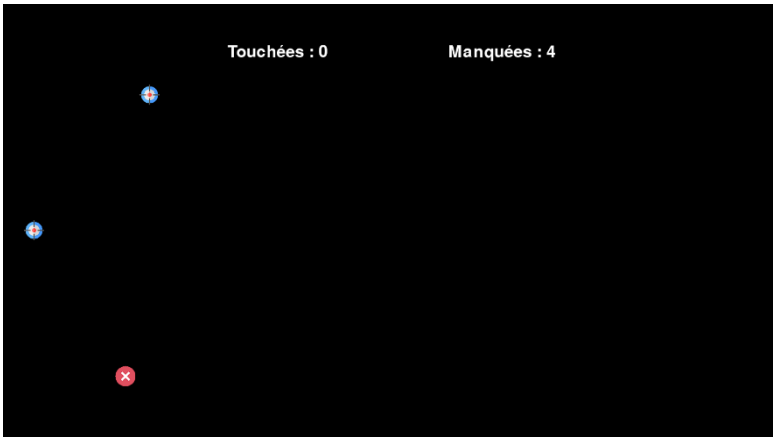


Figure 17 : Interface du jeu cible

Ici, le score est affiché en haut, tandis que des prises apparaissent partout sur l'écran, elle devienne rouge une fois que le temps est écoulé ou verte si le joueur a passé sa main ou son pied dessus.

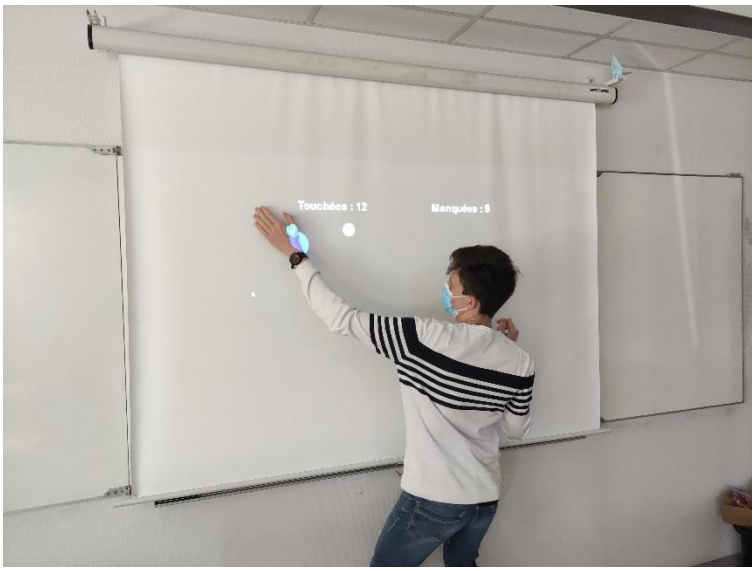


Figure 18 : Jeu cible

III.3 JEU PONG

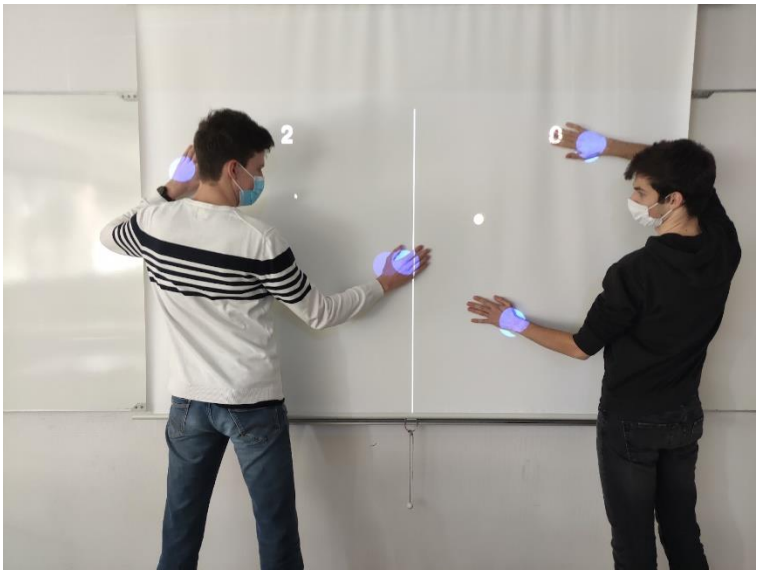


Figure 19 : Jeu pong

Le score de chaque joueur est affiché en haut, une barre centrale délimite la zone respective de chaque joueur.

