LYCÉE JEAN PERRIN, MARSEILLE

TERMINALE SCIENTIFIQUE

SPÉCIALITÉ SCIENCES DE L'INGÉNIEUR

Rapport de Projet

Auteurs :
Fabien CAYLUS
Thibault GRIMALDI
Taki HAZAM
Mathieu MERTINY

Superviseurs :
M. AMBROIS
M. CORCORAL
M. DESAUTEL
M. SAHAKIAN

4 mai 2015

Virtual Walker

« Rendre la Réalité Virtuelle toujours plus immersive »

Ce rapport a été rédigé dans le cadre du PPE (Projet Pluridisciplinaire Encadré) de Terminale Scientifique spécialité Sciences de l'Ingénieur et a pour but de présenter, dans son intégralité, notre réalisation, le projet Virtual Walker. Vous pourrez donc y trouver aussi bien une présentation simple et compréhensible au plus grand nombre que des détails poussés dans les domaines de l'ingénierie et de l'informatique. La seule lecture de ce rapport devrait donc permettre d'appréhender l'ensemble des éléments ainsi que la finalité du projet Virtual Walker.

De plus, dans le cadre des Olympiades des Sciences de l'Ingénieur, nous avons été choisis pour représenter le lycée Jean Perrin lors des qualifications académiques d'Aix-Marseille et avons fini 3ème ex-æquo (sur 33 groupes en compétition).

Bonne lecture.

Table des matières

Première partie .

Présentation générale

1. Description du projet

Le projet Virtual Walker est un système permettant de plonger l'utilisateur dans un environnement 3D grâce à une immersion visuelle mais aussi sensitive. En effet, afin de faire croire à l'utilisateur qu'il est réellement présent dans un environnement virtuel, le projet Virtual Walker utilise la technologie de la réalité virtuelle. Celle-ci permet de créer un visuel 3D à partir d'un simple écran, un peu à la manière de ce que font les télévisions 3D.

De plus, pour que l'utilisateur puisse se déplacer dans cet environnement, nous avons imaginé un système lui permettant de se déplacer physiquement (mais sur place) afin que ses mouvements soient retranscrits en déplacement dans l'environnement virtuel.

L'objectif principal du projet est donc d'immerger le joueur de manière totale afin d'améliorer son expérience et ses sensations.

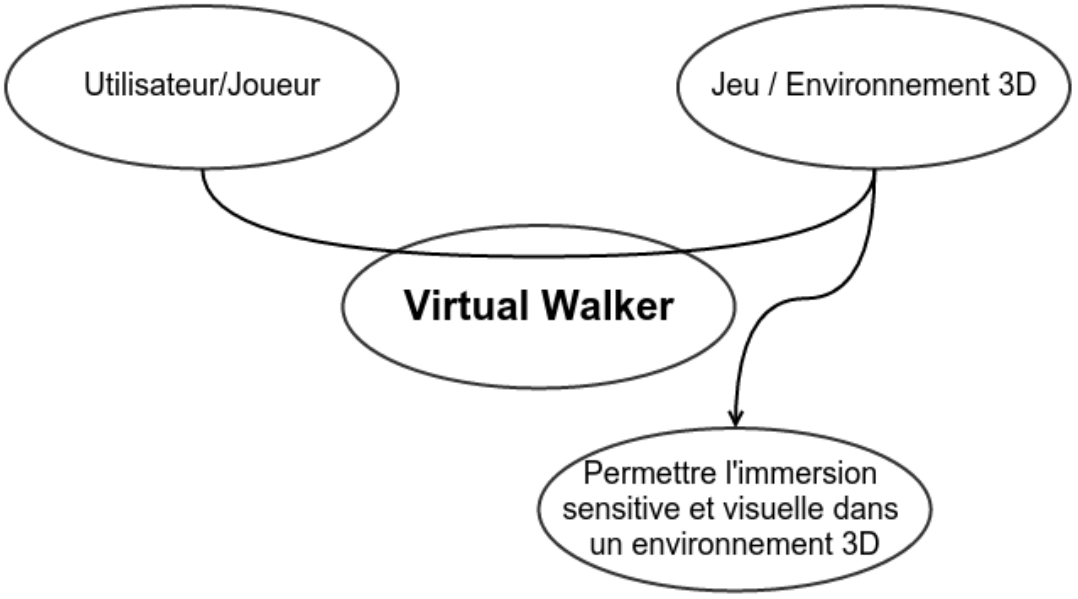


FIGURE 1.1.: Diagramme « Bête à cornes »

1.1. Fonctionnement

Afin de pouvoir réaliser cette immersion, nous avons décidé de diviser le projet en 4 grandes parties qui devront interagir les unes avec les autres. Les deux premières sont des réalisations matérielles tandis que les deux dernières sont des réalisations informatiques :

— **Le support du joueur :** Il a pour but de maintenir le joueur en sécurité, tout en lui permettant de courir « sur place ».
— **Le casque de réalité virtuelle :** Le joueur devra le porter afin d'obtenir l'immersion visuelle en 3 dimensions. Un smartphone sera utilisé en guise d'écran et d'unité de traitement.
— **Le logiciel de détection de mouvements :** Permet d'analyser les mouvements de l'utilisateur afin de les envoyer au casque (plus précisément au téléphone qu'il contient).
— **L'application pour smartphone :** Devra être exécuté sur le téléphone à l'intérieur du casque et est chargée d'afficher l'environnement 3D à l'écran et de recevoir les données de mouvements depuis le logiciel de détection.

Voici un schéma permettant de mieux comprendre l'utilité de chaque élément étape par étape :

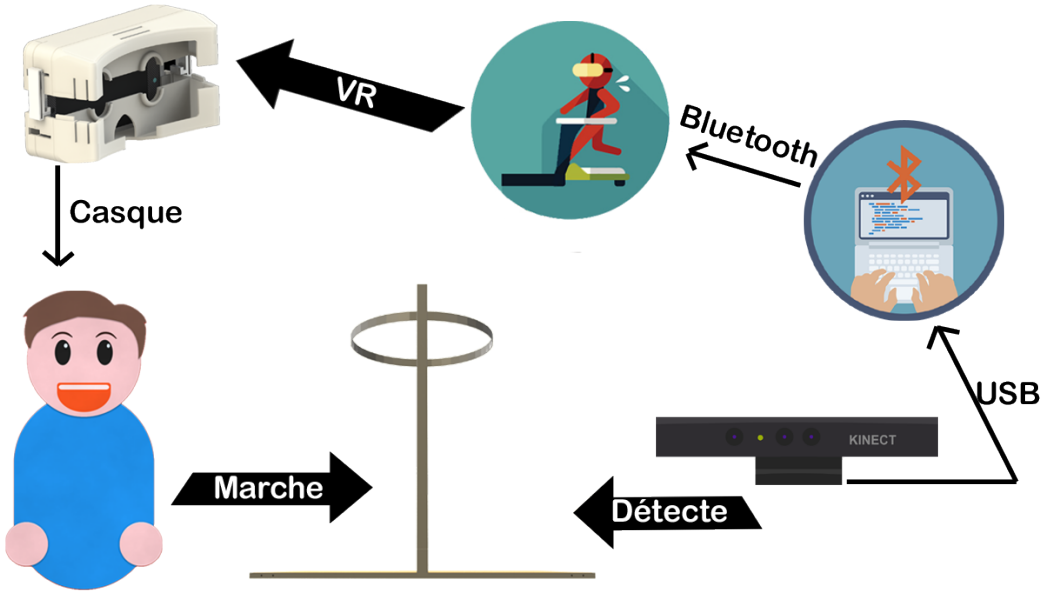


FIGURE 1.2.: Organigramme de présentation du projet

De ce schéma, on peut déduire le cheminement d'utilisation suivant :

1. L'utilisateur se déplace à l'intérieur du socle.
2. A l'aide d'une caméra Kinect reliée à un ordinateur, le logiciel détecte les mouvements de l'utilisateur.
3. Ces mouvements sont envoyés par Bluetooth vers l'application mobile.
4. L'application mobile affiche l'environnement 3D à l'écran et retranscrit les mouvements de l'utilisateur.
5. Le casque de réalité virtuelle transforme l'image du téléphone en image 3D à l'aide d'un système de lentille.
6. L'utilisateur (qui a le casque sur la tête) a donc eu la sensation de s'être déplacé.

L'ensemble de ce cheminement est réalisé des dizaines de fois par seconde, afin que l'utilisateur n'ait pas conscience de sa présence, et qu'il soit donc immergé un maximum.

1.2. Motivations

Les motivations qui nous ont poussé à la réalisation de ce projet sont multiples. En effet, lors de la phase initiale de recherche, nous avons exploré divers domaines conciliant les capacités de chacun des membres du groupe, c'est-à-dire la mécanique et l'informatique.

Partant de ce principe, nous avons rapidement dérivé sur la réalité virtuelle et sur certains projets existants comme le *Google Cardboard* ou *l'Occulus Rift*. Mais nous avons constaté un détail récurrent dans tous ces projets : ils ne proposent qu'une immersion visuelle à l'aide d'un casque et les mouvements doivent être contrôlés à l'aide d'une manette.

Nous nous sommes alors dis qu'il serait intéressant de combiner la réalité virtuelle avec un système de détection des mouvements pour que l'utilisateur puisse enfin se déplacer physiquement dans son jeu favori, ou dans un environnement choisi.

Le projet Virtual Walker était né !

1.3. Potentiel

Le projet que nous avons réalisé est un projet fonctionnel, mais il peut quand-même être considéré comme un prototype, un exemple de ce qui peut être réalisé.

En effet, nous pensons qu'au delà de la réalisation technique, le concept que nous avons développé tout au long de ce projet possède un fort potentiel.

Comme vous le verrez dans la suite de ce rapport, l'environnement que nous avons choisi de modéliser est orienté jeu-vidéo. Il est par exemple envisageable de prolonger cette idée en permettant aux joueurs de jouer à leurs jeux préférés en totale immersion. Qui n'a jamais rêvé d'être dans la peau de son personnage préféré ? d'éviter les obstacles comme Indiana Jones ? de partir au combat comme dans Counter-Strike ? ou même de sauter de plate-forme en plate-forme comme Mario ?

De plus, le projet Virtual Walker pourrait très bien être utilisé dans un but commercial. Nous pensons notamment au domaine de l'architecture et de la construction. Tout le monde sait aujourd'hui que tous les architectes commencent toujours par créer des maquettes numériques de leurs constructions. Pourquoi ne pas les rendre compatibles avec notre système afin que le client puisse visiter son futur appartement avant même qu'il soit construit ?

Le domaine de l'éducation et de la culture pourrait aussi être impacté. On pourrait imaginer une visite virtuelle du Louvre depuis n'importe où en France (ou à l'étranger), où juste une salle avec l'équipement adéquat, serait nécessaire. Dernièrement, on entend aussi beaucoup parler des reconstitutions de grottes préhistoriques afin de préserver les originales. Si notre système était amélioré, il pourrait sans doute remplacer ces volumineuses reconstructions par de simple représentations numériques accessibles à tous.

Ces quelques lignes suffisent à montrer que le potentiel est presque infini tellement les domaines d'applications sont vastes

2. Contraintes

Comme spécifié au début de ce document, se projet a été réalisé dans le cadre du PPE de Terminale S spécialité Sciences de l'Ingénieur. De ce fait, nous avons dû composer avec plusieurs contraintes qui en découlent.

2.1. Délais et Matériels

Pour la réalisation de ce projet, environ 70 heures au lycée nous ont été accordées. Cependant, le majorité du travail a dû être fait en dehors de ce laps de temps pour plusieurs raisons, que ce soit au niveau des contraintes matérielles (pour la réalisation du socle) ou du temps nécessaire à la réalisation (pour la partie informatique par exemple). Ainsi, même si le projet est considéré comme fini, il y a certains éléments que nous considérons comme améliorables, comme l'algorithme de détection des mouvements, l'esthétique du support et la prise en charge du son sur le casque.

De plus, comme tout projet, nous avons dû faire attention aux coûts nécessaires à la réalisation, et utiliser au maximum ce que le lycée a mis à notre disposition. Comme vous le verrez dans le suite de ce document, cela explique l'utilisation de l'imprimante 3D ou bien de la caméra Kinect.

2.2. Pluridisciplinarité

Comme son nom l'indique, le Projet Pluridisciplinaire Encadré doit s'inscrire dans une démarche de pluridisciplinarité, que ce soit au niveau des sciences de l'ingénieur ou de l'ensemble des matières de l'enseignement.

Nous avons donc croisé les disciplines suivantes dans notre projet :

- **Sciences de l'Ingénieur :**
  - Mécanique (support du joueur, casque)
  - CAO (Conception Assistée par Ordinateur, *SolidWorks*, ...)
  - Tribologie (étude des frottements pour le support)
  - Informatique (programmation du logiciel de détection et de l'application mobile)
- **Physique :** Optique (lentilles pour le casque)
- **Anglais :** Largement utilisé dans toute la partie informatique

Deuxième partie .

Support du Joueur

3. Support : Objectifs et étude de marché

3.1. Objectifs

Cette partie du projet consiste à réaliser un support servant à accueillir le joueur afin que celui-ci puisse interagir avec le jeu ou l'application tout en étant en sécurité. Le joueur doit pouvoir être stable une fois installé sur le support, et doit pouvoir se mouvoir.

Il s'agit donc ici d'une partie quasiment entièrement mécanique qui constitue l'une des deux réalisations matérielles de ce projet.

Afin d'illustrer la fonction principale du support en voici un diagramme dit « Bête à cornes » :

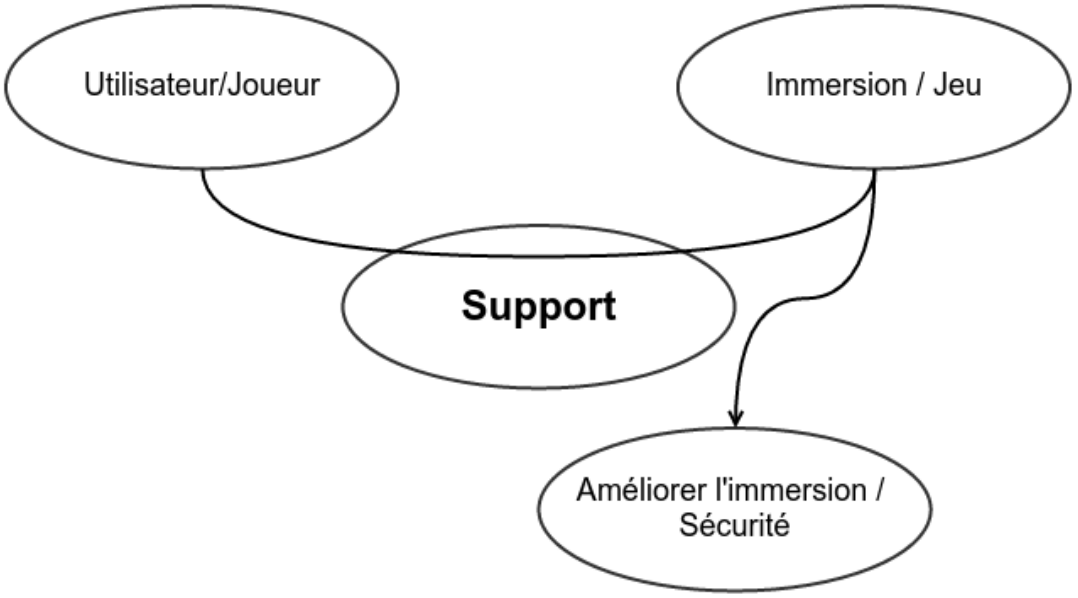


FIGURE 3.1.: Diagramme « Bête à cornes » du support

Pour mieux comprendre les objectifs du support, voici son actigramme aussi appelé « SADT » :

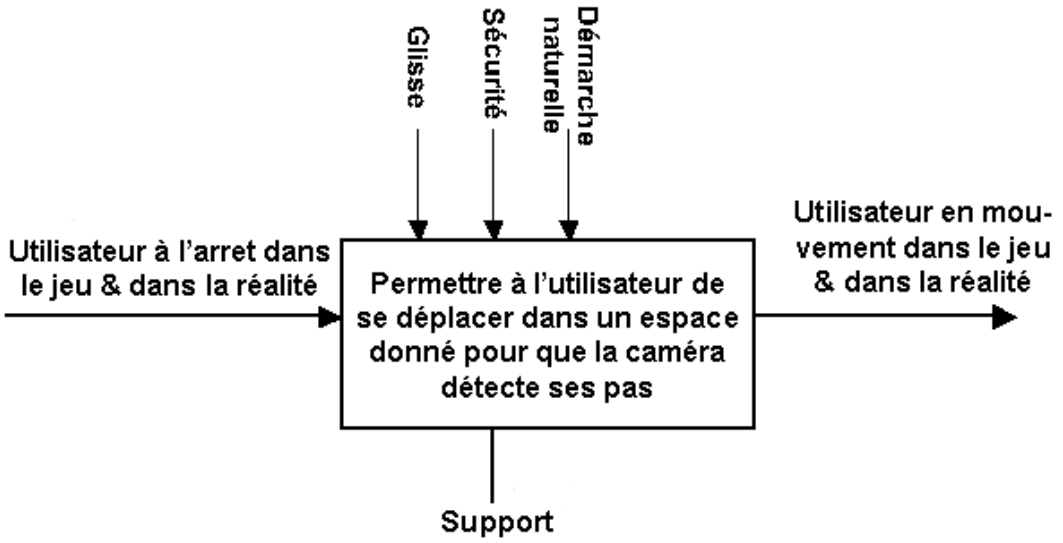


FIGURE 3.2.: Diagramme « SADT » du support

3.2. Solutions existantes

Avant de commencer l'étape de la conception, nous nous sommes intéressés aux solutions déjà existantes sur le marché. A l'heure actuelle, il existe deux projets du même type, mais aucun d'eux n'est encore à l'étape de production et de mise en circulation du projet. Ils se nomment :

— Cyberith Virtualizer
— Virtuix Omni

Voici une présentation de chacun, afin d'observer les avantages et défauts qu'ils possèdent.

3.2.1. Cyberith Virtualizer

Chez Cyberith, avec leur Virtualizer, le support est principalement constitué d'une partie très glissante, et d'un « arceau », dans lequel vient se placer le joueur afin que celui ci puisse se déplacer tout en restant dans un espace délimité.

FIGURE 3.3.: Support « Virtualizer » de Cyberith

Pour que l'utilisateur ait l'impression de marcher, Cyberith a doté son support d'une partie glissante. Cyberith a aussi muni son Virtualizer de vérins, cachés dans les montants, afin que le joueur puisse sauter tout en étant en sécurité.

FIGURE 3.4.: Virtualizer : surface glissante

FIGURE 3.5.: Virtualizer : sauts

Le Virtualizer est composé en grande partie d'acier et il n'est pas totalement démontable. La détection des mouvements se fait à l'aide de divers capteurs intégrés à l'arceau et aux montants.

3.2.2. Virtuix Omni

L'Omni de Virtuix utilise une approche totalement différente. En effet, le support n'est pas très glissant en lui même. Il est assez grand et possède une forme « en bol »Ċomme le Virtualizer, l'Omni possède un « arceau », par contre cet « arceau »peut s'ouvrir afin de faciliter l'entrée du joueur.

FIGURE 3.6.: Support « Omni » de Virtuix

Pour que le joueur ait l'impression de marcher, Virtuix a choisi de ne pas utiliser le caractère glissant du socle. Ici, le joueur doit se munir de chaussures spéciales, avec

une semelle en plastique dur, possédant une sorte de crochet. La surface du support est constituée de rainures assez profondes, dans lesquelles le joueur va y glisser le crochet des chaussures afin de créer une sensation de marche tout en restant à l'arrêt.

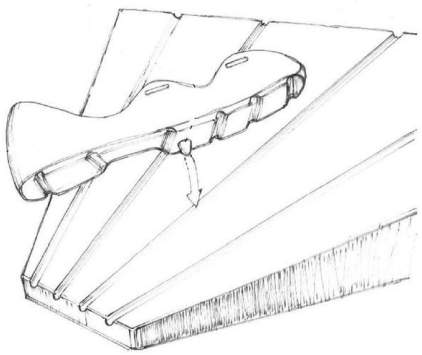


FIGURE 3.7.: Chaussures spéciales de l'Omni

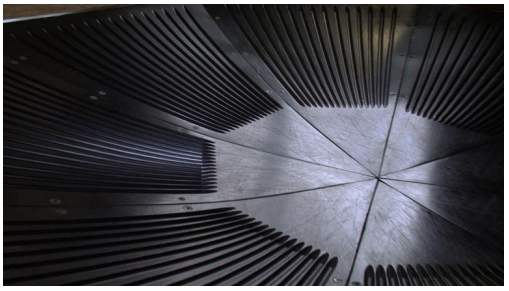


FIGURE 3.8.: Forme « en bol » de l'Omni

Ici tout est fixe (mis à part l'ouverture de l'arceau), et l'Omni est principalement constitué de plastique dur (type ABS) ainsi que de bois. Pour la détection des mouvements, une caméra extérieure au support est utilisée et c'est cette solution que nous retiendrons pour la détection des mouvements.