III.4 RESULTATS GENERAUX

Pour comparer avec des logiciels concurrents, notre programme propose une vraie solution fiable et portable. Notre interface de jeu est plus minimaliste et notre catalogue de jeu plus réduit mais nous offrons une solution adaptée à de nombreuses salles là où nos concurrents sont limités à la seule salle où a été installé le logiciel. Il est aussi très facile de créer de nouveaux parcours.

Même si les caméras USB mettent un certain temps à s'allumer lors du démarrage du logiciel (environ 45 secondes), l'utilisation reste très optimisée. L'étalonnage du programme ne prend pas plus de deux secondes et la fluidité du logiciel a été notre priorité durant tout le projet.

De plus, notre logiciel aurait pu être amélioré sur certains aspects esthétiques pour paraître plus professionnel. L'utilisation de Pygame pour créer l'interface nous a contraint à faire des menus très simples. Nous avons préféré nous concentrer sur la partie jeu pour obtenir une expérience fonctionnelle quitte à passer moins de temps sur le développement des menus et des options.

Nous avons décidé de tester notre logiciel sur un vrai mur d'escalade. Cependant, le projecteur du premier test n'était pas du tout assez puissant pour projeter un mur en plein jour, la zone de test était donc très limitée. Toutefois les premiers résultats sont assez concluants, l'étalonnage et la détection des joueurs n'a pas été impactées par les murs composés de nombreuses couleurs. Nous avons également fait la demande auprès de l'IUT pour l'achat d'un projecteur adapté au plein jour pour essayer de tester dans de meilleures conditions avant la présentation du projet.

IV. CONCLUSION

IV.1 BILAN GENERAL

Ce projet est tout d'abord une mise en pratique de toutes les compétences acquises durant notre formation. Nous sommes fiers d'avoir obtenu un résultat pouvant répondre aux principaux besoins exprimés par M. Jaillet. Même si toutes les fonctionnalités ne sont pas présentes, le code présent permet de facilement intégrer de nouvelles options ou jeux.

Ce projet nous a aussi fait découvrir un nouveau langage (Python) et l'utilisation de librairies de traitement d'images (OpenCV et MediaPipe). Toutefois certains choix techniques nous ont fait perdre du temps : Pygame est certes très pratique pour les jeux mais ne facilite pas le développement de menus. Le développement sous C++ aurait laissé la possibilité de mieux configurer MediaPipe et même de développer des fonctionnalités selon nos besoins (Récupérer directement les coordonnées des deux joueurs).

Toutefois nos principaux objectifs ont été respectés : l'étalonnage, la détection des joueurs sont fonctionnels. L'interface correspond au cahier des charges, même si certaines fonctionnalités n'ont pas été implémentées car jugées peu nécessaires pour le temps accordé (ajout de certains jeux, gestions des scores dans tous les jeux…).

Enfin durant la réalisation du projet, nous avons reçu plusieurs demandes de la part de professeurs de sport pour utiliser notre logiciel. Ce projet semble donc susciter l'intérêt mais aussi répondre à un vrai besoin de la part des professionnels.

IV.2 BILAN PERSONNEL DE DALIL

Le projet tutoré m'a permis d'apprendre beaucoup de nouvelles technologies et de nouvelles manières de travailler. J'ai dû m'occuper de la partie Jeu. La partie back-end pour les 3 jeux, ainsi que le front-end pour les jeux « Cible » et « Pong ». J'ai donc proposé au groupe de partir sur Pygame, librairie Python reconnue, ce qui nous permettait de garder le même environnement pour MediaPipe et pour l'affichage.

La première difficulté rencontrée a été de devoir créer 2 fenêtres à l'aide de Pygame. J'ai passé énormément de temps sans finalement trouver de solution, ce qui m'a obligé à changer complètement la vision de l'interface désirée de base.