4. Conception et réalisation du support

4.1. Solutions envisagées

En observant ce qui se fait actuellement, nous avons pu émettre une solution partielle pour la réalisation d'un support pour le projet Virtual Walker.

Le socle devra donc comporter certaines caractéristiques importantes :
- Il sera muni d'un arceau, qui permettra à l'utilisateur de ne pas sortir hors de la zone de couverture du logiciel de détection des mouvements. Celui-ci devra être assez résistant et de dimensions acceptables pour tout type de morphologie.
- Il sera muni d'un sol glissant, soit à la manière du Virtualizer, soit par l'ajout d'une autre réalisation telles que des chaussures comme sur l'Omni.
- Il devra glisser un maximum afin que l'on ait une reproduction naturelle de la marche lors de son utilisation.
- Il sera démontable, afin d'assurer son déplacement et son transport.
- Il aura des dimensions suffisantes afin que l'on ne soit pas obligé de changer notre manière de marcher.
- Il sera constitué de matériaux résistants comme ceux utilisés par le Virtualizer et l'Omni.

Pour aboutir au socle que nous avons finalement réalisé, de nombreuses étapes ont été nécessaires. Il y eu plusieurs phases de développement, qui proposait chacune des solutions techniques différentes.

Nous avons donc étudié toutes les possibilités que nous connaissions afin d'obtenir un socle qui respecte au maximum les caractéristiques que nous venons d'exprimer (arceau, sol glissant, démontable, dimensions suffisantes, matériaux résistants).

Ainsi, plusieurs versions ont été imaginées, chacune possédant ses propres caractéristiques :

- **Version Alpha :** Version avec des tôles inclinées
- **Version Bravo :** Version avec des billes/rouleaux de manutention
- **Version Charlie :** Version avec un tapis motorisé
- **Version Delta :** Version avec un sol du type « Air Hockey »
- **Version Écho :** Version Alpha avec plans horizontaux en bois
- **Version Foxtrot :** Version finale, retenue et réalisée

4.1.1. Version Alpha

La Version Alpha est la toute première version du support que nous avons imaginé. C'est une version assez basique, puisqu'elle n'est pas du tout démontable.

Elle se compose d'une base de forme triangulaire, constituée de barre IPN soudées, de plans inclinés en métal, et d'un arceau en profilé tubulaire cintré. L'utilisateur doit utiliser des chaussettes ou des patins, le plan incliné en métal devant créer une sensation de marche. Il permet au joueur de se déplacer à 180°.

Cette version est inspirée de *l'Omni*, et de sa forme en « bol » . Elle a été abandonnée par la suite.

Voici quelques rendus et dessins de cette version :

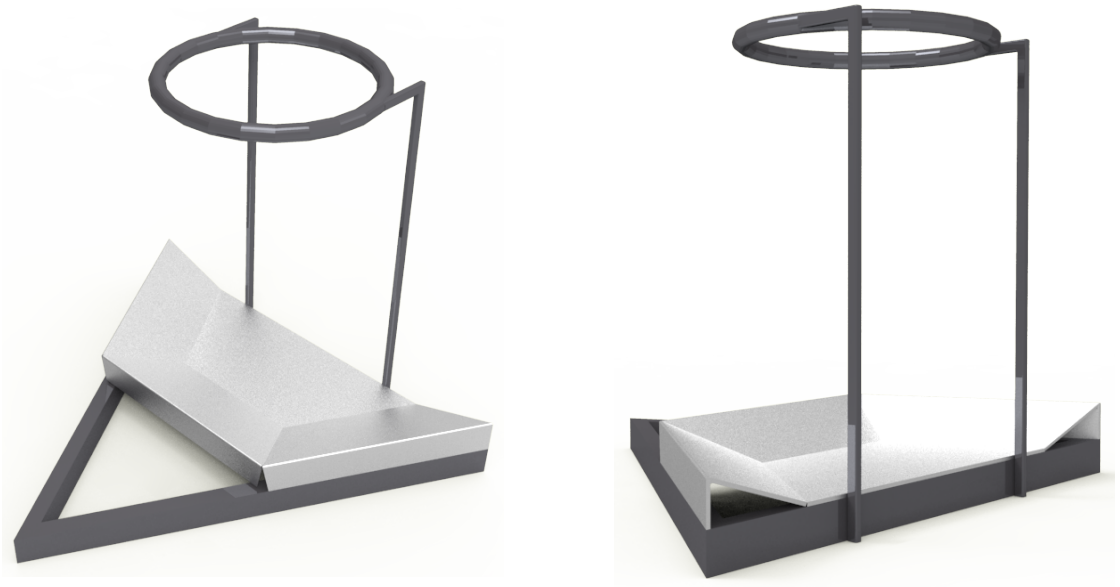


FIGURE 4.1.: Version Alpha du support

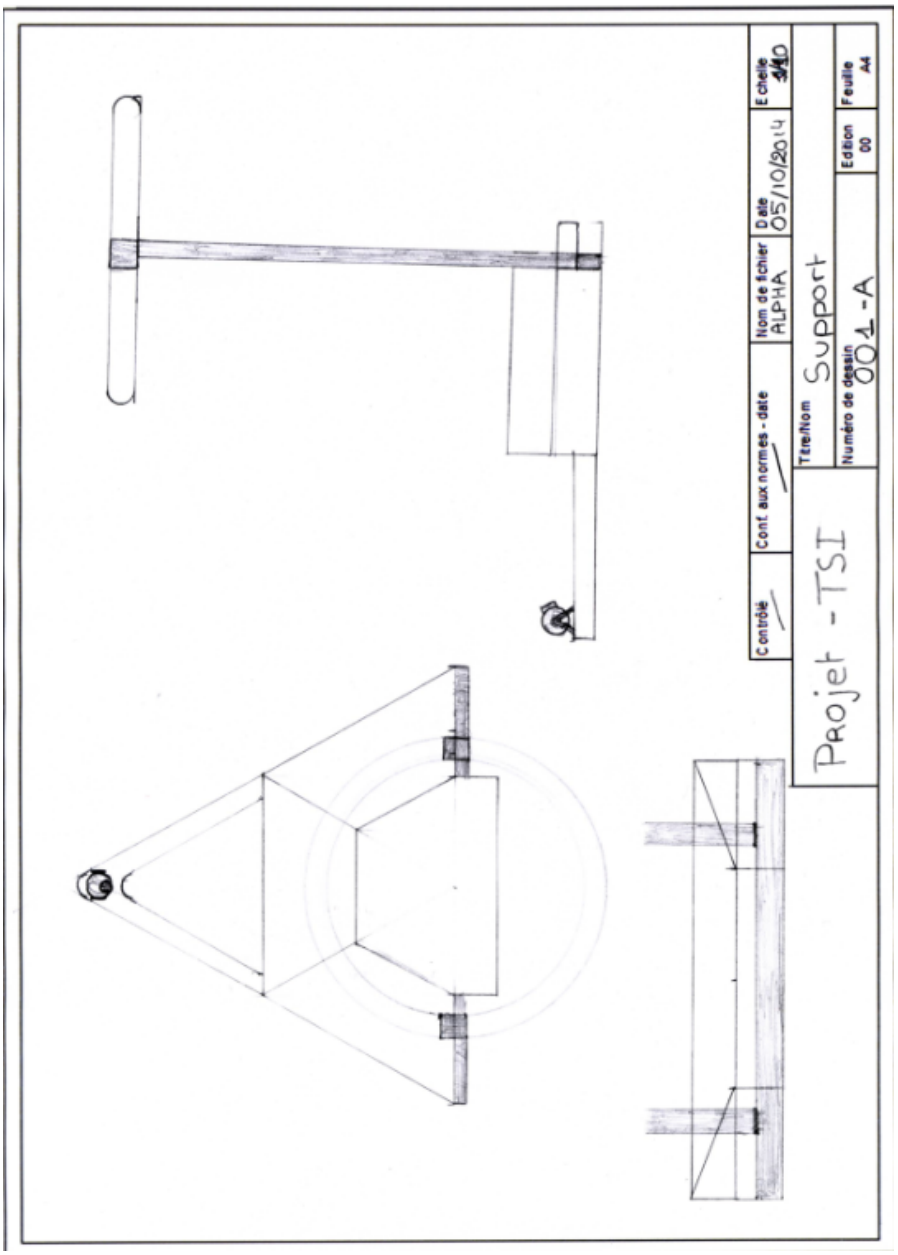


FIGURE 4.2.: Plans de la version Alpha du support

Avantages	**Inconvénients**
Assez jolie	Poids
Dimensions proches du support actuel	Impossibilité de se déplacer à 360°
Solidité (due aux IPN)	Solution peu efficace pour les glissements
	Coût (prix des IPN)
	Non démontable
	Cintrage du profilé tubulaire assez complexe

4.1.2. Version Bravo

Suite à la version Alpha, nous avons commencé à expérimenter des solutions plus improbables.

Cette version Bravo, était une « mise à jour » de la version Alpha. En effet, celle-ci garde le principe des plans inclinés, mais nous y avons ajouté des billes ou des rouleaux de manutention.

Ces billes (ou rouleaux de manutention) agissent comment des roulements, facilitant le glissement de l'utilisateur lors de son expérience de jeu.

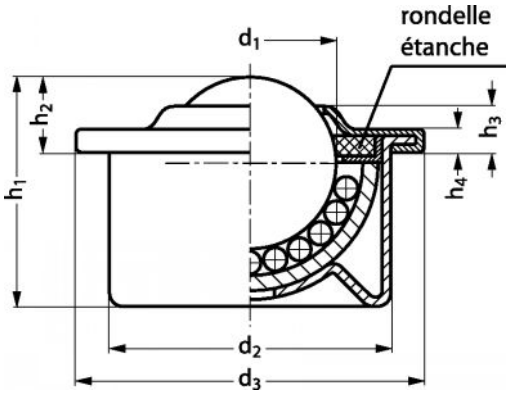


FIGURE 4.3.: Exemple de bille

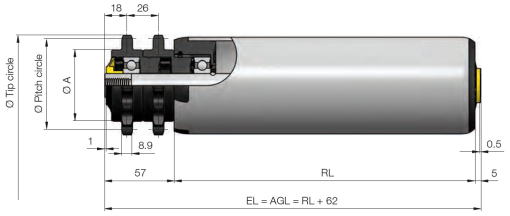


FIGURE 4.4.: Exemple de rouleau de manutention

Trois possibilités sont ainsi possibles avec des billes ou des rouleaux de manutentions :

1. La première est d'ajouter sur les plaques de métal des rouleaux de manutention. Ceux-ci peuvent prendre la forme de plaques où plusieurs rouleaux sont alignés ou bien la forme d'un tapis roulant, où deux rouleaux éloignés interagissent à l'aide d'une courroie.

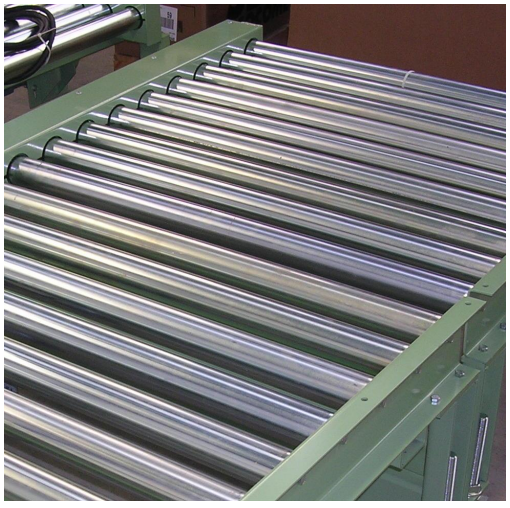


FIGURE 4.5.: Rouleaux mis côte à côte

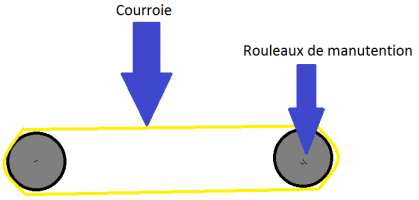


FIGURE 4.6.: Rouleaux reliés par une courroie

2. La seconde est de créer un sol en « bol » comme celui de *l'Omni* et de le remplir de billes de manutention. Cela aurait créé un sol glissant pouvant ainsi nous permettre de marcher de façon naturelle tout en restant sur place. Or pour cette version il fallait créer des billes de manutention spéciales afin que celles-ci soient seulement aimantées. Il aurait aussi fallu un nombre très important de billes pour remplir toute la surface du support.

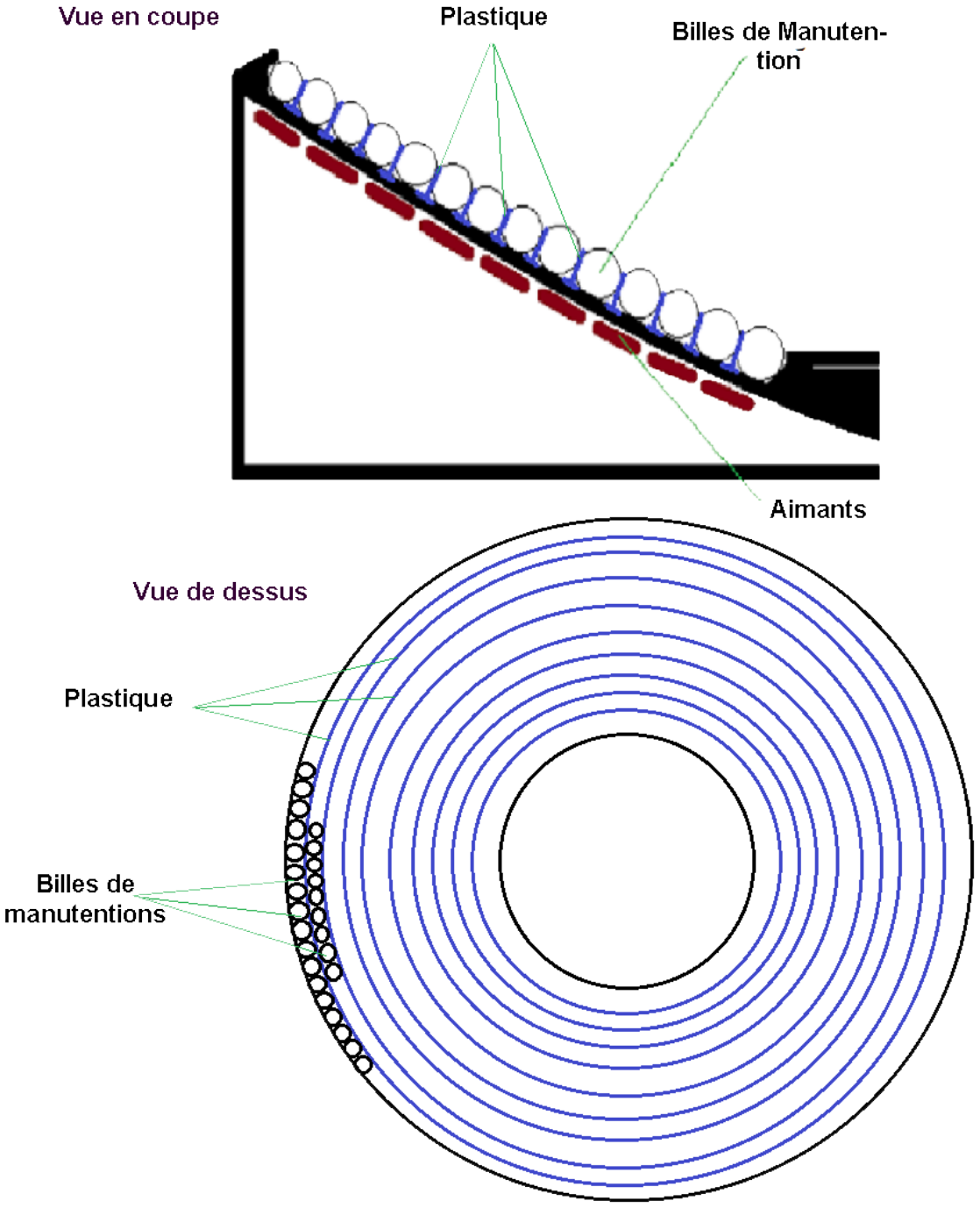


FIGURE 4.7.: Support rempli de billes de manutention

3. La dernière méthode utilisant des billes de manutention est une méthode qui ressemble à s'y méprendre à celle de *Virtuix*. Ici on garde le socle Alpha où l'on remplace les plaques de métal par de la tôle ondulée (censée recréer les rainures de *Virtuix*), et l'on crée une paire de chaussures comportant une bille de manutention sur le devant. Ces chaussures seront un peu comme des « Heelys » chaussures comportant une roulette sur le derrière, mais où la roulette serait remplacée par une bille de manutention et serait sur le devant.

FIGURE 4.8.: Tôle ondulée

FIGURE 4.9.: Chaussure « Heelys »

Avantages	**Inconvénients**
Esthétique	Coût (prix des billes et des rouleaux)
Glisse quasi-parfaite	Fabrication difficile
Déplacement à 360° (Deuxième méthode)	Faible efficacité de la dernière méthode
	Non démontable
	Poids

4.1.3. Version Charlie

Dans cette version on repart de zéro. Ici on s'inspire grandement des tapis de course que l'on utilise pour faire du sport. Il n'y a plus de plan incliné, le sol est plat et comporte quatre tapis de course.

Ainsi, le déplacement du joueur ne se fera pas par glissement mais grâce aux mouvements des tapis entraînés par des moteurs. Cela permet d'avoir une réelle sensation de marche, comme celle qu'on peut ressentir en marchant dans la rue par exemple.

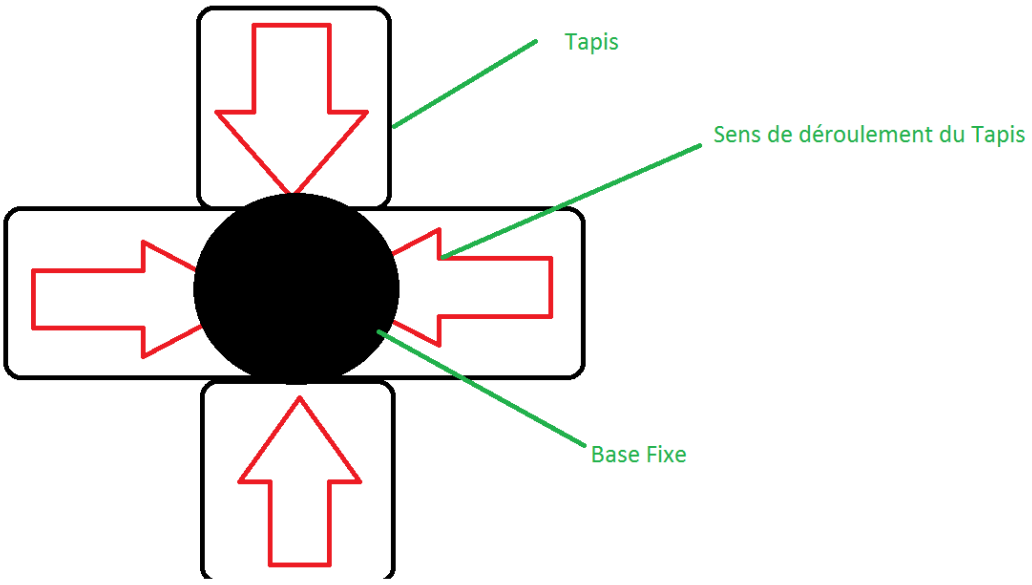


FIGURE 4.10.: Version Charlie du support avec des tapis roulants

Avantages	Inconvénients
Esthétique	Coût
Démarche naturelle	Fabrication difficile
Utilisation de l'électronique	Insécurité
	Non démontable
	Poids
	Diagonales non couvertes

L'inconvénient majeur de cette solution est l'impossibilité pour le joueur de marcher dans certaines directions, notamment dans les diagonales.

4.1.4. Version Delta

Il s'agit ici d'une version assez complexe et dont l'efficacité n'a pas été testée.

Cette version est munie d'un sol plat du type « Air Hockey ». N'avez vous donc jamais joué au Air Hockey dans une salle d'arcade ? Ce jeu où le palet glisse sur une table en métal muni de petits trous d'où sort de l'air comprimé ?

Nous aurions donc du créer un support muni d'un sol de ce type sur lequel le joueur vient se placer. Avec la pression générée par un compresseur, l'air sortant des trous est censée nous faire glisser sur une couche d'air comme un palet de Air Hockey.

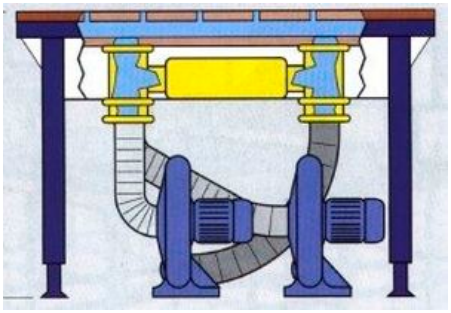


FIGURE 4.11.: Fonctionnement d'un compresseur

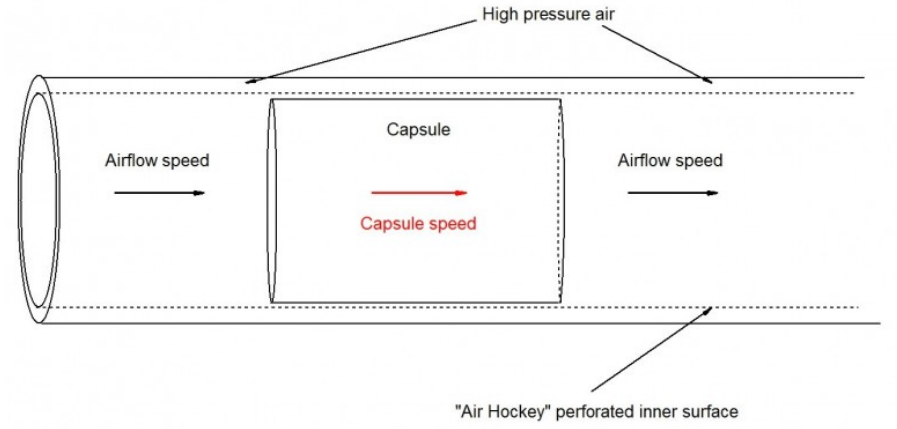


FIGURE 4.12.: Fonctionnement d'une table de « Air Hockey »

Avantages	**Inconvénients**
Glisse	Bruit
Apport d'une partie électricité	Coût
Déplacements à 360°	Puissance nécessaire
Stabilité	Poids
Sécurité	Non démontable / Réalisation complexe
	Résistance ?

4.1.5. Version Écho

La version Écho est la dernière version avant d'aboutir à la solution retenue. Cette version réalise un retour aux sources avec une amélioration de la version Alpha.

En lieu et place des plans inclinés, on retrouve un panneau en bois recouvert d'un produit qui provoque le glissement. Les dimensions ont été agrandies, afin d'avoir un meilleur confort d'utilisation.

Cette version s'utilise elle aussi en chaussettes. Elle n'est pas non plus démontable, par contre elle abandonne les poutres de types IPN pour des profilés en tubes carrés. Il s'agit d'une construction totalement soudée. La planche, elle, est un morceau de panneau en mélaminé avec plaquage en bois. Sont présents trois supports pour Webcam afin de reconnaître les mouvements de l'utilisateur (solution retenue aux débuts du projet mais abandonnée).

Cette version est assez ressemblante au Virtualizer. Voici un rendu et des dessins de définition de ce support :

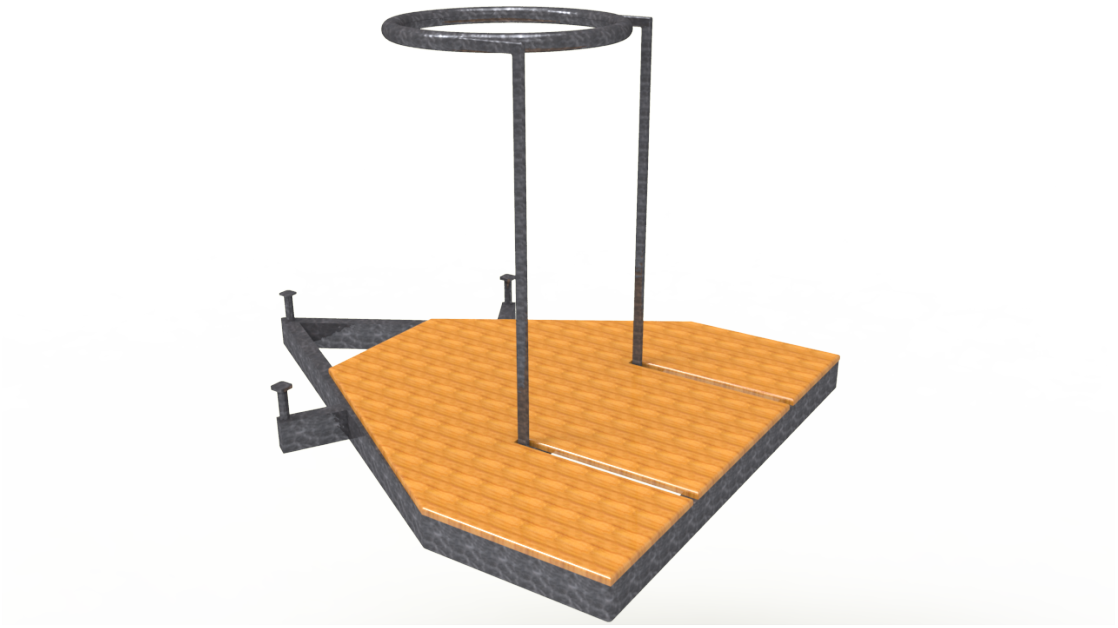


FIGURE 4.13.: Version Écho du support

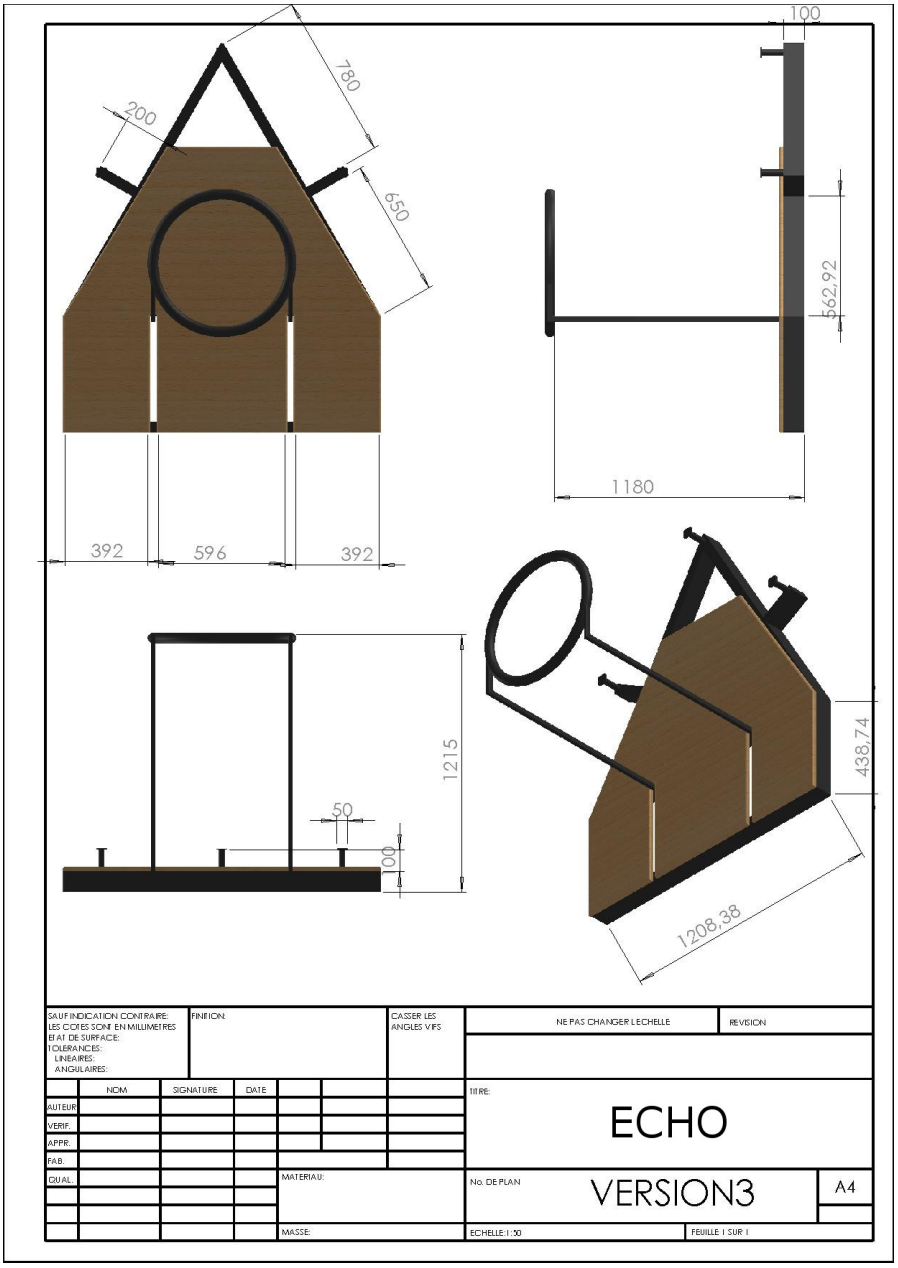


FIGURE 4.14.: Plans de la version Écho du support

Avantages	**Inconvénients**
Glisse	Non démontable
Stabilité	Pas de déplacement à 360°
Sécurité	
Esthétique	
Coût réduit par rapport à la version Alpha	

4.2. Solution retenue (Version Foxtrot)

Après avoir réfléchi sur différentes réalisations du socle, une version s'est enfin démarquée : la version Foxtrot.

Cette version est inspirée du *Virtualizer de Cyberith*, mais est assez éloignée de cette réalisation. En effet, celle-ci se présente sous la forme d'un socle démontable, composé d'une partie glissante en bois, d'un cadre et d'un arceau soutenu par deux montants tous deux en acier.

Cette version doit s'utiliser en chaussettes ou en patins d'appartements (pour avoir de meilleurs glissements).

Elle permet donc d'avoir la présence d'une glisse maîtrisable, de faire des déplacements à 360°, d'être assez stable pour une utilisation normale et d'être réalisable par nos soins.

Voici donc des rendus de cette version ainsi qu'un dessin de définition général utilisé lors de la réalisation :

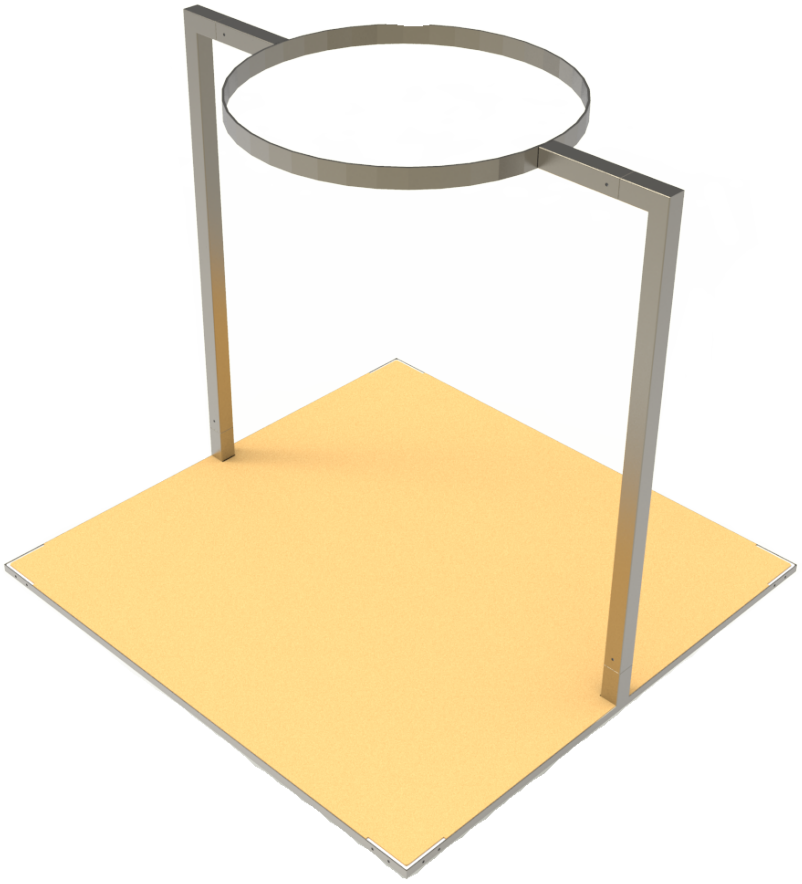


FIGURE 4.15.: Rendu de la version finale du support (vue de dessus)

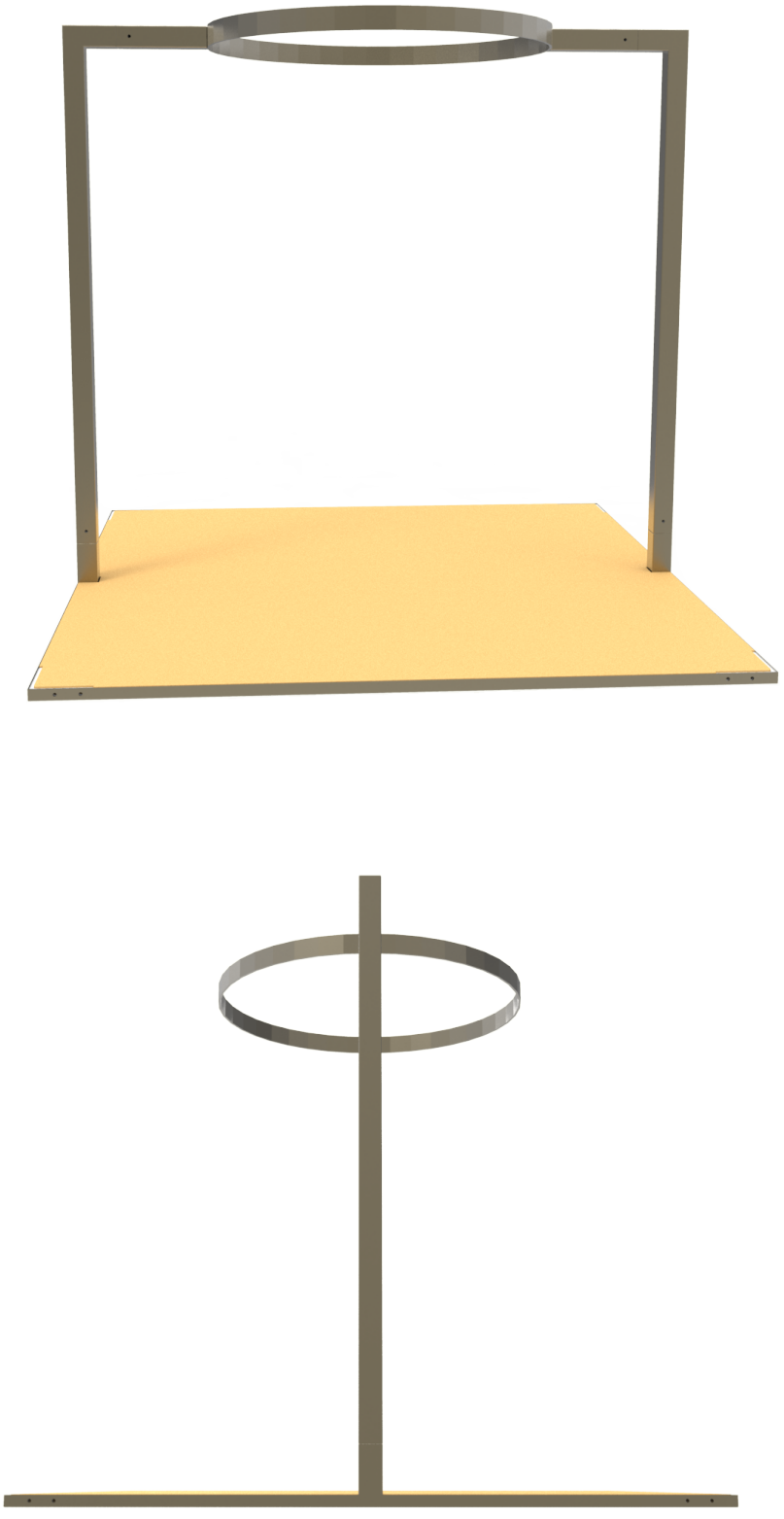


FIGURE 4.16.: Rendus de la version finale du support (vues de côtés)

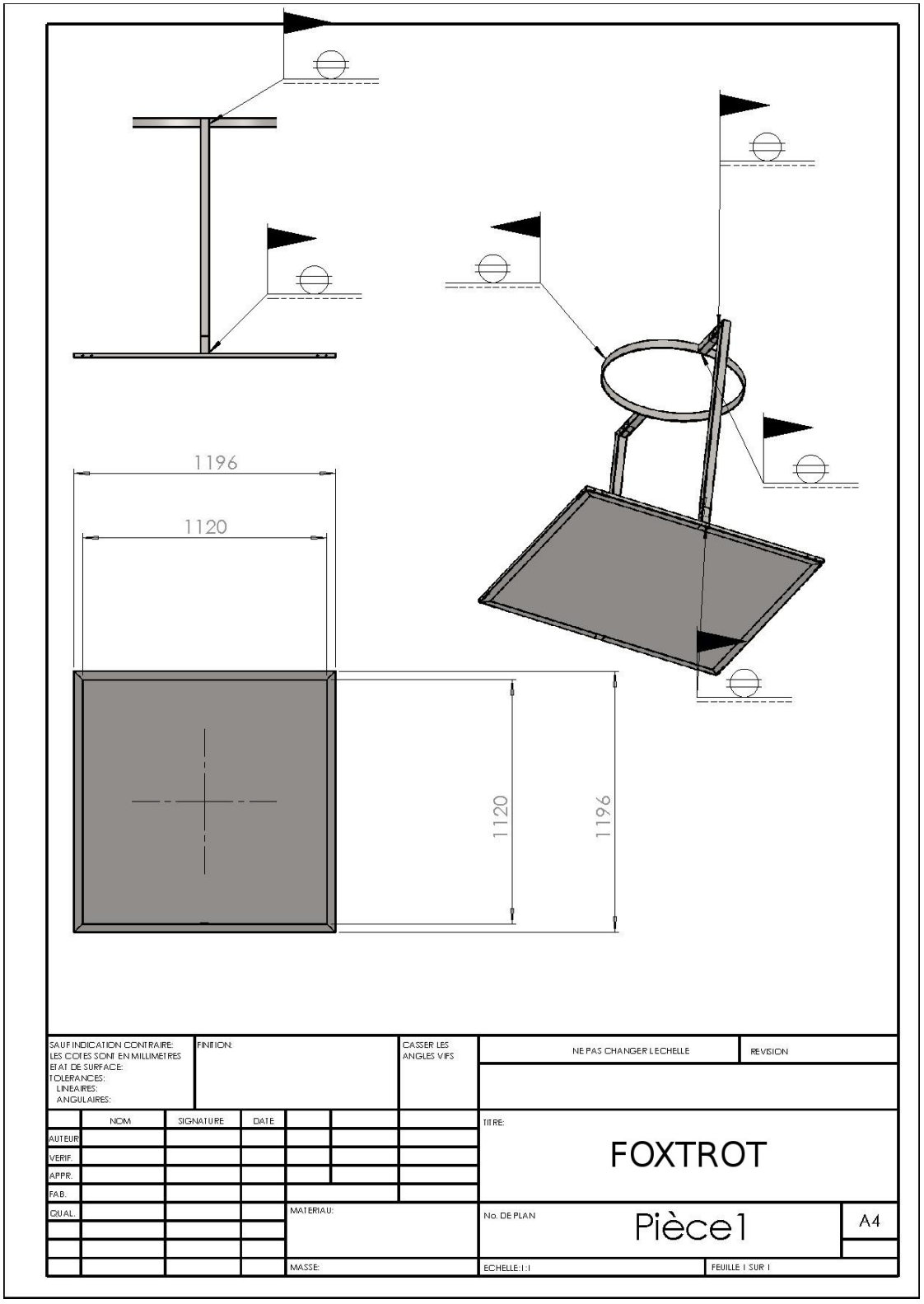


FIGURE 4.17.: Plan de la version finale du support

4.2.1. Matériaux utilisés

Deux matériaux différents sont utilisés :

— Pour les profilés, de l'acier brut laminé à froid (pour les plats et les tubes carrés) et à chaud (cornière). Ces profilés proposent donc une grande résistance, ainsi qu'une relative facilité de travail (par rapport à l'aluminium qui est difficile à souder par exemple). Leurs coûts assez bas facilitent aussi grandement l'étape de la réalisation. Seul le poids apparaît comme un problème ($8010\,\mathrm{kg\,m^{-3}}$), mais cela peut servir à la stabilité.