J'ai aussi rencontré un problème de fluidité, mes jeux, dû au traitement de l'image, étaient extrêmement lents. La solution trouvée a été d'utiliser le multi-threading pour partager les tâches, et avoir un gain certain en images par seconde. L'apprentissage en cours de la programmation concurrente m'a été très utile car cela m'a permis de savoir comment partager les données entre les différents threads sans pour autant qu'il y est un problème de synchronisation.

La principale difficulté du projet a été la structure de celui-ci, notamment le modèle Modèle Vue Contrôleur indiqué par notre tuteur. Nous n'avons jamais appris à faire une structure comme cela ce

qui nous a mis en difficulté pendant tout le projet, jusqu'à janvier environ lorsque nous avons eu les cours de Design Pattern.

Si le projet devait être refait, j'aurai utilisé la version 2.0 de Pygame, qui nous permet de créer plusieurs fenêtres. Pour le reste je pense n'avoir rien à changer.

IV.3 Bilan personnel de Guilhem

Ce projet m'a permis d'appliquer un grand nombre des savoirs que j'ai acquis au cours de mon DUT, en allant de la conception au développement, et en passant par la communication et le travail collaboratif.

Tout d'abord, à travers le choix d'implémenter notre projet en suivant la structure MVC, j'ai eu l'occasion de vraiment m'approprier ce design pattern sur un projet conséquent. Cependant, bien que M. JAILLET nous eût recommandé d'utiliser ce modèle de conception dès le début, nous n'avons réellement appris à l'utiliser qu'au mois de décembre, ce qui nous a mené à devoir restructurer l'ensemble du projet sur la fin, car nous avions mal compris le fonctionnement du modèle MVC avant. Notamment, les échanges entre les parties et notamment avec la partie vue ont été à revoir. Etant en charge de la restructuration, j'ai passé beaucoup de temps à adapter notre code à ce nouveau modèle. A termes, bien qu'il nous ait permis de gagner du temps lors de la correction de bugs, en ayant un code plus propre et modulaire, je me rends désormais compte que son intérêt est assez limité pour des projets comme celui-ci, ne s'étalant pas sur une période suffisamment importante. En effet, comme nous passons plus de temps à concevoir et développer le produit qu'à le maintenir, le gain en temps est à la fin assez faible. C'est d'ailleurs pour cette raison que j'ai fini par mélanger la partie modèle et vue, par exemple, pour la classe joueur, qui stocke à la fois les informations du grimpeur, et les méthodes nécessaires à son affichage. Si je devais refaire le projet, je pense que j'aurais codé de manière plus directe, dans un premier temps, afin d'avoir d'abord quelque chose de convainquant pour le client, puis, dans un second temps, de manière plus structurée, pour les développeurs. Il est néanmoins important de noter que c'est surtout notre manque de maitrise du design pattern MVC au début qui a joué sur le fait que notre architecture a été autant à revoir, ce qui n'arriverai sûrement pas dans un projet futur. J'en retiens donc qu'il important de bien comprendre chaque tenant en aboutissant des solutions et méthodes proposées lors d'un projet, afin de réaliser un code efficace, structuré, et compréhensible sur l'ensemble de sa durée de vie.

Ensuite un autre problème rencontré a été celui de l'over-engineering. En effet, beaucoup de code a été fourni, pour des fonctionnalités qui n'ont finalement soit pas vu le jour, soit été sous-exploitées, soit en ont perdu en souplesse.