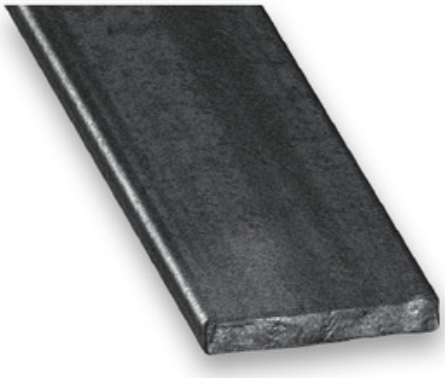


FIGURE 4.18.: Profilé laminé à chaud

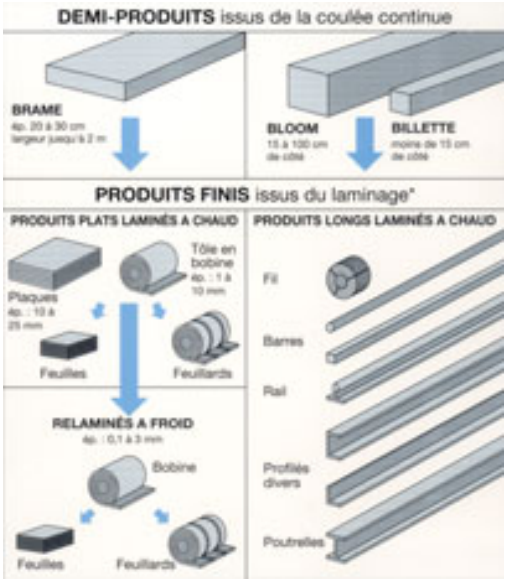


FIGURE 4.19.: Différents principes de laminage

— Pour les planches, il s'agit ici de panneaux en mélaminé plaqué bois vernis. Le mélaminé est obtenu par le plaquage à chaud d'une feuille imprégnée de mélamine sur un panneau de particules de bois qui est obtenu par agglomération des déchets de scierie et ajout d'une résine. L'avantage de ce type de panneau est qu'il est très répandu, l'un des moins cher, qu'il possède une bonne résistance et qu'il est aussi simple à travailler que le bois.

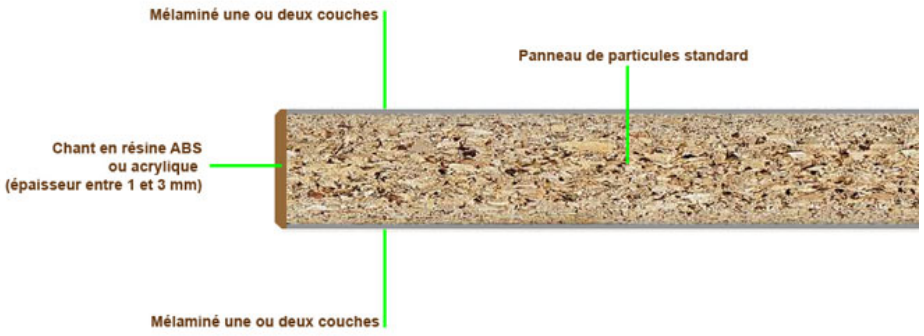


FIGURE 4.20.: Panneau en mélaminé plaqué bois

4.2.2. Étude qualitative

Avant de procéder à la réalisation de ce support nous avons du procéder à différentes études. La première qui a été faite est l'étude qualitative.

Mais qu'est ce qu'une étude qualitative ?

Une étude qualitative est une étude destinée à recueillir des éléments qualitatifs, qui sont le plus souvent non directement chiffrables par les individus interrogés ou étudiés. Cela permet d'avoir un aperçu de la conformité des solutions trouvées vis-à-vis des attentes exigées.

Voici donc cette étude qualitative :

	Fonction ou critère	Satisfaction	Importance
A	*Transportabilité*	*4*	*5*
B	*Démontable*	*4*	*5*
C	*Esthétique*	*2*	*2*
D	*Facilité d'utilisation*	*5*	*4*
E	*Glissant*	*4*	*5*
F	*Dimension*	*5*	*3*
G	*Stabilité*	*4*	*3*

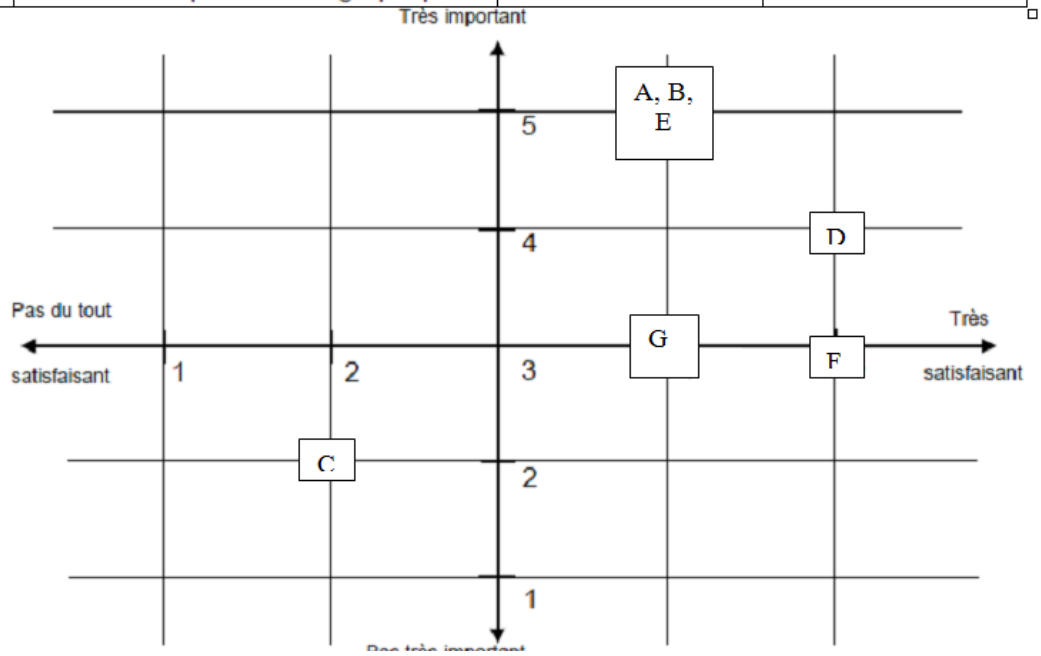


FIGURE 4.21.: Étude qualitative du support

4.2.3. Fiche technique

Après avoir réalisé une étude qualitative, nous avons réalisé une fiche technique générale de ce support. Cette fiche a pour but de concentrer en un seul document les détails principaux du support.

Voici donc cette fiche :

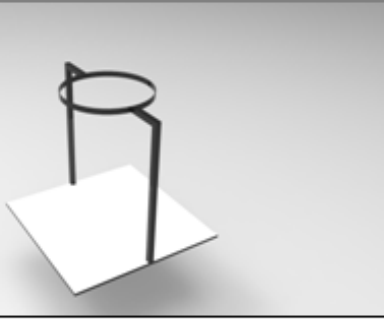
Fiche Technique du support	
Support Joueur	
Fonctions principales	Permettre de marcher de façon naturelle Pouvoir tourner à 360°
Fonctions secondaires	Etre accessible à tout le monde Etre solide Possède une surface glissante Etre stable Ne pas occulter la vision de la Kinect
Données Techniques	Lxlxh : 1200mm x 1200 mm x 1100mm Durée de Vie : > à 5 ans
Contraintes économiques	Aucune
Principaux Matériaux & Composants	Acier, Contreplaqué Stratifié, Boulons
Contraintes de Fabrication	Respect de délais Sécurité Drastique
Caractéristiques	Peint Transportable Entretien limité Démontable

FIGURE 4.22.: Fiche technique du support

4.2.4. Diagramme FAST

Nous avons aussi réalisé un diagramme FAST afin de rendre plus claires les solutions techniques retenues et leur utilité dans le système.

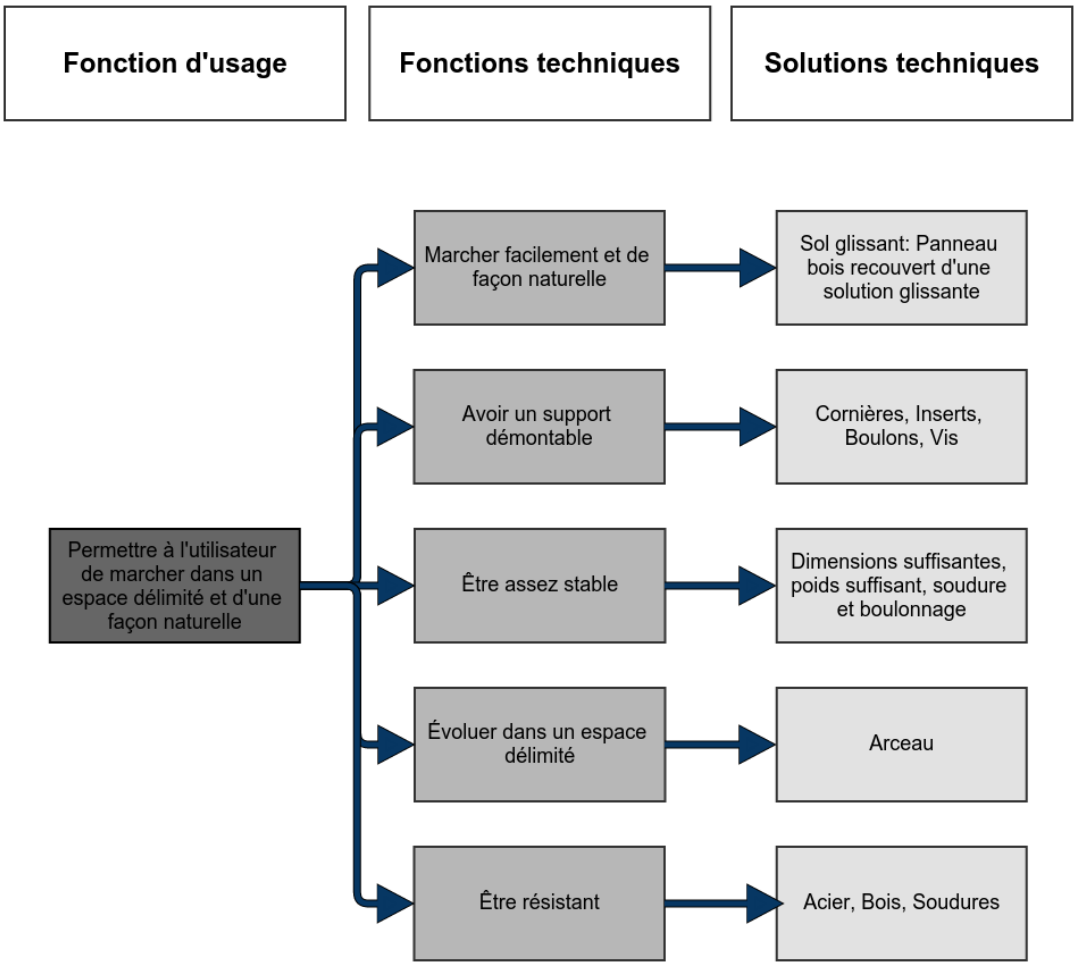


FIGURE 4.23.: Diagramme FAST du support

4.2.5. Simulation

Avant de réaliser le support, nous devions d'abord effectuer des simulations afin de voir si celui-ci pourrait résister à un usage intensif. Pour la simulation, il fallait créer une étude principalement sur la partie métallique de ce support.

Pour ce faire, nous avons utilisé *Autodesk Inventor Professionnal*, qui nous a permis de réaliser l'ensemble des études de ce support.

Ici, deux forces sont appliquées : Une sur l'arceau et une autre sur le cadre inférieur. Toutes deux ont une valeur de 800 N ce qui correspond au poids moyen d'un homme de 80 kg sur Terre.

Grâce à ces simulations, nous avons pu vérifié que le support résisterait à un usage intensif et supporterait bien les forces qui lui seront appliquées sans céder.

Voici les résultats obtenus :

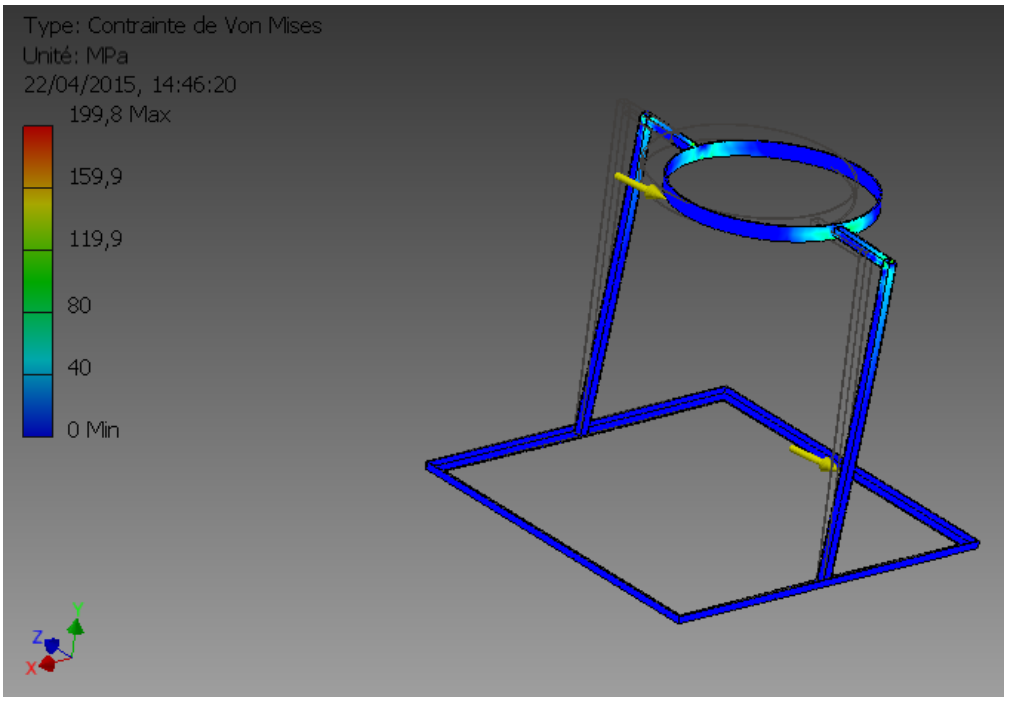


FIGURE 4.24.: Simulation du support : Contrainte de Von Mises

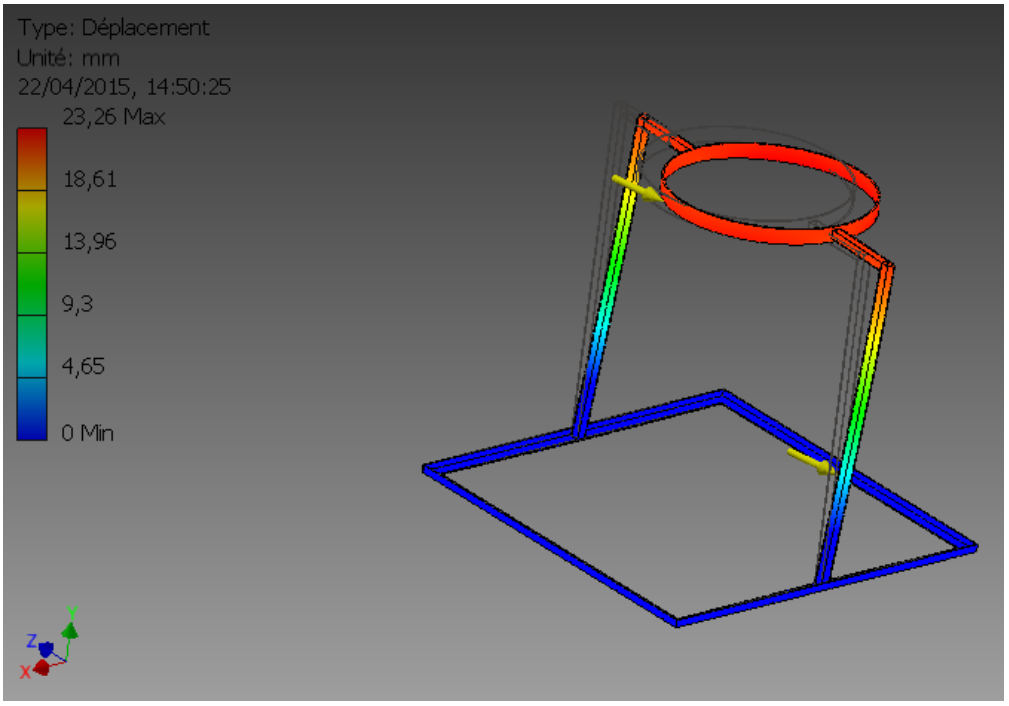


FIGURE 4.25.: Simulation du support : Déplacements

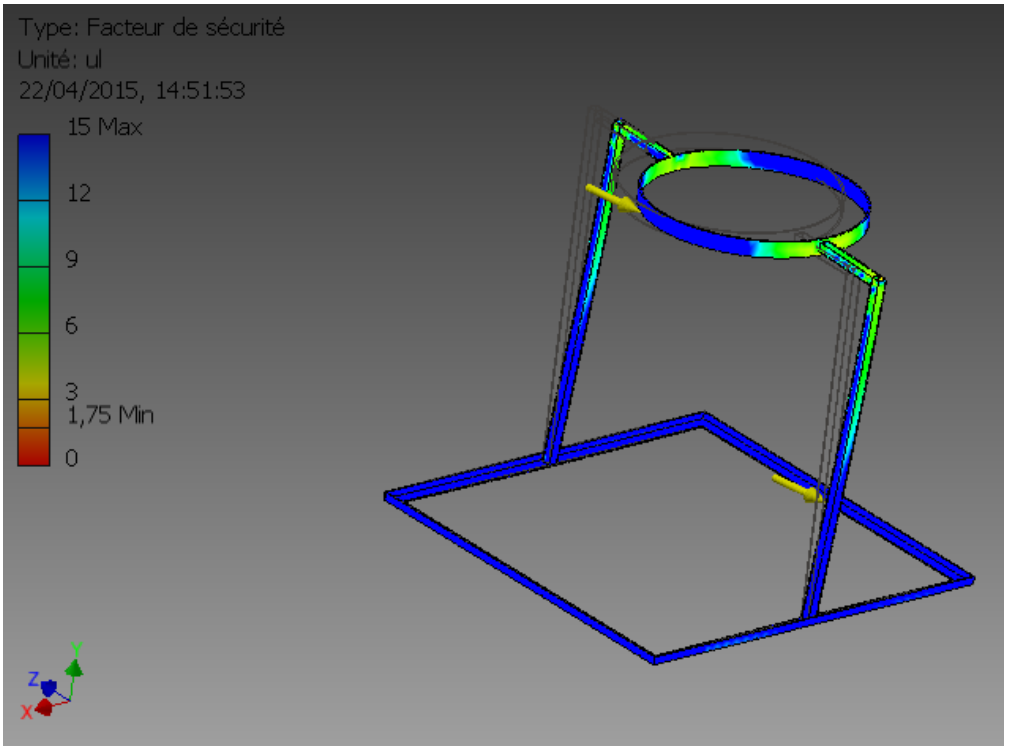


FIGURE 4.26.: Simulation du support : Facteur de sécurité

4.2.6. Réalisation

Cette partie revient pas à pas sur toutes les étapes nécessaires à la réalisation du support, certaines d'entre-elles nécessitant des procédés spécifiques. En voici la liste :

— Soudure au MIG
— Soudure à l'arc (réalisé par les chaudronniers du lycée)
— Taraudage
— Autres machines utilisées
— Montage final

4.2.6.1. Soudure au MIG

Ce type de soudure est celle qui a été le plus utilisée lors de la réalisation du support. Elle a servi à souder les montants, l'arceau et le support pour la planche de bois.

Il s'agit d'un procédé de soudure sous protection de gaz inerte. Il est très utilisé de nos jours. Ce procédé utilise la création d'un arc électrique entre la pièce à souder et le fil d'apport. Quand nous obtenons notre arc, on dévide (ou on déroule) ce fil d'apport, toujours à la même vitesse et en continu dans notre métal en fusion, qui est créé par l'arc. Ainsi, on obtient par la suite un cordon de soudure, constitué d'un mélange entre le métal de base et celui du fil d'apport.

Cette soudure est réalisée sous protection gazeuse, ici un gaz inerte (mélange de CO_2 et d'Argon pour les aciers), qui sert à protéger de l'air le métal en fusion afin d'éviter l'oxydation.

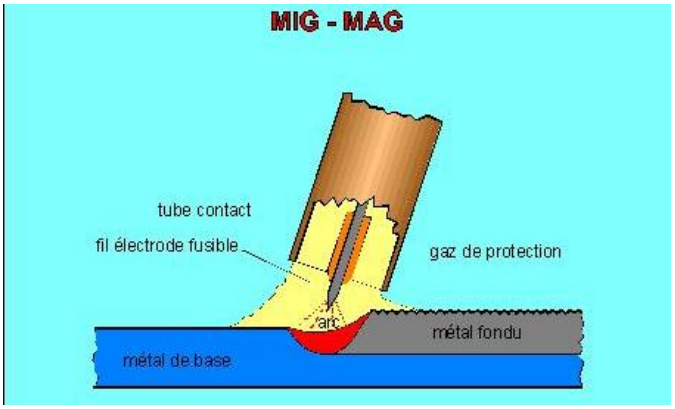


FIGURE 4.27.: Schéma d'explication de la soudure au MIG

Ce procédé de soudure fonctionne principalement avec une bonbonne de gaz, ainsi qu'un générateur qui agissent ensemble selon le schéma suivant :

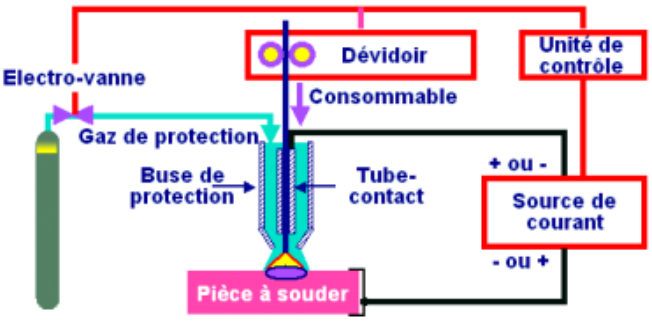


FIGURE 4.28.: Schéma d'explication de la soudure au MIG (2)

Il est possible de souder par points, ou en continu. Dans notre projet, ce sont des soudures en continu qui ont été réalisées.