Pour commencer, nous ne sommes pas arrivés à assez bien définir le cadre du sujet, dès le départ, ce qui nous a conduit à implémenter des fonctionnalités « au cas où », ou « dans l'espoir de… », pour ne finalement jamais les utiliser. Notamment, nous souhaitions au départ offrir la possibilité à nos utilisateurs de stocker des murs d'escalade (contenant toutes les prises), et de stocker des parcours, dans ces murs, ne contenant qu'une partie des prises. Ceci m'a mené à concevoir une base de données pour répondre à ces besoins, et à implémenter toutes les méthodes nécessaires à la sauvegarde des éléments en Python. Cependant, par manque de temps, et par simplicité, nous nous sommes rendu compte après coup seulement que le stockage des murs ne s'avérait pas nécessaire, et qu'une partie de mon travail n'était finalement pas utile. Et plus nous avancions dans le projet, et meilleur en était

notre compréhension, et avec elle, l'indentification des fonctionnalités moins prioritaires, voir inutiles. Cependant, ceci aurait pu être évité, si nous avions mieux identifié les besoins et su les catégoriser par ordre d'importance dès le départ (ce que nous avons fait, mais visiblement pas assez judicieusement). Nous ne nous sommes pas assez projetés, et avons été trop ambitieux par moments.

De même, après avoir restructuré le projet en MVC, je suis passé sur la partie interface, et ai développé de nombreuses classes, très modulaires, et facilement réutilisables partout dans le projet, ce qui m'a pris beaucoup de temps. Pour autant, la plupart de ces classes n'ont été utilisé que de rares fois (par exemple, les fenêtres internes flottantes, ou encore les champs textuels). J'ai cherché à optimiser avant d'implémenter, ce qui m'a conduit à développer des classes surdimensionnées par rapport aux besoins réels. Cependant, tout ce travail m'a permis d'apprendre énormément, autant au niveau des conséquences citées avant, qu'en terme de connaissances conceptuelles de programmation, et sur le langage Python.

Enfin, en voulant bien faire, j'ai parfois fait perdre de la souplesse au code général, en offrant moins de contrôle aux autres développeurs, qui souhaitaient utiliser mes fonctions dans leurs parties respectives. C'était notamment le cas des fonctions de sauvegarde du modèle dans la base de données. En effet, comme nous l'avons déjà précisé plus haut, ces fonctions intégraient déjà toutes les fonctionnalités CRUD. Je me rends cependant désormais compte, avec le recul, qu'il aurait sans doute été plus simple de laisser ces fonctionnalités séparées et directement accessibles, plutôt que de les regrouper et d'en bloquer l'accès direct à ceux en ayant besoin (ce qui a d'ailleurs été les cas, et m'a finalement mené à repasser certaines méthodes en public), ce qui, en plus de leur offrir plus de contrôle, m'aurait fait gagner un temps précieux, et aurait même pu améliorer les performances dans certaines conditions.

Un autre élément clé de ce projet a d'ailleurs été le travail en équipe, et avec comme maître mot la communication. Tout au long du projet, nous avons dû communiquer entre nous, au sein du groupe, ce qui n'a pas posé de problème particulier, y étant habitué mais aussi avec M. JAILLET au travers des comptes-rendus et rendez-vous hebdomadaires. Cette dernière phase a vraiment été une nouveauté importante pour nous, lors de la réalisation d'un projet. Bien que contraignante, car nous obligeant à rédiger des rapports en respectant les nombreuses échéances, ce fût un concept intéressant, nous forçant à travailler sur le projet chaque semaine, et nous permettant de mieux planifier les tâches à réaliser. Toujours dans la même idée, l'utilisation de Git nous a permis d'avancer plus efficacement sur nos parties respectives, et ma maîtrise de cet outil s'est grandement améliorée par rapport à l'année dernière. Je pense cependant que la répartition du travail a été un peu moins équitable sur la fin, me retrouvant seul sur la partie interface, qui a mes yeux a été une des parties les plus conséquentes du projet (bien que cela soit aussi dû, comme je l'ai déjà dit, à mon entêtement à vouloir faire quelque chose de trop optimisé et finalement surdimensionné. J'aurais ici dû accorder plus de confiance à mes camarades qui m'ont pourtant plusieurs fois averti sur le fait je cherchais à faire plus que nécessaire, erreur que j'éviterai de répéter à l'avenir).

Enfin, le développement de notre application m'a vraiment permis d'accroître mes connaissances, et ce, dans de nombreux domaines.

En premier lieu, la conception et la construction de la base de données était une première pour moi sur un tel projet. C'est d'ailleurs pour cette raison que j'ai décidé de prendre en charge cette partie. Ça a été pour moi l'occasion de mettre en pratique mes connaissances, mais surtout de les consolider.

Ayant eu quelques difficultés à faire fonctionner correctement la base avec Python, j'ai eu l'occasion de parcourir la documentation de sqlite3, ce qui a amélioré ma capacité à chercher des solutions par moi-même.

Par ailleurs il s'agissait également pour moi de ma première utilisation de Python. Et, bien que ce langage m'ait beaucoup repoussé au départ, car proposant une approche très différente de ce à quoi C++ et Java m'avaient habitué (presque pas de niveau de visibilité, compliquant ainsi la mise en place d'un code sécurisé, ou de design patterns comme le Singleton), j'ai, au fur et à mesure de mon utilisation, appris à l'apprécier. En effet, en cherchant bien, le langage offre de nombreuses fonctionnalités très avancées, pouvant non seulement remplacer celles des autres langages, mais allant même jusqu'à les améliorer, dans certains cas. C'est par exemple le cas des mots clés @abstractmethod, @property, ou encore des metaclass.

Pour finir, la programmation de l'interface graphique a représenté pour moi la plus grosse difficulté. Pas temps d'un point de vue technique, mais surtout d'un point de vue conception, car c'était une partie tentaculaire, reliée à toutes les autres. De plus, contrairement à des librairies à l'instar de Swing pour Java, Pygame ne propose que peu de fonctionnalités, très primaires. Ainsi, j'ai dû réimplémenter par moi-même un ensemble de classes similaires à celles constituant la base de Swing, avec la gestion de listeners, de boutons, de listes et de champs de textes, seulement avec les évènements systèmes, et la représentation de formes géométrique et d'images. Comme dit plus haut, je me rends désormais compte qu'aller aussi loin dans les outils était très certainement plus que nécessaire, et très couteux en temps, d'autant que cela fait partie des choses que l'utilisateur final ne voit pas, et qui sont donc, en quelques sorte, superflues. L'utilisation d'une autre bibliothèque plus adaptée, comme PySide ou PyQt4, aurait peut-être pu me faire gagner un temps précieux, ce qui m'a appris qu'il était important de bien sélectionner mes solutions techniques, avant de commencer un projet.

Sommes toutes, j'ai retiré beaucoup d'enseignements de ce projet. Ainsi, je me rends désormais compte qu'il vaut mieux chercher à d'abord avoir un code fonctionnel, avec un rendu rapide pour le client, puis seulement après, chercher à avoir un code optimisé, et maintenable. Bien entendu, il faut que le code reste lisible, et performant, mais pour des projets qui ne sont pas voués à avoir une durée de vie longue, et être repris par d'autres développeurs, chercher à le rendre accessible et maintenable est finalement plus une perte de temps qu'autre chose. Bien entendu, je ne regrette pas pour autant d'avoir fourni plus de travail que nécessaire, car cela m'a beaucoup appris, que ce soit du côté technique ou que du côté méthodologique. Pour conclure, je dirai que bien que nous pourrions encore ajouter de nombreuses fonctionnalités à notre projet, je suis satisfait du résultat que nous avons proposé, puisque nous avons répondu à l'ensemble des consignes non facultatives imposées par le projet.

IV.4 Bilan personnel de Baptiste

Durant le projet tuteuré j'ai principalement géré la partie étalonnage et détection des prises. J'ai notamment été confronté aux problèmes de matrice de transformation. De nombreuses solutions étaient mises en avant mais ne correspondaient pas à nos besoins. De plus, MediaPipe est encore sous sa version bêta, même si le logiciel est très puissant, la documentation était très maigre. J'ai donc pris une partie de mon temps à comprendre le fonctionnement et la configuration nécessaire à l'étalonnage et la transformation des points.