FIGURE 4.29.: Soudure au MIG

Lorsque l'on réalise ce type de soudure, le courant de soudage utilisé est continu. On doit veiller a vérifier la distance entre le tube et la pièce pour obtenir une distance optimale de soudure. En effet si elle est trop grande, l'intensité du courant diminue quand la longueur de fil va augmenter, et si elle est trop petite il y a risque de soudure entre le fil d'apport et le tube de contact.

Lors de la réalisation du socle, nous avons réglé cette distance à 8 mm.

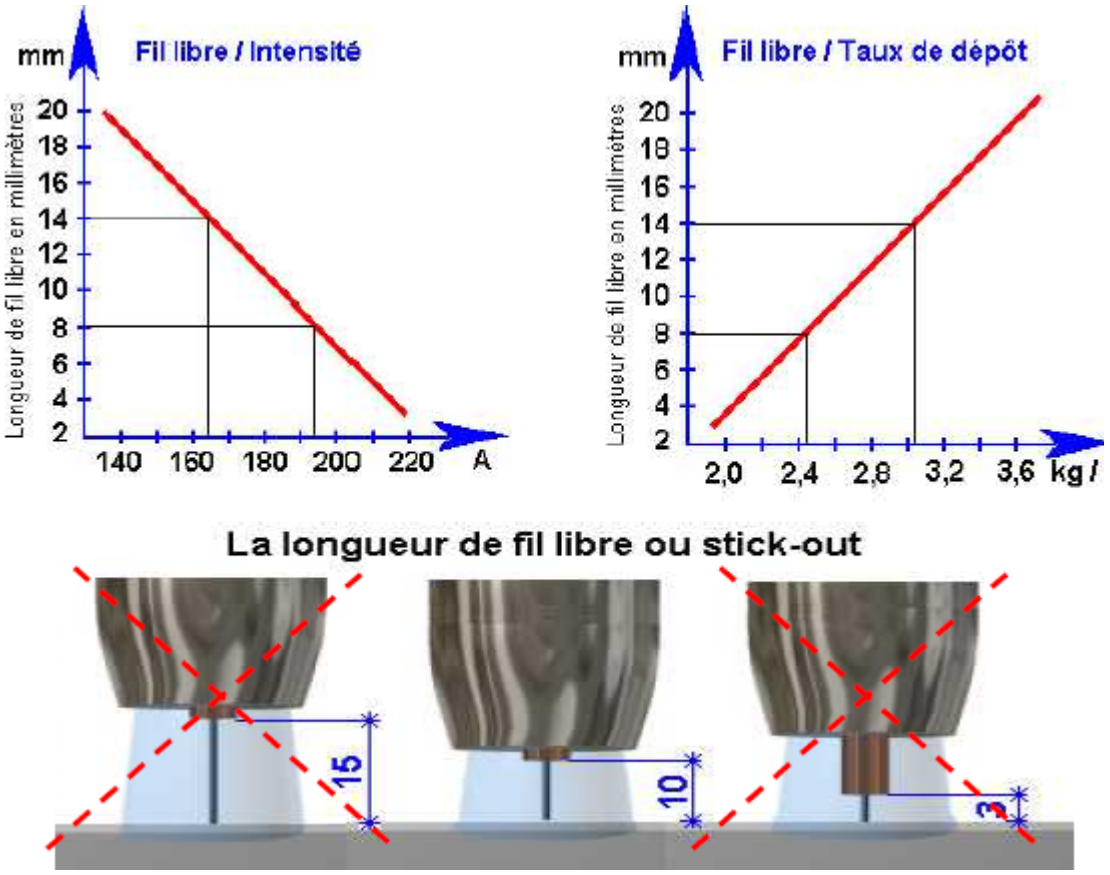


FIGURE 4.30.: Distances utilisées pour la soudure au MIG

4.2.6.2. Soudure à l'arc

Lors du montage au lycée, nous avons remarqué un fort jeu entre plusieurs pièces du support. Ainsi nous avons donc du nous résoudre à demander aux chaudronniers de réaliser quelques cordons de soudure afin de nous aider. Ces derniers ont utilisé la soudure à l'arc dont voici le principe :

Le soudage à l'arc consiste à réaliser un court-circuit qui provoque un arc électrique, qui lui-même va dégager beaucoup de chaleur. L'électrode, ou baguette fond et va permettre de réaliser la soudure. Ces baguettes sont constituées d'une « âme » de même nature que la pièce à souder ainsi que d'un enrobage servant de décapant.

1- âme de l'électrode
2- enrobage de l'électrode
3- extrémité en fusion de l'âme
4- arc
5- goutte de métal de laitier passant dans l'arc.

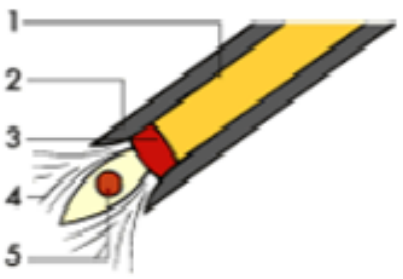


FIGURE 4.31.: Explicatif de la soudure à l'arc

Lors de la réalisation d'une soudure à l'arc, il se forme du « laitier » par la fusion et l'oxydation de l'arrosage de l'électrode. Voici une image expliquant ce phénomène :

1- laitier solidifié
2- laitier en fusion
3- arc
4- âme
5- enrobage
6- métal de base
7- bain de fusion (cratère)
8- métal solidifié (métal déposé + métal pièce)

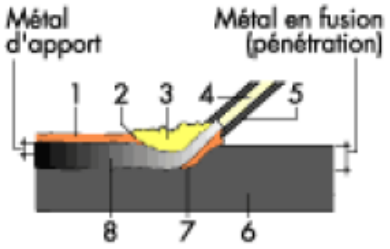


FIGURE 4.32.: Présence de « laitier »suite à une soudure à l'arc

FIGURE 4.33.: Soudure à l'arc du support

4.2.6.3. Taraudage

Afin que le support utilisé dans notre projet soit démontable, nous avons du utiliser un système de boulonnage. Si certains boulons se serrent dans un écrou, d'autres ont eu besoin d'un filetage.

C'est ainsi que nous avons du tarauder les équerres qui maintiennent le cadre inférieur du socle.

Mais qu'est-ce que le taraudage ?

Les taraudages constituent des surfaces hélicoïdales intérieures. Leur rôle est de permettre des assemblages qui sont démontables à l'aide de vis ou encore de boulons.

Dans un assemblage, une seule pièce est taraudée afin que la vis ou le boulon rentre librement dans la pièce avec le trou lisse puis se visse dans celle qui est taraudée. En effet il est impossible de serrer l'une contre l'autre deux pièces taraudées. On appelle ce trou lisse trou de passage.

Pour éviter d'endommager le filetage de la vis ou du boulon au moment du montage, le diamètre du trou de passage sera supérieur au diamètre de la vis ou du boulon (environ 2 mm de plus).

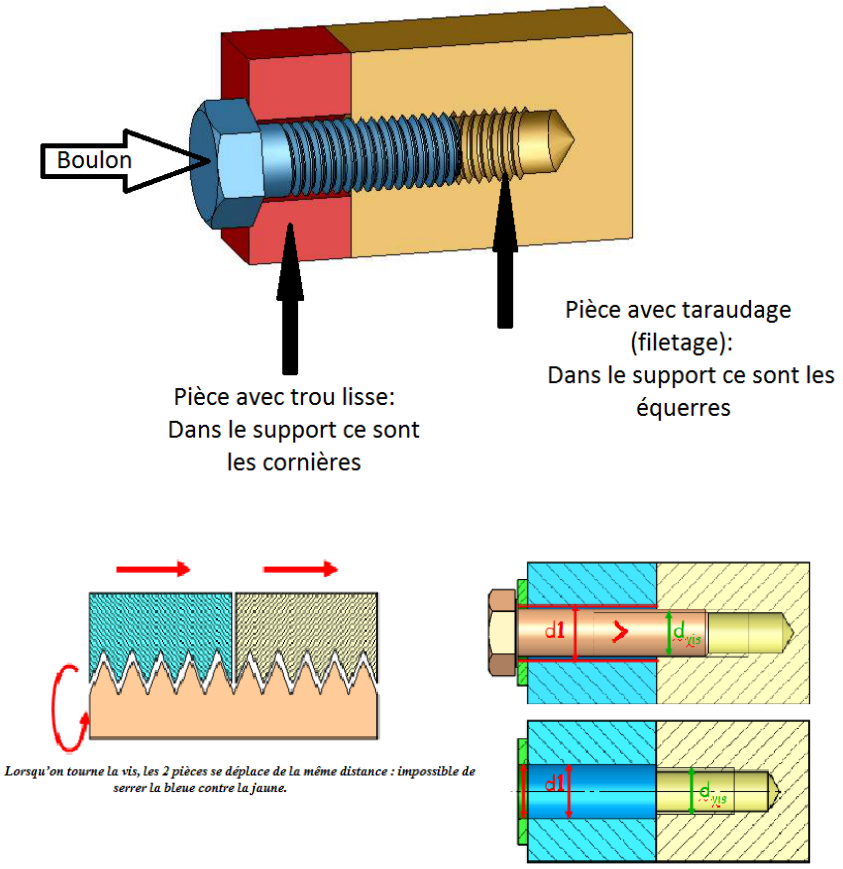


FIGURE 4.34.: Explications du taraudage

FIGURE 4.35.: Taraudage du support

Avant de tarauder une pièce, il faut toujours la percer à l'aide d'un foret. Par la suite, ce n'est pas moins de trois tarauds qui sont nécessaires pour tarauder. Pour vous éclairer, voici deux images explicatives :

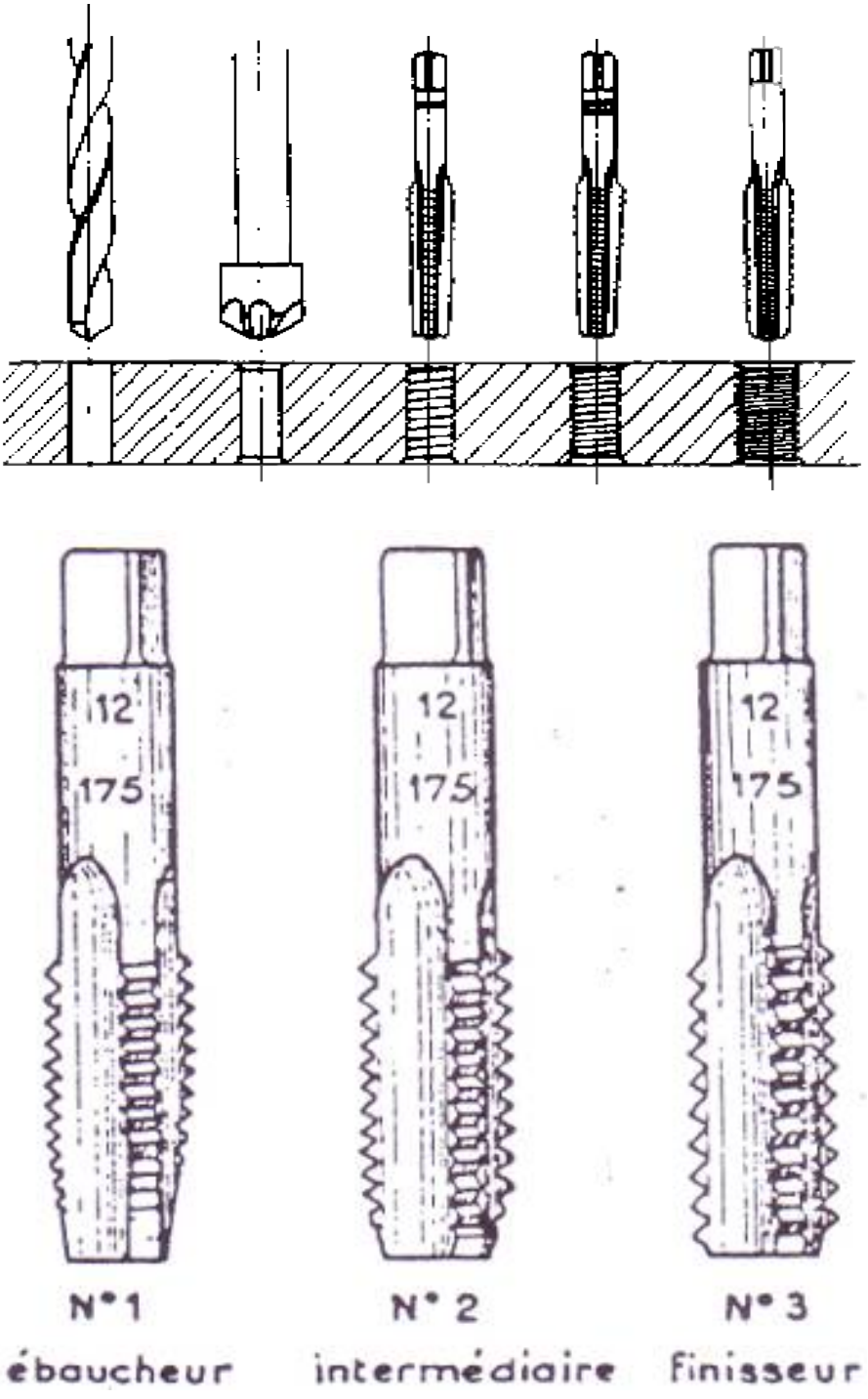


FIGURE 4.36.: Étapes du taraudage

4.2.6.4. Autres machines utilisées

Mise à part les postes à souder, d'autres machines ont été utilisées afin de mener à bien la réalisation de ce support. Nous allons vous en présenter les principales :

— **La scie à ruban horizontale à eau :** Cette machine à permis de découper les profilés de façon simple et rapide. Elle permet d'avoir une grande précision de coupe. La descente du ruban est hydraulique et réglable. Cette machine permet de faire des coupes sous différents angles car l'archet (support du ruban) est inclinable. Le système d'arrosage d'eau intégré permet de couper le métal sans étincelles ni projections.

FIGURE 4.37.: Scie à ruban horizontale à eau

— **La dégauchisseuse :** Cette machine outil de menuiserie nous a permis de dégauchir les angles des planches afin que les équerres puissent s'insérer.

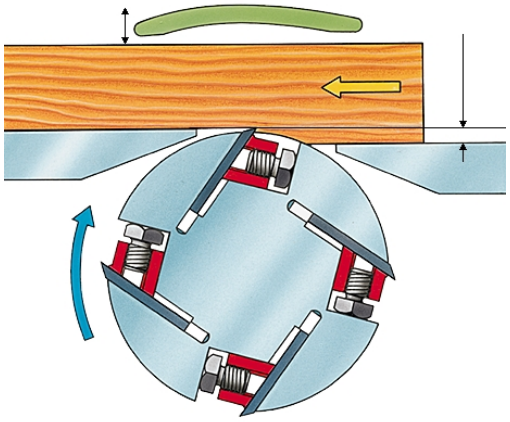


FIGURE 4.38.: Schéma d'une Dégauchisseuse

FIGURE 4.39.: Dégauchisseuse

— **La scie à débit à commande numérique :** Cette machine est une scie circulaire montée sur un axe. Le déplacement parfaitement linéaire de cette scie permet des coupes parfaitement droites, et sa commande numérique, des précisions de coupes au dixième de millimètre.

FIGURE 4.40.: Scie à débit à commande numérique

— **Une perceuse à colonne :** Elle nous a permis de réaliser tout les perçages du support. Elle permet une meilleure précision de perçage qu'une perceuse sans colonne, ainsi qu'un parfait alignement de deux perçages.

FIGURE 4.41.: Perceuse à colonne

4.2.6.5. Montage final

Après avoir réalisé tout les éléments composant le support du projet Virtual Walker, il a fallu passer à l'étape du montage. Cette étape s'est avérée déterminante pour cette partie du projet, car c'est à cet instant seulement que l'on a pu constater les éventuels défauts de ce support.

Fort heureusement, aucun problème majeur n'est apparu et le montage a fonctionné comme nous l'avions souhaité. Voici donc deux photos, la premère correspond au 1er montage réalisé en atelier, et la seconde montre le socle en action aux Olympiades de SI (le socle a été peint en noir).

FIGURE 4.42.: 1er montage du support

FIGURE 4.43.: Montage final du support

4.2.7. Recherche d'un agent glissant

Pour que l'utilisateur ait une démarche naturelle tout en restant dans un espace délimité, nous devions diminuer les frottements au maximum. Dans notre cas il s'agissait de frottements cinétiques. Voici donc un court rappel sur les frottements cinétiques :

Lorsqu'un objet glisse sur une surface, la force de frottement est appelée frottement cinétique. Ce frottement tend à ralentir l'objet. L'observation expérimentale montre que l'intensité du frottement cinétique varie en fonction du poids de l'objet et du coefficient de frottement cinétique, mais pas de l'aire de contact ni de la vitesse. Le coefficient de frottement cinétique varie selon le type de matériaux en contact.

Pour diminuer ces frottements, nous pouvions seulement agir sur le coefficient de frottement cinétique en le diminuant. Ainsi plusieurs solutions se sont offertes à nous : soit modifier le matériau, soit lubrifier la surface.

Voici un tableau représentatif de différents coefficients de frottements :

Matériaux	Coefficient statique (μ_s)	Coefficient cinétique (μ_k)
Velcro – velcro	6.0	5.9
Aluminium – aluminium	1.4	1.2
Verre – verre	1.0	0.4
Caoutchouc – béton	0.9	0.68
Caoutchouc – asphalte	0.85	0.67
Acier – acier	0.75	0.57
Cuir – bois	0.61	0.52
Bois – bois	0.58	0.4
Cuivre – Acier	0.53	0.36
Ski – neige	0.14	0.05
Acier – glace	0.1	0.05

FIGURE 4.44.: Coefficients de frottements

4.2.7.1. Solutions abandonnées

La première idée que nous avons eu est celle d'ajouter une couche de résine époxy sur le mélaminé. Cette résine aurait permis, une fois sèche et uniformément répartie, de réduire le coefficient de frottement du support. Or nous ne possédions pas les compétences nécessaires pour mettre une résine de ce type, et surtout le coût élevé de cette résine nous a fait abandonner cette idée. Voici une photo d'un sol recouvert de ce type de résine, où l'on devine que la surface est glissante :

FIGURE 4.45.: Sol recouvert de résine époxy

La deuxième idée que nous avions eu consistait à coller un film PTFE ou teflon sur la planche. Comme sur certaines casseroles, ce produit aurait permis d'éviter un maximum les frottements. Cette solution est surement celle qui aurait fonctionné le mieux, mais malheureusement un coût très élevé nous a vite fait abandonner cette idée. Voici quelques images pour vous donner une idée du potentiel de ce matériau :

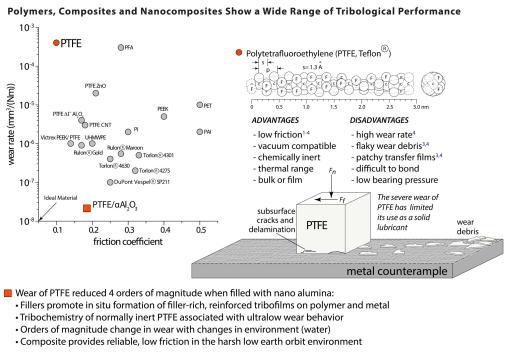


FIGURE 4.46.: Description du PTFE

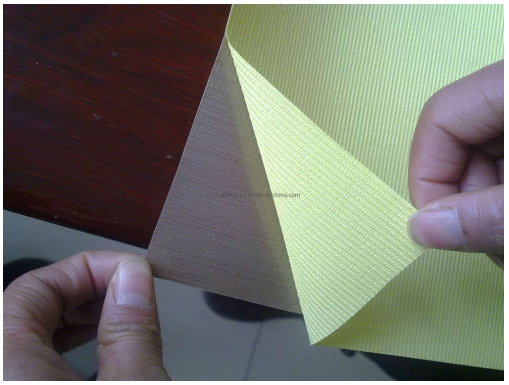


FIGURE 4.47.: Film PTFE

4.2.7.2. Solutions retenues

Pour notre projet nous avons décidé de retenir deux agents :

— Une solution d'essence de térébenthine et de paraffine très glissante sur le centre
— De la cire pour meubles, moins glissante sur les bords

Nous avons donc du réaliser une solution d'essence de térébenthine et de paraffine. Cette solution est très utilisée en menuiserie pour faire glisser des éléments tels que des tiroirs, ou bien sur les machines outils pour favoriser la maniabilité des planches. Cette solution lubrifie la planche, et va donc diminuer le coefficient de frottement. Voici des images de ces deux matières premières :

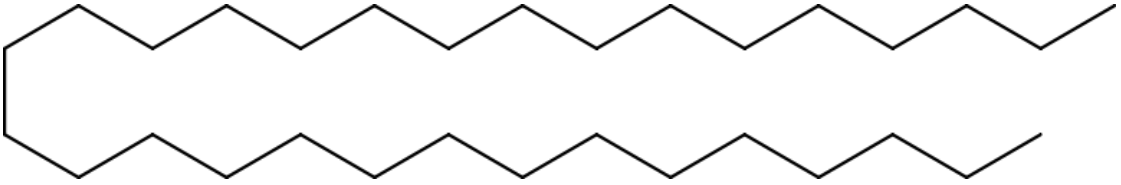


FIGURE 4.48.: Molécule de paraffine

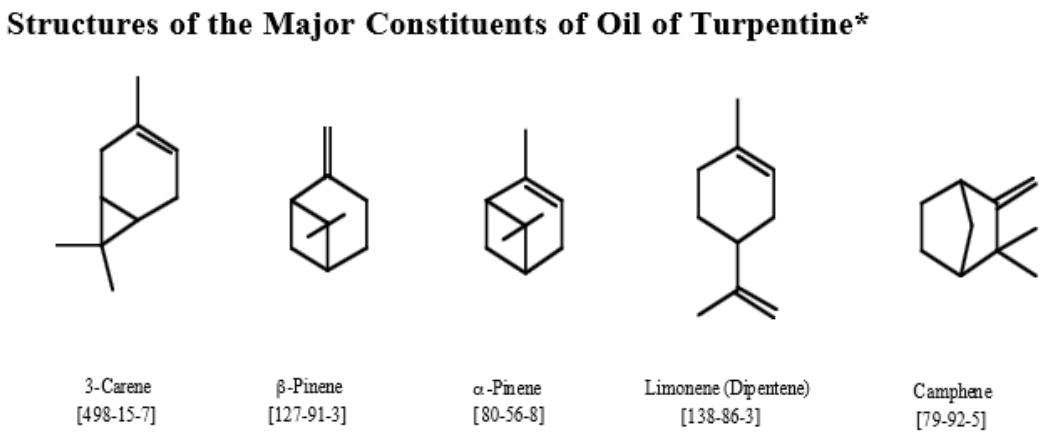


FIGURE 4.49.: Molécule de térébenthine

Pour accompagner cette solution nous avons acheté dans le commerce de la cire pour meubles en bombe. Celle-ci va lubrifier le bois mais va être moins glissante que la solution précédente. Les avantages de la cire pour meubles et de la solution préparée sont principalement le coût et la facilité d'utilisation.