Pour la détection des points, j'ai également eu du mal à trouver la bonne fonction proposée par OpenCV. J'ai par exemple essayé de projeter un rectangle de couleur puis de détecter ses 4 coins mais les résultats étaient peu concluants. Enfin, même lorsque j'ai découvert la fonction *findHomography*, j'ai rencontré des difficultés à trouver la configuration optimale (transformation de l'image, sélection de l'image de repère…). Lorsque j'ai terminé la partie étalonnage et transformation des points j'ai également rencontré des difficultés à continuer la partie interface, déjà commencée par Dalil et Guilhem car je n'avais pas assez prêté attention à leurs tâches et leur avancement. Enfin, sur l'ensemble du projet, l'utilisation du modèle MVC dans le projet tuteuré avant de réellement comprendre son fonctionnement en cours, nous a obligé à reconstruire tout le projet correctement.

Si le projet était à refaire aujourd'hui, je n'aurais, je pense, rien changé sur la partie étalonnage et transformation des points. J'essayerais de plus m'informer du travail des autres membres de l'équipe pour pouvoir récupérer leur travail sans perdre trop de temps à comprendre le fonctionnement du projet global.

En tant que développeur junior, ce projet m'a fait utiliser toutes les compétences de développeur accumulées jusqu'au projet. J'ai pu apprendre à effectuer des recherches et sélectionner les meilleures technologies, répondant aux besoins. J'ai également découvert le travail d'équipe et l'utilisation des outils de développement en groupe comme Git sur de plus longs projets que ce que nous avions fait durant les semaines spéciales.

Pour conclure, cette expérience m'a fait découvrir Python. Souhaitant m'orienter vers la cyberdéfense, ce langage est très utilisé dans le pentest. Certes les librairies de traitement d'images ne me serviront probablement pas mais elles m'ont permis d'apprendre à chercher dans une documentation. De plus, comme mon stage se fera avec des langages non étudiés à l'IUT, le projet tutoré aura amélioré mes compétences d'apprentissage en autonomie.

V. REFERENCES

Foundation, P. S. (s.d.). *Python 3.10.2 documentation*. Récupéré sur Python: https://docs.python.org/3/

Google. (s.d.). *Mediapipe*. Récupéré sur Github: https://google.github.io/mediapipe/

OpenCV. (s.d.). *Detection of ChArUco Boards.* Récupéré sur OpenCV: https://docs.opencv.org/3.4/df/d4a/tutorial_charuco_detection.html

OpenCV. (s.d.). *Feature Matching + Homography to find Objects.* Récupéré sur OpenCV: https://docs.opencv.org/3.4/d1/de0/tutorial_py_feature_homography.html

Pygame. (s.d.). *Pygame*. Récupéré sur Pygame: https://www.pygame.org/

Shaikh, R. (s.d.). *OpenCv Perspective Transformation.* Récupéré sur Medium: https://medium.com/analytics-vidhya/opencv-perspective-transformation-9edffefb2143#:~:text=Perspective%20Transform%20is%20a%20feature,Transformation%20is%20applied%20to%20it.

RESUME EN FRANÇAIS :

Rapport du projet tuteuré : escalade en réalité augmenté réalisé par Dalil MAADOUR, Guilhem RICHAUD et Baptiste STUBLAR durant l'année 2021/2022

Le projet consiste à réaliser un logiciel permettant de pratiquer l'escalade en réalité augmentée. L'objectif était ainsi de proposer une expérience étendue de ce sport, au travers de jeux vidéo grandeur nature.

Certaines contraintes étaient à respecter : n'importe qui devait être en mesure d'utiliser le logiciel, ce qui impliquait utilisation de matériel simple : une caméra pour détecter la position des joueurs, un vidéoprojecteur pour diffuser les jeux sur le mur d'escalade, et un ordinateur pour le traitement. L'objectif était de proposer une interface simple permettant à l'administrateur de gérer les pistes et les grimpeurs, et de lancer différents jeux, facilement réalisables sur un mur d'escalade

Vous trouverez dans ce rapport, une présentation de l'équipe, puis les difficultés rencontrées durant tout le projet. Nous verrons ensuite les solutions apportées aux problèmes et enfin un aperçu du résultat.

MOTS CLES :

Evènement
Parcours
Escalade
Projection
Détection de mouvement

MATÉRIEL / LOGICIEL / MÉTHODE UTILISÉ(E)(S) :

OpenCV
MediaPipe
Pygame
Python
Caméra
Projecteur