Troisième partie .

Casque de Réalité Virtuelle

5. Objectifs du casque et solutions existantes

5.1. Objectifs

De nos jours, les téléphones portables sont de plus en plus puissants, et certains dépassent même la puissance de calcul des ordinateurs traditionnels. De plus, ceux-ci intègrent une multitude de capteurs (comme des accéléromètres ou des gyroscopes). Partant de cette constatation, nous nous sommes dit qu'il serait judicieux d'utiliser un smartphone pour gérer la partie « immersion visuelle ».

Cependant, pour que cela puisse se faire, il nous fallait trouver un moyen de maintenir l'écran du téléphone au niveau des yeux de l'utilisateur et c'est ainsi que nous avons imaginé un casque de réalité virtuelle.

Ce casque permettrait donc d'immerger le joueur, en maintenant la réalité virtuelle devant ses yeux. Pour ce faire, le casque doit être capable de contenir un téléphone et de s'attacher à la tête de l'utilisateur. Encore une fois, le facteur sécurité est important puisque le téléphone ne doit pas tomber du casque. De plus ce casque doit être confortable, surtout en cas d'utilisation prolongée.

Pour résumer les critères importants, voici le diagramme « Bête à cornes » ainsi que le diagramme SADT du casque :

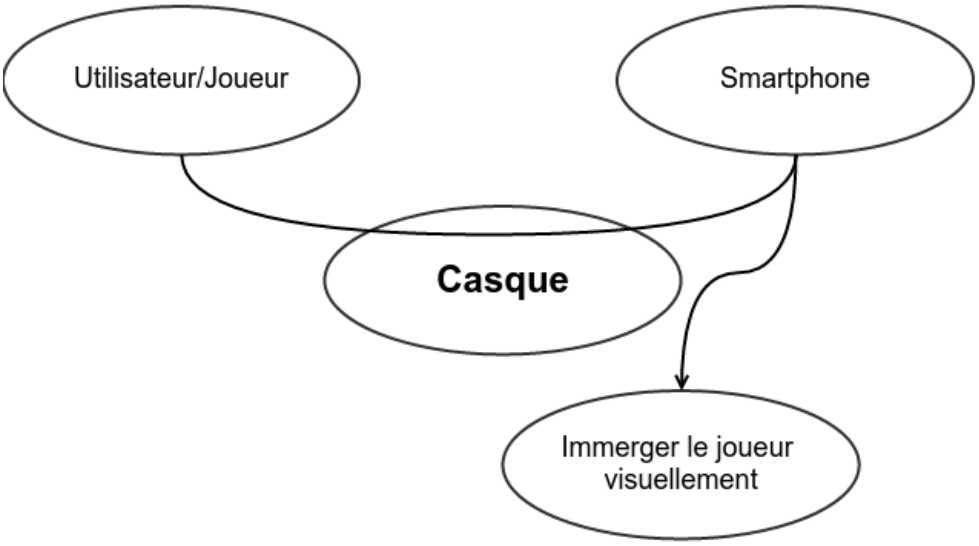


FIGURE 5.1.: Diagramme « Bête à cornes » du casque

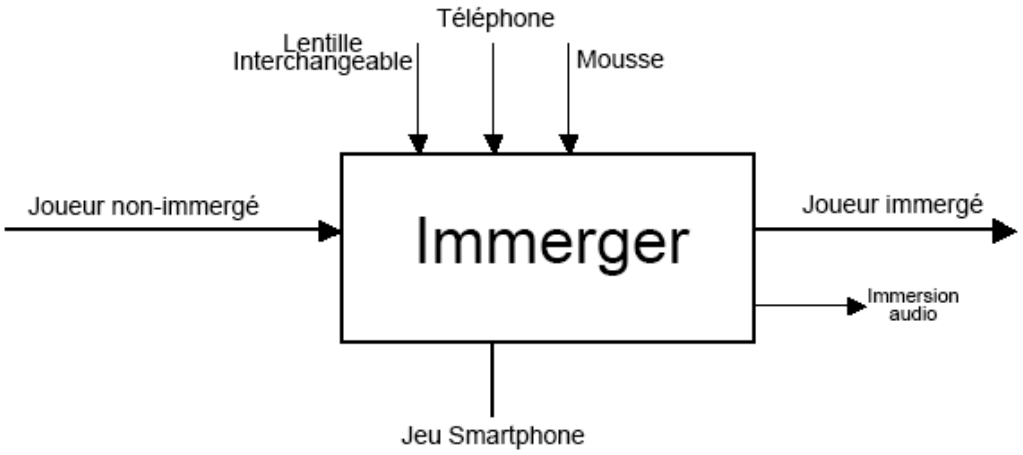


FIGURE 5.2.: Diagramme « SADT » du casque

5.2. Solutions Existantes

Le monde de la réalité virtuelle est en pleine expansion et de nombreux constructeurs commencent à développer leur propre produit. Des firmes telles que *Samsung*, *HTC*, *Occulus*, *Razer*, ... proposent leur solution dédiée à la réalité virtuelle. La quasi totalité de ces constructeurs propose soit un masque incluant un ou deux écrans, qui ne peuvent donc pas être changés, soit la possibilité de mettre un smartphone qui servira d'écran à l'intérieur du casque. Mais bien trop souvent, les marques limitent l'utilisation du casque aux téléphones de la même marque, souvent chers et parfois mal adaptés.

A l'heure actuelle, il existe de nombreux produits sur le marché, mais nous avons choisi d'en sélectionner 3, chacun ayant une caractéristique qui lui est propre :

— Occulus Rift
— Google Cardboard
— Homido

5.2.1. Occulus Rift

L'Occulus Rift est l'un des tout premiers produits parus sur le marché. Il se présente sous la forme d'un masque recouvrant les yeux. Il est tenu au visage par une sangle fermée à l'arrière du crâne. Un écran plat numérique est placé à quelques centimètres en face des yeux, perpendiculairement à l'axe du regard. Divers capteurs permettent de détecter les mouvements de tête de l'utilisateur, ce qui permet d'adapter en temps réel l'image projetée sur l'écran, afin de produire l'illusion d'une immersion dans la scène restituée. Ce masque est relié à un boîtier extérieur relié à un ordinateur. Il y a donc une connexion filaire qui entrave son utilisation.

FIGURE 5.3.: Occulus Rift

5.2.2. Google Cardboard

Le Google Cardboard est un masque de réalité virtuelle fonctionnant à l'aide d'un smartphone compatible (n'importe quel smartphone sous Android). Il fut développé par Google et présenté pour la première fois au grand public en 2014. Le masque permet de visualiser des images de réalité virtuelle générées par des applications mobiles. L'appareil se présente sous la forme d'un masque recouvrant les yeux à l'intérieur duquel un téléphone fait office d'écran. Il ne dispose pas de sangle d'attache et doit être maintenu au niveau des yeux par son utilisateur durant son usage.

Google diffuse librement les patrons en carton ainsi qu'une notice de création du Google Cardboard sur son site internet. Il est ainsi possible pour tous de construire son propre Cardboard. Cette initiative avait pour but de montrer qu'il n'y a pas besoin de depenser des fortunes pour profiter de la réalité virtuelle.

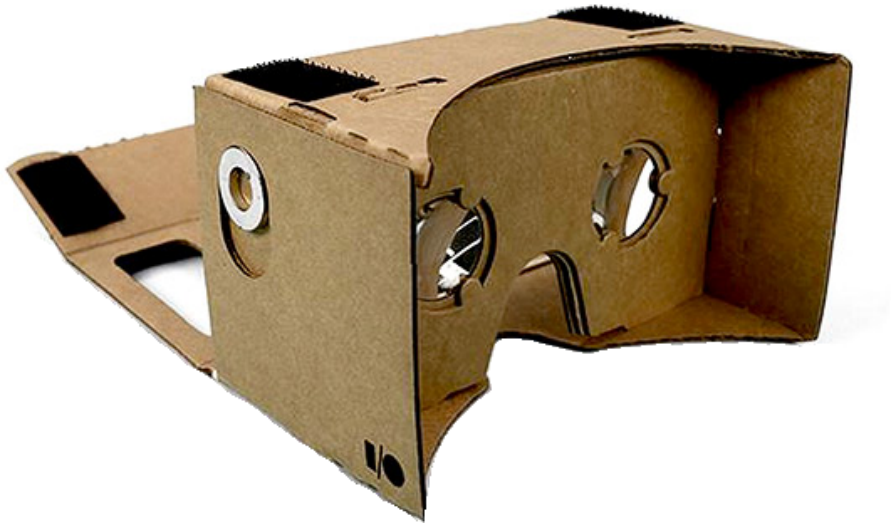


FIGURE 5.4.: Google Cardboard

FIGURE 5.5.: Mise en place du téléphone dans le Cardboard

5.2.3. Homido

L' Homido est un casque de réalité virtuelle fonctionnant à l'aide d'un smartphone compatible, semblable au Google Cardboard. Il se présente sous la forme d'un masque recouvrant les yeux à l'intérieur duquel un téléphone fait office d'écran. Il dispose d'une sangle permettant de maintenir le masque sur le visage de l'utilisateur. Le masque permet de régler la distance œil-lentilles ; ainsi que la possibilité d'inter-changer les jeux de lentilles pour certaines maladies telles que la myopie et l'hypermétropie.

Sa particularité est que le téléphone n'est pas totalement recouvert par le casque et qu'il est situé à l'extérieur.

FIGURE 5.6.: Homido

6. Conception et réalisation du casque

6.1. Cahier des charges

Au vu des solutions utilisées par les grandes marques (qui utilisent des casques se portant comme des masques de ski et épousant parfaitement les formes du visage), notre choix s'est donc naturellement porté vers la conception d'un dispositif de réalité virtuelle sous forme d'un masque pouvant accueillir un téléphone.

Grâce à l'observation des masques existants, de leurs points forts et de leurs défauts, nous avons décidé de modéliser notre propre masque pour répondre à toutes les attentes de notre projet.

Les points forts que nous avons décidé de conserver :

— Maintien du masque par trois sangles réglables
— Possibilité de changer les jeux de lentilles
— Possibilité de fonctionner avec plusieurs modèles de smartphone
— Possibilité de brancher un dispositif audio sur le smartphone

Nous avons également relevé les points faibles suivants, que nous avons décidé de ne pas reproduire :

— Être constamment relié à un PC par une connexion filaire
— Être relativement lourd
— Absence de mousses

Grâce à ces différents points, nous pouvons établir un cahier des charges des caractéristiques qui répondent au mieux à notre utilisation du masque pour notre projet.

Casque de réalité virtuelle	
Fonctions principales	- Pouvoir contenir un Nexus 5 à hauteur des yeux pour servir d'écran - Être porté sur le visage sans gêne - Ne doit pas bouger lors des mouvements - Doit pouvoir accueillir des lentilles
Fonctions secondaires	- Contenir des haut-parleurs pour restituer un environnement audio - Doit contenir un système de lentilles interchangeables - Doit s'adapter à différentes morphologies du visage
Contraintes économique	- Pouvoir être réalisé entièrement avec une imprimante 3D - Faire des économies de matières
Principaux matériaux & composants	- ABS - Bandes élastiques - Puce NFC - Mousse - Haut-parleurs
Contraintes de fabrication	- Respect des délais - Économie de matières - Liaison sans-fil avec l'ordinateur - Léger

FIGURE 6.1.: Cahier des Charges du casque

Nous avons donc pu établir clairement ce dont nous avions besoin et les défauts que nous devions éviter :

— Notre masque doit pouvoir accueillir de nombreux téléphones de taille moyenne entre 4.8 et 5.1 pouces.
— Il doit pouvoir être porté sans gêne.
— Il ne doit pas bouger lorsque l'utilisateur est en mouvement, ni tomber ou se casser lors de mouvements brusques. Pour cela, nous devons veiller à réduire au maximum son poids tout en conservant une sécurité suffisante.
— Nous devons faire des économies de matières, pour diminuer les coûts de fabrication.
— Nous devons concevoir un système de sangles réglables pour maintenir au mieux le masque sur le visage.

— Tous les utilisateurs doivent pouvoir utiliser le masque grâce à un système de lentilles interchangeables, ceci afin de s'adapter à toutes les visions.
— Une mousse sera mise en place pour accroître le confort d'utilisation mais aussi pour s'adapter à toutes les morphologies.
— Le masque doit permettre le branchement d'un casque audio externe pour une immersion auditive.

Il est important de savoir que, même si le casque doit être conçu pour accueillir différents types de téléphones, tous nos tests ont été réalisés avec un LG Nexus 5. La compatibilité avec cet appareil était donc la priorité.

FIGURE 6.2.: Smartphone Nexus 5

6.2. Logiciel de simulation utilisé

Pour concevoir ce masque, nous avons fait appel à un logiciel de conception assistée par ordinateur 3D (CAO), appelé *SolidWorks*. Ce logiciel est couramment utilisé au sein de notre établissement pour la modélisation. *SolidWorks* fonctionne sous Windows. Il permet de modéliser différentes pièces, de fabriquer des surfaces variées, et de les assembler. Les pièces peuvent être observées sous différents angles et manipulées selon nos désirs. Tout comme *Autodesk Inventor Professionnal*, il permet de créer des simulations afin de tester la résistance des pièces.

Ce logiciel est utilisé par des ingénieurs et des concepteurs pour l'élaboration de plans de pièces mécaniques.

FIGURE 6.3.: Logo du logiciel SolidWorks

SolidWorks est un modeleur 3D utilisant la conception paramétrique (chaque pièce est définie par un ensemble de fonctions et de variables). Il génère 3 types de fichiers relatifs à trois concepts de base : la pièce, l'assemblage et la mise en plan. Ces fichiers sont en relation. Toute modification à quelque niveau que ce soit est répercutée vers tous les fichiers concernés.

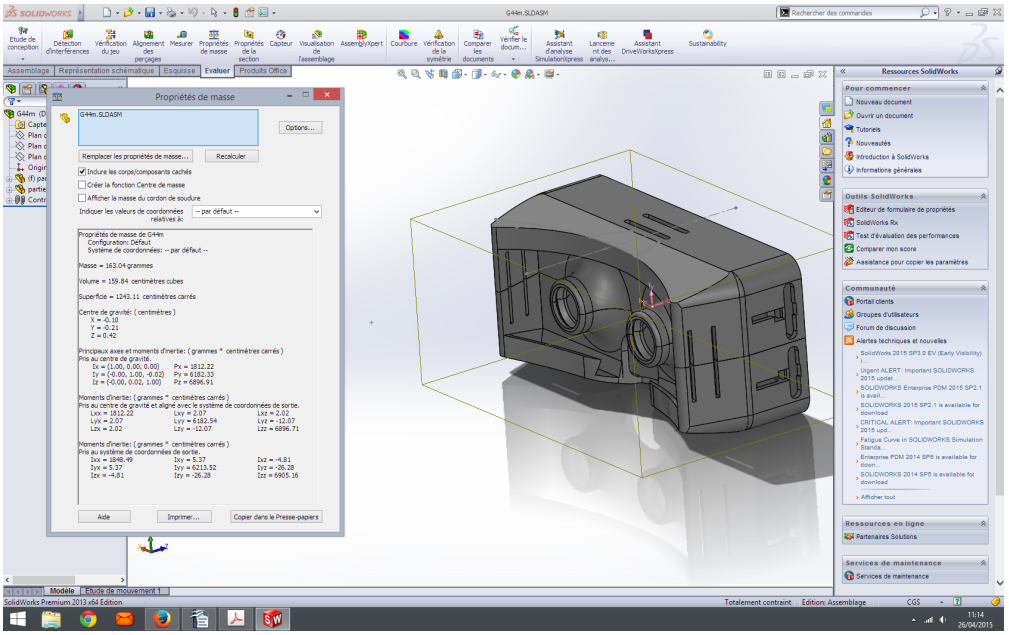


FIGURE 6.4.: Logiciel SolidWorks

6.3. Solutions envisagées

Avant d'arriver au modèle final que nous avons pu imprimer à l'aide d'une imprimante 3D, nous sommes passés par différents prototypes :

— **Version Alpha :** version très design mais peu pratique
— **Version Bravo :** ne supporte qu'un type de téléphone
— **Version Charlie :** distance des lentilles réglable (inspiré du *Homido*)
— **Version Delta :** fixation pour les sangles
— **Version Écho :** version finale et retenue

6.3.1. Version Alpha

Pour la création de notre premier prototype, nous avons voulu adopter un design futuriste. En effet, nous avions adopté une forme ovale qui ferait le tour de la tête en un seul bloc. Le téléphone devait s'insérer par le dessus.

FIGURE 6.5.: Version Alpha du casque

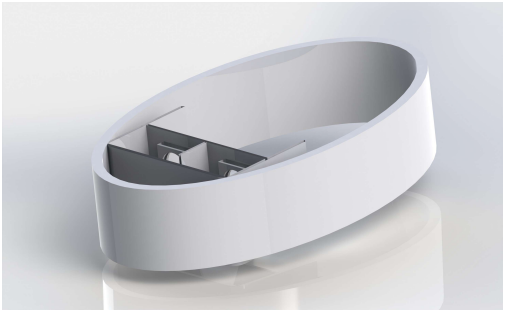


FIGURE 6.6.: Version Alpha du casque (vue de dessus)

Le plus gros inconvénient de cette solution est son poids. En effet, ce casque pesait près de 2 kg, ce qui est beaucoup trop lourd pour être porté sur la tête. De plus, il était très imposant et nous avons réalisé qu'il serait impossible de le faire imprimer dans l'imprimante 3D du lycée.

Pour les prochains modèles, nous avons donc dû réfléchir au poids et à l'encombrement.

6.3.2. Version Bravo

Dans cette version, nous avons supprimé la partie « englobante »du casque, pour ne laisser que les éléments essentiels. Ici, le téléphone doit s'insérer depuis le dessus, puis un couvercle est apposé.

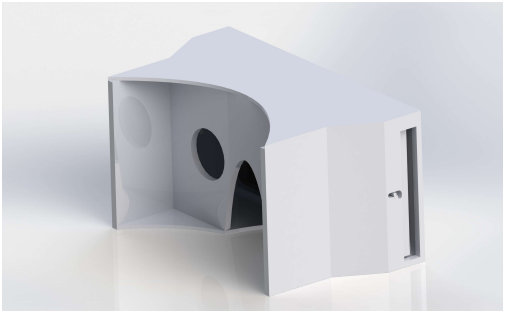


FIGURE 6.7.: Version Bravo du casque

FIGURE 6.8.: Version Bravo du casque (ouvert)

Cependant, cette version comporte plusieurs problèmes :

— Si le casque venait à se retourner, le téléphone ne serait pas retenu et tomberait
— Le casque a été fait aux dimensions du LG Nexus 5. Il est donc impossible d'y insérer un autre téléphone.

6.3.3. Version Charlie

Pour cette version, nous nous sommes largement inspirés de *l'Homido*. En effet, nous avons positionné le téléphone au dos du casque, à l'extérieur. Ainsi, il n'y a plus de restriction au niveau du téléphone utilisé.

Nous avons aussi rajouté un système permettant de régler la distance œil-lentille grâce à deux molettes sur les côtés. Ce système aurait permis d'adapter le casque à tout type de visions, notamment aux porteurs de lunettes correctrices.

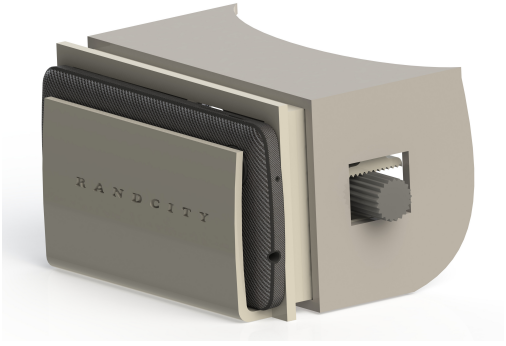


FIGURE 6.9.: Version Charlie du casque

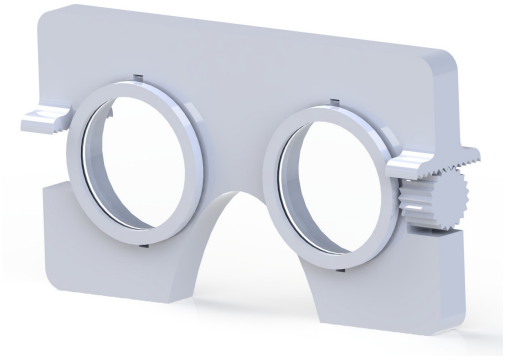


FIGURE 6.10.: Système de réglage des lentilles

Ce système semblait plutôt efficace, mais nous nous sommes rendus compte qu'il n'était pas nécessaire. En effet, pour pouvoir corriger les déficiences visuelles (myopie, presbytie, ...), le plus simple est de permettre à l'utilisateur de changer les lentilles en fonction de sa vue.

Nous avons aussi réalisé que cette version manquait cruellement d'ergonomie : sa forme n'est pas du tout adaptée au visage humain et son utilisation serait inconfortable.

6.3.4. Version Delta

Cette version est la dernière à avoir été abandonnée. Elle possède deux éléments importants :

— La prise en charge des Haut-parleurs pour une immersion auditive.
— La mise en place de fixations pour les sangles élastiques (qui maintiennent le casque sur la tête)

FIGURE 6.11.: Version Delta du casque

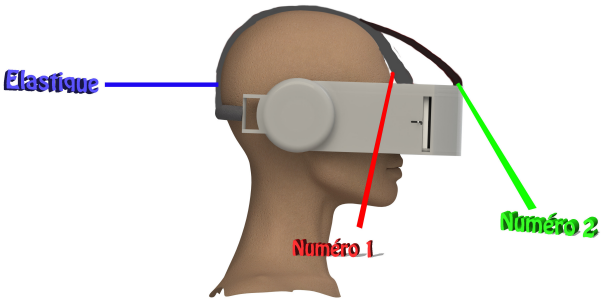


FIGURE 6.12.: Sangles élastiques

Cependant, cette version possède des défauts récurrents qui nous ont fait l'abandonner : elle est toujours trop lourde et très peu esthétique.

6.4. Solution retenue (Version Écho)

Après avoir réfléchi sur diverses solutions, nous nous sommes finalement arrêtés sur une version, dont voici quelques images :

FIGURE 6.13.: Version finale du casque (vue de dos)

FIGURE 6.14.: Version finale du casque (vue de face)

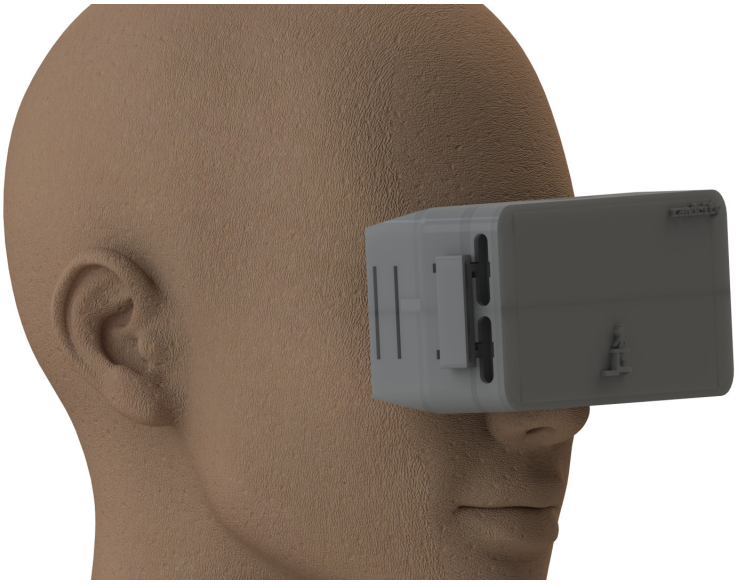


FIGURE 6.15.: Version finale du casque (sur un mannequin)

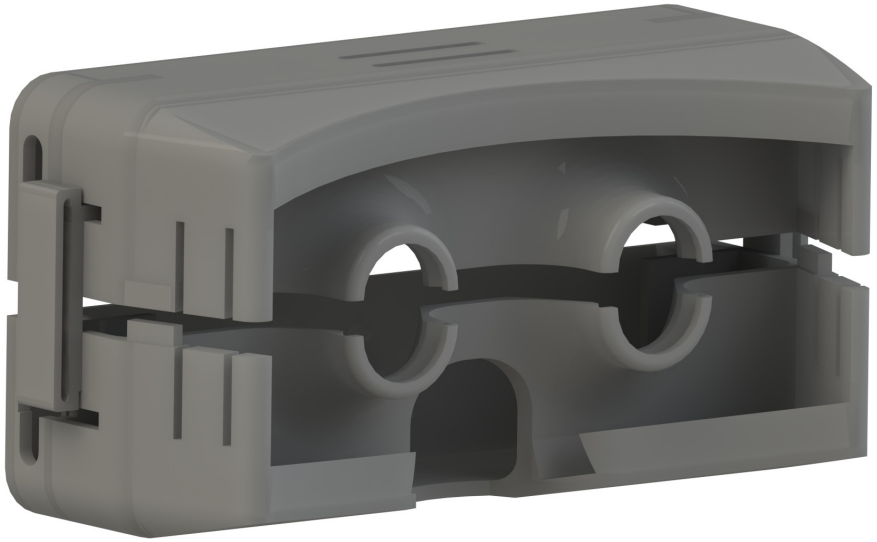


FIGURE 6.16.: Version finale du casque (ouvert)

Cette version possède une particularité inédite : le casque peut s'ouvrir en deux parties. Nous avons fait ce choix pour plusieurs raisons.

D'abord, cela nous permet d'insérer le téléphone à l'intérieur du casque lorsque celui-ci est ouvert. De plus, cela permet un accès facilité aux lentilles, afin de pouvoir les changer en cas de besoin.

La réalisation en deux parties permet aussi d'optimiser les coûts de production. En effet, l'imprimante 3D est obligée d'utiliser de la « matière noire »en dessous de chaque morceau « flottant ». Si le casque avait été d'une seule pièce, il aurait fallu le remplir entièrement de cette matière. En deux parties, ce problème est évité et les coûts sont diminués.

Le poids de l'ensemble a aussi été optimisé. Cette version ne pèse que 165 g à vide, soit moins de 300 g avec le téléphone à l'intérieur (si on utilise un Nexus 5).

Pour pouvoir insérer n'importe quel téléphone nous avons imaginé une pièce qui aurait des dimensions extérieurs standard. Elle servirait « d'adaptateur ». Ainsi, si on veut changer de téléphone, il suffirait de ré-imprimer cette pièce.

Au niveau de la sécurité, les attaches pour les sangles sur le coté servent aussi d'attaches entre les deux parties. Il est donc impossible que les deux parties se séparent pendant l'utilisation.

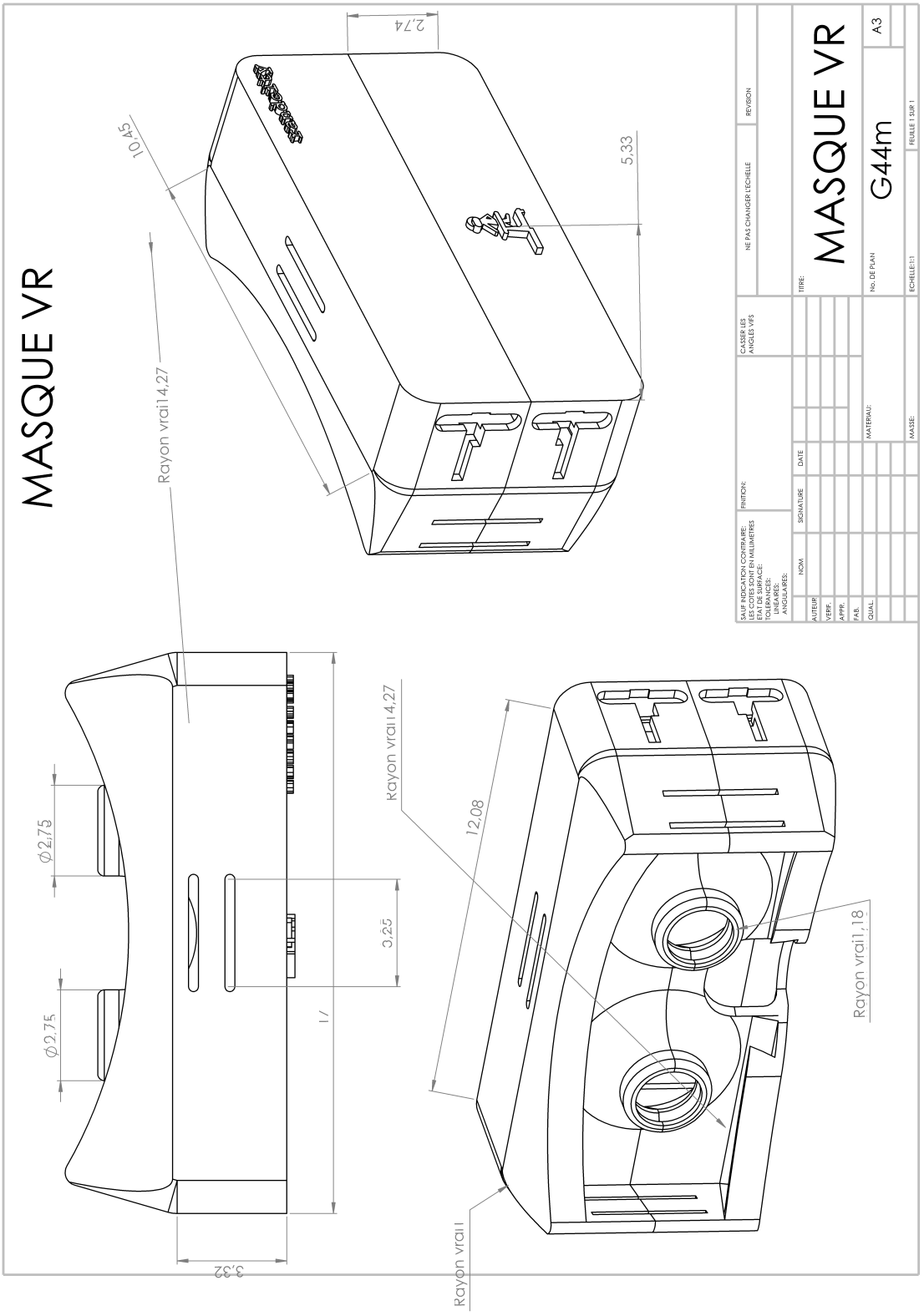


FIGURE 6.17.: Plans du casque

6.4.1. Étude qualitative

Une fois le prototypage du casque réalisé, nous avons procédé a une étude qualitative afin de vérifier que nos solutions étaient bien adéquates.

La voici :

		Satisfaction	Importance
A	Esthétique	4	3
B	Facilité D'utilisation	5	5
C	Dimension	4	4
D	Confortable	5	5
E	Léger	5	4
F	Universel	3	3

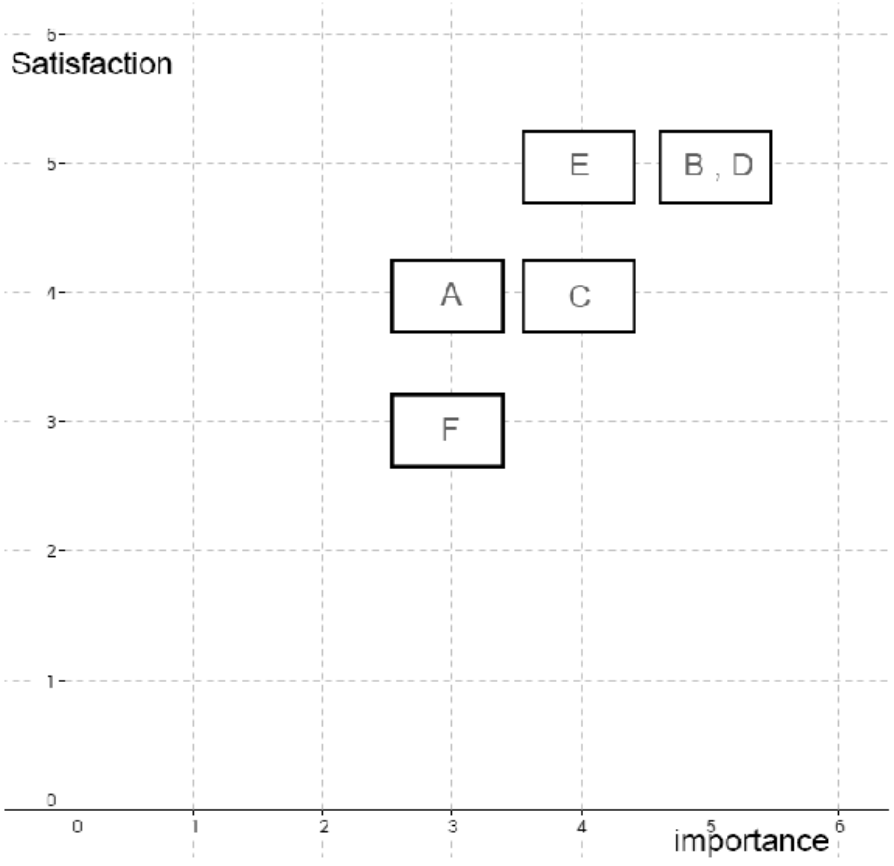


FIGURE 6.18.: Étude qualitative du casque

6.4.2. Diagramme FAST

Afin de rendre plus clair les différentes fonctions remplies par le casque, voici son diagramme FAST :

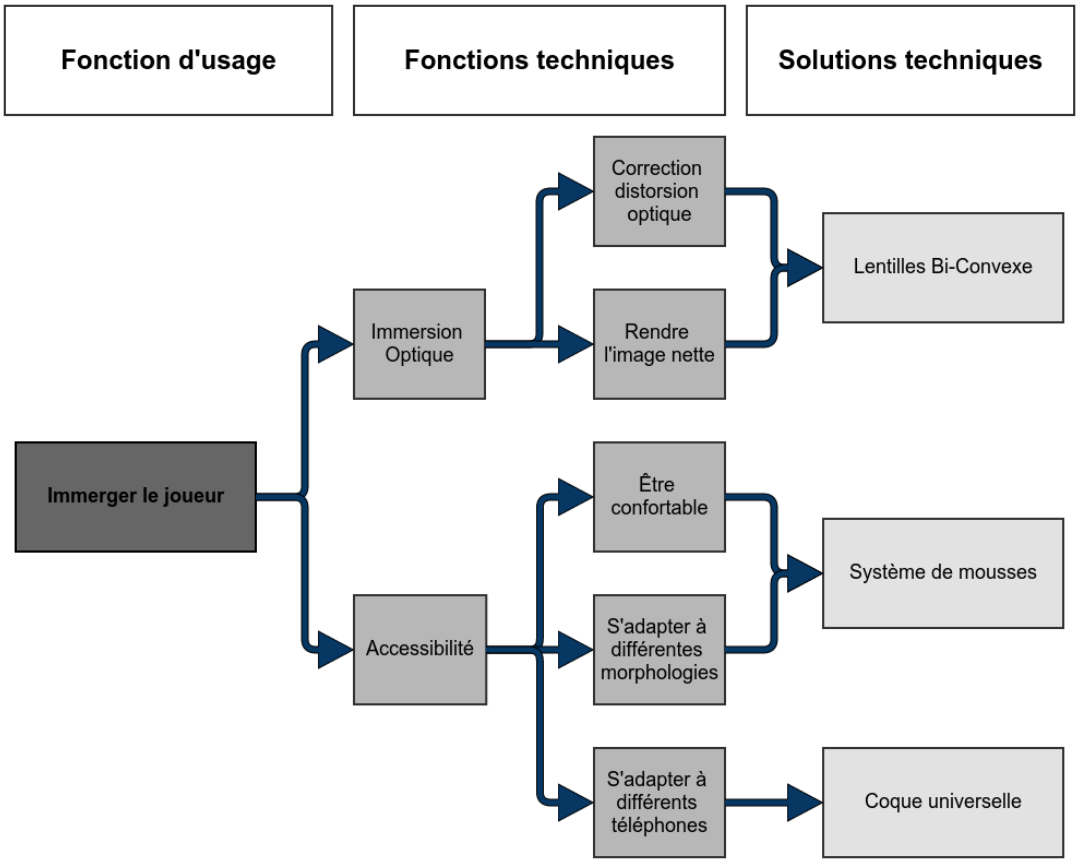


FIGURE 6.19.: Diagramme FAST du casque

6.4.3. Choix des lentilles

Pour offrir la meilleure expérience à l'utilisateur, nous devons choisir un système de lentilles capable de corriger les distorsions optiques du jeu, mais aussi offrir un large angle de vision. Le système de lentille doit aussi permettre d'avoir une vue en 3D, un peu à la manière des téléviseurs 3D (qui doivent se regarder avec des lunettes spéciales).

Dans ce but, nous avons donc effectué des calculs de distance et d'optique.

Parmi les lentilles convergentes, c'est à dire les lentilles qui vont donner un effet de « zoom », il existe 3 grandes catégories :

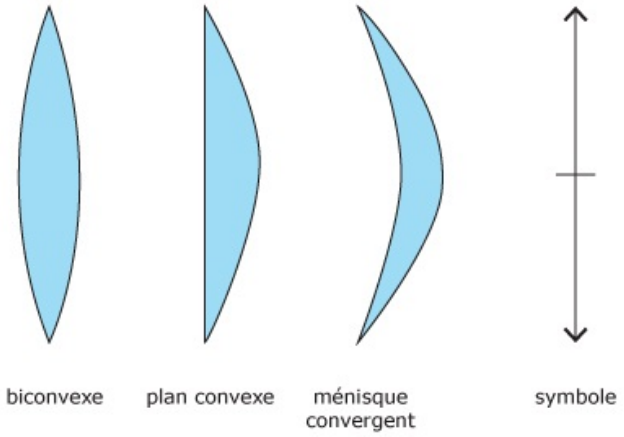


FIGURE 6.20.: Différents types de lentilles convergentes

Nous avons choisi d'utiliser des lentilles biconvexes car ce type est beaucoup plus répandu que les autres dans les dimensions que nous souhaitons (les ménisques convergents sont surtout utilisés pour les lentilles de contact par exemple).

Dans notre casque, les oculaires droit et gauche sont composés d'une simple lentille biconvexe asymétrique en plastique dont le foyer objet est situé dans le plan de l'écran d'affichage. Les rayons lumineux émis par chacun des points de l'écran (par chaque pixel) sont réfractés par l'oculaire et renvoyés vers l'infini. L'œil perçoit une image agrandie de l'écran, dont l'observation est relativement confortable. L'accommodation n'est donc pas stimulée.

L'angle d'observation découle de la faible distance entre l'œil et l'oculaire (environ 5 mm), de la distance focale de l'oculaire (environ 44 mm), et du diamètre de l'oculaire (environ 25 mm).

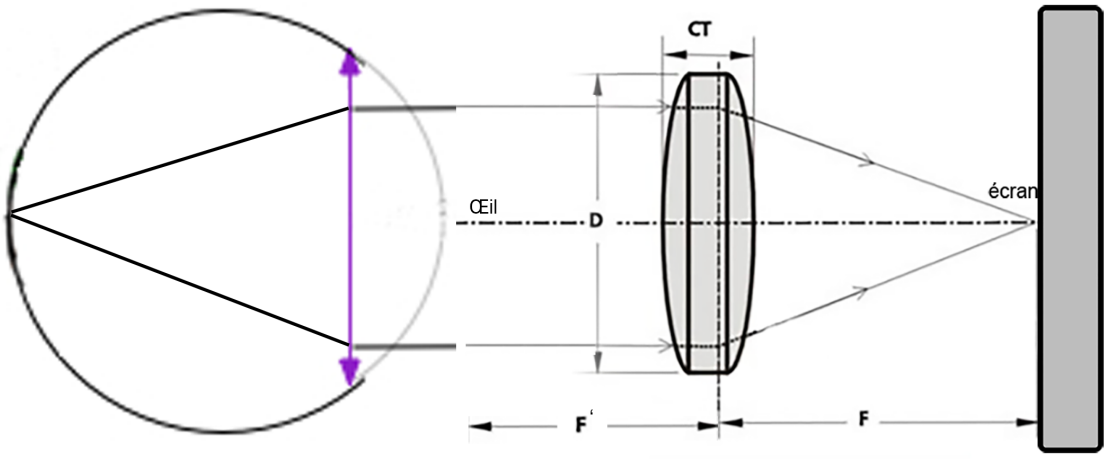


FIGURE 6.21.: Schéma optique d'une lentille Bi-Convex

D'après les calculs réalisés, les distances optimales que nous devons utiliser sont les suivantes :

— Distance écran-lentille (distance focale) : $F = 44\,\text{mm}$
— Distance œil-lentille : $F' = 5\,\text{mm}$
— Épaisseur de la lentille : $CT = 5\,\text{mm}$
— Diamètre lentille : $D = 44\,\text{mm}$
— Dimensions de l'image sur l'écran (HxL) : 55 mm x 52 mm

6.4.3.1. Pour les porteurs de lunettes

Beaucoup de personnes portent des lunettes et nous en avons tenu compte. Nous avons voulu faire un masque qui soit utilisable par le plus grand nombre.

Notre masque a la caractéristique de pouvoir inter-changer les jeux de lentilles et permettre aux myopes et aux hypermétropes d'utiliser notre masque.

Plusieurs types de lentilles peuvent être insérées dans notre masque. Les personnes myopes peuvent utiliser des verres correcteurs divergents à dioptrie négative (jusqu'à -4 dioptrie), et les personnes hypermétropes/presbytes des verres convergents à dioptrie positive (jusque +3 dioptrie).

Il suffit simplement de changer les lentilles en fonction de sa vision. Les personnes utilisant des lentilles de contact pourront utiliser les lentilles standard.

6.4.4. Réalisation

6.4.4.1. Impression 3D

Notre lycée possédant une imprimante 3D, nous avons eu l'occasion de pouvoir faire imprimer notre casque avec cette technique. Voici donc une brève explication de son fonctionnement.

Le principe des imprimantes 3D est de déposer de la matière (en générale plastique) pour construire une pièce en volume. Actuellement, il existe plusieurs technologies d'impression 3D :

— Par dépôt de résine polymérisée aux UV
— Par dépôt de poudre frittée au laser
— Par dépôt de résine polymérisée au laser
— Par dépôt de matière fondue « FDM »

L'imprimante dont le lycée est équipé utilise l'impression par dépôt de matière fondue « FDM ». L'acronyme « F.D.M. » signifie en anglais *« Fused Deposition Modeling »* et se traduit par : « Modelage par Dépôt de Filament en Fusion ». Ce procédé a été inventé en 1988 par la société *Stratasys* (marque déposée). Il consiste à déposer couche par couche un filament de matière thermoplastique fondu à 200 °C (en moyenne) qui en se superposant donne forme à l'objet.

Voici un schéma expliquant ce procédé :

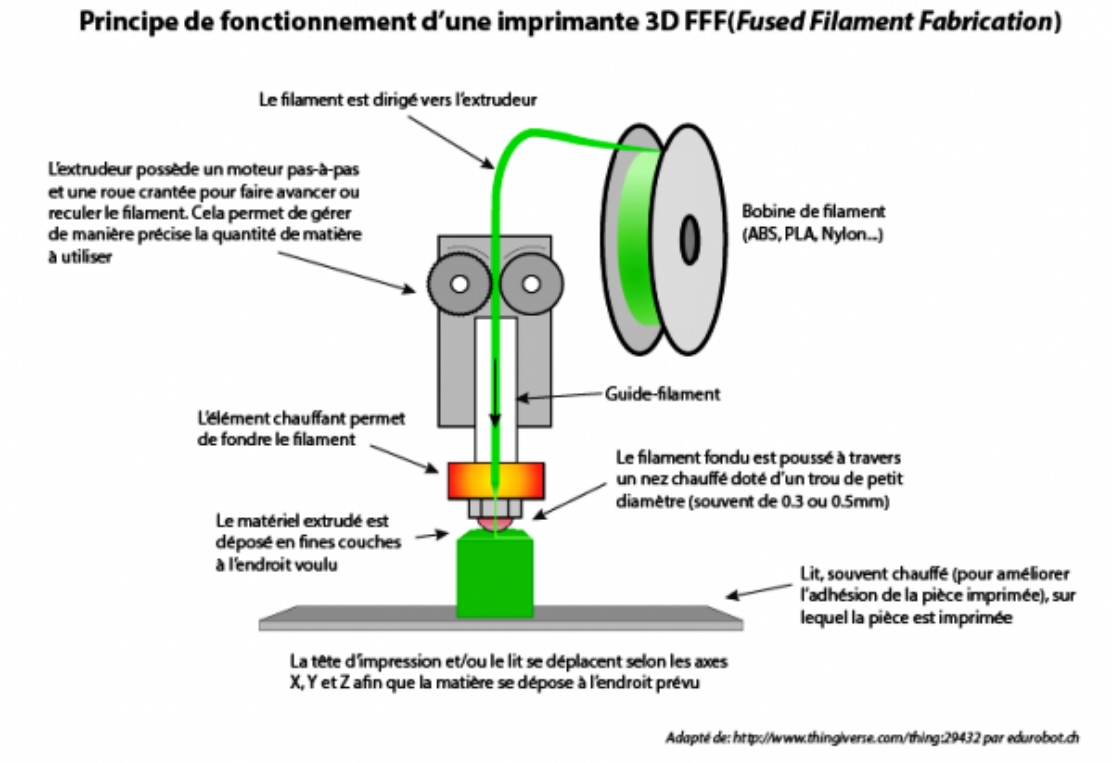


FIGURE 6.22.: Explication de l'impression 3D

La tête d'impression se déplace selon les coordonnées X, Y et Z (longueur, largeur et hauteur) transmise par un fichier 3D correspondant au modèle 3D de l'objet à imprimer. Limitée pendant longtemps à des matériaux de type plastique tels que l'ABS, l'impression 3D voit arriver de nouveaux filaments composites à base de métal (cuivre, bronze, ...) et même de bois. A l'heure actuelle, l'industrie agroalimentaire et la médecine sont en train de s'emparer de cette technique pour imprimer des aliments et des cellules en adaptant la tête d'extrusion.

Grâce à cette technique d'impression, nous avons pu imprimer notre masque précédemment conçu sous C.A.O. En voici quelques photos :

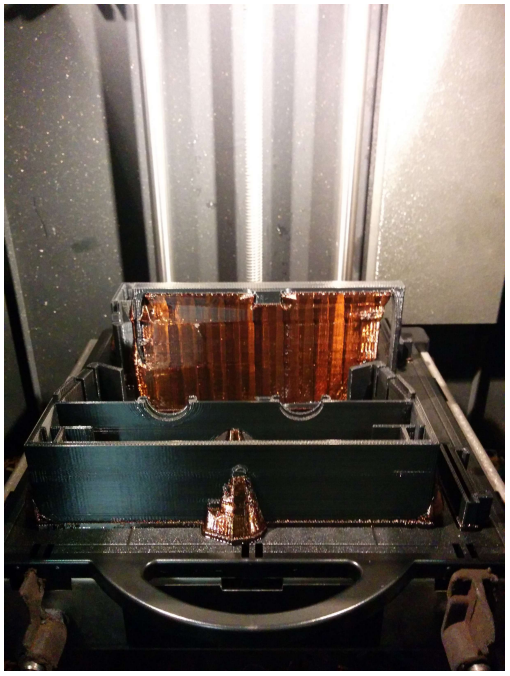


FIGURE 6.23.: Plateau d'impression 3D

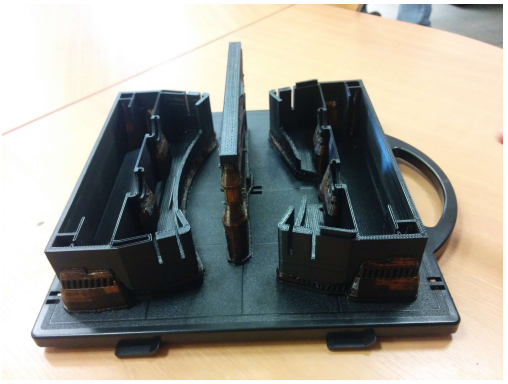


FIGURE 6.24.: Plateau d'impression 3D Bis

L'impression a duré environ 23 heures. Les plans du masque réalisés ont été préalablement transmis à l'imprimante afin que celle-ci assemble les matériaux couche après couche à la manière d'un mille-feuille. La machine doit toujours créer un support (matière cuivrée sur les photos), sorte d'échafaudage qui permet de soutenir les parties en porte à faux. C'est précisément pour économiser cette matière que le casque a été réalisé en deux parties.

La machine a donc recours à deux matériaux (et donc deux têtes d'impression) pour déposer un support de nature différente de la pièce, qui pourra se détacher facilement.

Une fois imprimées, les pièces doivent être immergées dans un bain d'acide, afin de dissoudre cette matière « support », et ne garder que la pièce définitive désirée.

6.4.4.2. Assemblage final

Lors de l'impression, toutes les pièces ont été correctement réalisées, aucune n'a cassé et nous n'avons pas eu d'erreurs graves lors de l'étape de la simulation qui aurait impactées l'impression. Le poids est aussi celui escompté, à savoir 165 g. Il ne reste plus qu'à positionner les sangles et l'ensemble sera fonctionnel.

Voici donc une revue des différents éléments du casque :

— Mise en place des lentilles sur la partie inférieure du casque

FIGURE 6.25.: Mise en place des lentilles

— Insertion du téléphone dans la coque adaptée (afin de le maintenir)

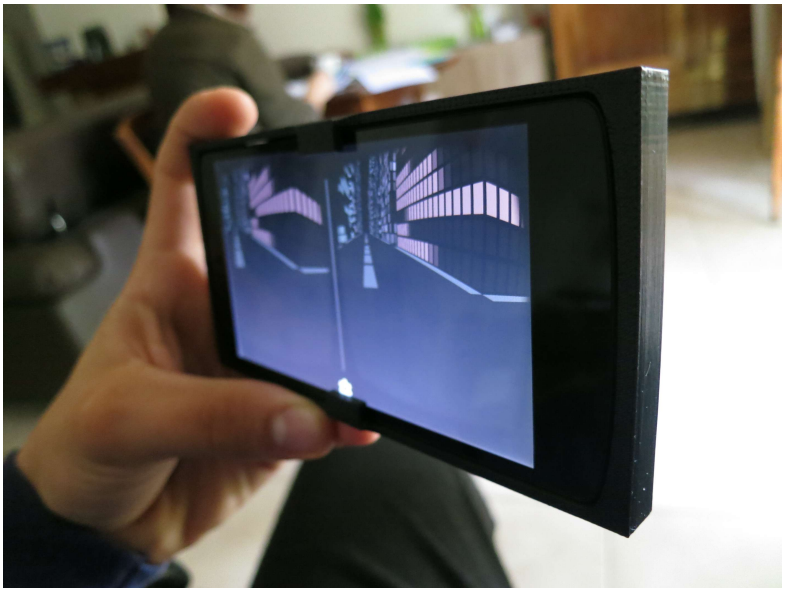


FIGURE 6.26.: Coque du téléphone

— Insertion du téléphone dans le casque

FIGURE 6.27.: Insertion du téléphone

— Mise en place des crochets (permettent d'enfiler les sangles et de lier les deux parties du casque entre elles)

FIGURE 6.28.: Crochet de sangle

— Assemblage final des deux parties avec les sangles

FIGURE 6.29.: Assemblage final du casque

Quatrième partie .

Partie Informatique : logiciel et application mobile

La partie informatique du projet se divise en deux sous-ensembles, chacun déployé sur un périphérique différent. Le premier sous-ensemble est le logiciel de détection du joueur, exécuté sur un ordinateur (de préférence portable pour être utilisé dans des environnements différents). Ce logiciel est chargé de calculer l'orientation et la vitesse de marche du joueur et de l'envoyer à l'aide d'une connexion Bluetooth vers le deuxième sous-ensemble, à savoir l'application pour smartphone Android. Cette application est chargée de recevoir les données et de les transformer en mouvements dans un environnement virtuel en 3D.

Ces deux logiciels doivent être capables de communiquer entre eux et de de s'exécuter sans latence, afin de ne pas nuire à l'expérience utilisateur.

Comme décrit à la fin de ce rapport, les logiciels ont été publiés sous des licenses OpenSource pour des raisons qui seront développées. De ce fait, nous avons fait le choix de rédiger l'ensemble de nos programmes en Anglais, qui est la langue universelle, surtout dans le domaine informatique. Cela permet aussi d'accentuer l'aspect pluridisciplinaire du projet.

Nous avons également nommé nos programmes, de la manière suivante :

— **VRController :** Logiciel de détection (constitué de « VR » qui signifie « Virtual Reality » de « Controller », c'est-à-dire « Contrôleur de Réalité Virtuelle »)
— **RandCity :** Application pour mobile (constitué de « Rand » pour « Random / Aléatoire » et de « City »). Ce nom a été choisi car la ville que l'on doit explorer est générée aléatoirement à chaque démarrage.

Voici donc une description détaillée de chacun d'eux.

7. Logiciel de détection de mouvements (VRController)

FIGURE 7.1.: Logo du logiciel VRController

7.1. Objectifs et cahier des charges

Afin de permettre l'immersion sensitive du joueur, le système doit pouvoir analyser ses mouvements. Il doit pouvoir en déduire deux valeurs : sa vitesse de marche (afin de le faire avancer dans le jeu) et son orientation (afin de savoir dans quelle direction il marche). Cette analyse doit se faire le plus rapidement possible afin d'éviter les latences.

De plus, le logiciel doit être capable d'envoyer ces données vers le téléphone de manière sans fil. Pour un meilleur confort d'utilisation, il doit pouvoir aussi afficher ces valeurs sur l'écran de l'ordinateur (afin de vérifier la justesse des informations).

De préférence, le programme devait être écrit dans un langage de programmation que nous connaissions bien, c'est-à-dire le *C++* (ou le *C*). Il doit aussi être conçu pour fonctionner sous un système d'exploitation dérivé de Linux (comme *Ubuntu* ou *Debian*). Les ordinateurs sous Windows ne sont donc pas supportés. Ceci est dû au fait que l'environnement Linux nous est plus familier et que la programmation/création de logiciels y est facilitée.

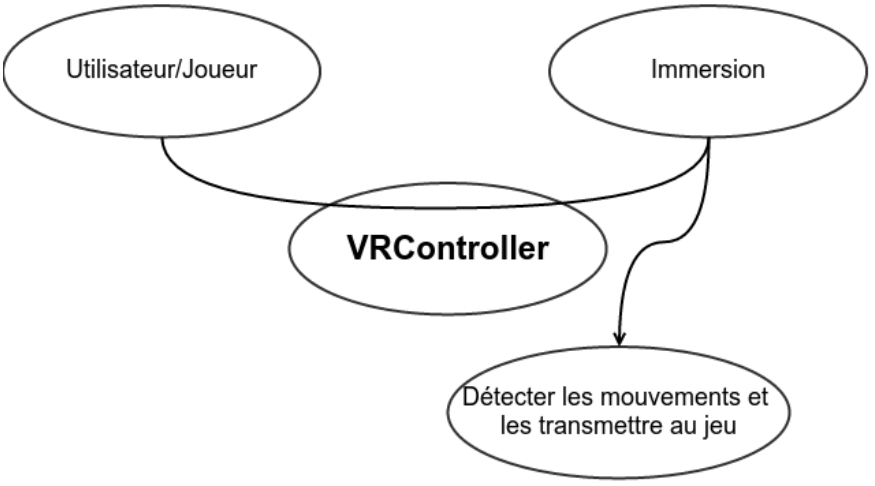


FIGURE 7.2.: Diagramme « Bête à cornes » du logiciel VRController

Pour pouvoir réaliser ce cahier des charges, nous avons donc exploré plusieurs pistes que nous allons vous présenter.

7.2. Recherche de solutions

7.2.1. Transmission des données sans fil

Afin de pouvoir envoyer les données calculées par le logiciel, nous avons dès le début omis les connexions par câbles. En effet, si le casque devait être relié à l'ordinateur avec un câble, cela aurait entravé les mouvements du joueur et aurait nuit à son immersion. Nous avons donc décidé d'utiliser une méthode de communication sans fil. Or, il fallait que la méthode utilisée soit à la fois supportée par le téléphone et l'ordinateur.

Nous avons donc étudié 2 protocoles en particulier :
— WiFi
— Bluetooth

Le *WiFi* est idéal pour transmettre des données importantes sur des distances allant jusqu'à quelques dizaines de mètres. Cependant, dans le cadre de notre projet, nous n'avions pas besoin d'une telle portée. Aussi, les données que nous faisons transiter sont très minimalistes et se résument à 4 octets par paquet. C'est pour cela que nous avons choisi d'utiliser le *Bluetooth*, qui est pratique pour ce genre d'échange (même si sa portée ne dépasse pas les 5 mètres).

Afin d'utiliser le *Bluetooth* comme moyen de communication, nous avons décidé d'utiliser la librairie **BlueZ**, qui permet de contrôler la carte Bluetooth de l'ordinateur. Cette librairie est écrite en *C* et n'est disponible que sur Linux (c'est d'ailleurs la librairie Bluetooth officielle du Kernel Linux).

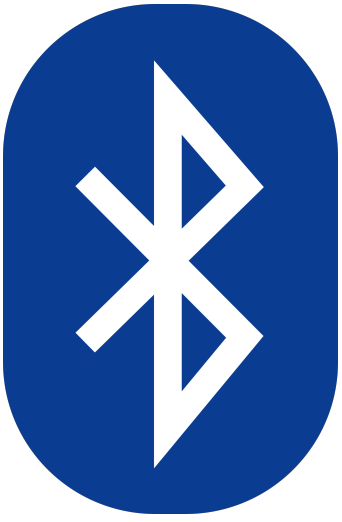


FIGURE 7.3.: Logo du protocole Bluetooth

7.2.2. Détection des mouvements

Pour pouvoir détecter les mouvements du joueur, nous ne voulions pas que celui-ci soit équipé de capteurs. En effet, nous voulions que le joueur puisse utiliser notre système sans avoir un équipement spécifique. Nous nous sommes donc tournés vers les caméras. Au début, nous voulions utiliser un ensemble de 3 Webcam (pour éviter les angles morts) qui filmaient le joueur et qui devaient analyser ses mouvements juste grâce aux traitements de l'image. Mais nous sommes rendus compte que cela était bien trop compliqué et pas accessible à notre niveau.

A ce moment là, nous avons réalisé que le lycée possédait une caméra Kinect et qu'il serait intéressant de s'en servir.

FIGURE 7.4.: Caméra Kinect de Microsoft

A l'heure où ce projet a été réalisé, il n'existe que peu de solutions pour exploiter ses capacités :

— OpenKinect
— OpenNI

7.2.2.1. OpenKinect

OpenKinect est un projet OpenSource visant à contrôler et à exploiter la puissance de la Kinect. Il est multi-plateforme et n'a été réalisé qu'en utilisant des outils de rétro-ingénierie (la Kinect étant un appareil propriétaire, son fonctionnement interne n'a pas été dévoilé officiellement).

Cependant, ce projet est ambitieux et encore jeune. Il ne permet pour l'instant que de récupérer les coordonnées dans l'espace de chaque pixel affiché, ainsi que le contrôle des moteurs (pour faire pivoter la Kinect). Nous avons donc dû choisir une autre solution pour notre projet car OpenKinect ne prend pas en charge la reconnaissance d'un joueur.

FIGURE 7.5.: Logo de la librairie OpenKinect

Par exemple, les programmes mis a disposition par OpenKinect permettent de récupérer des images de ce genre :

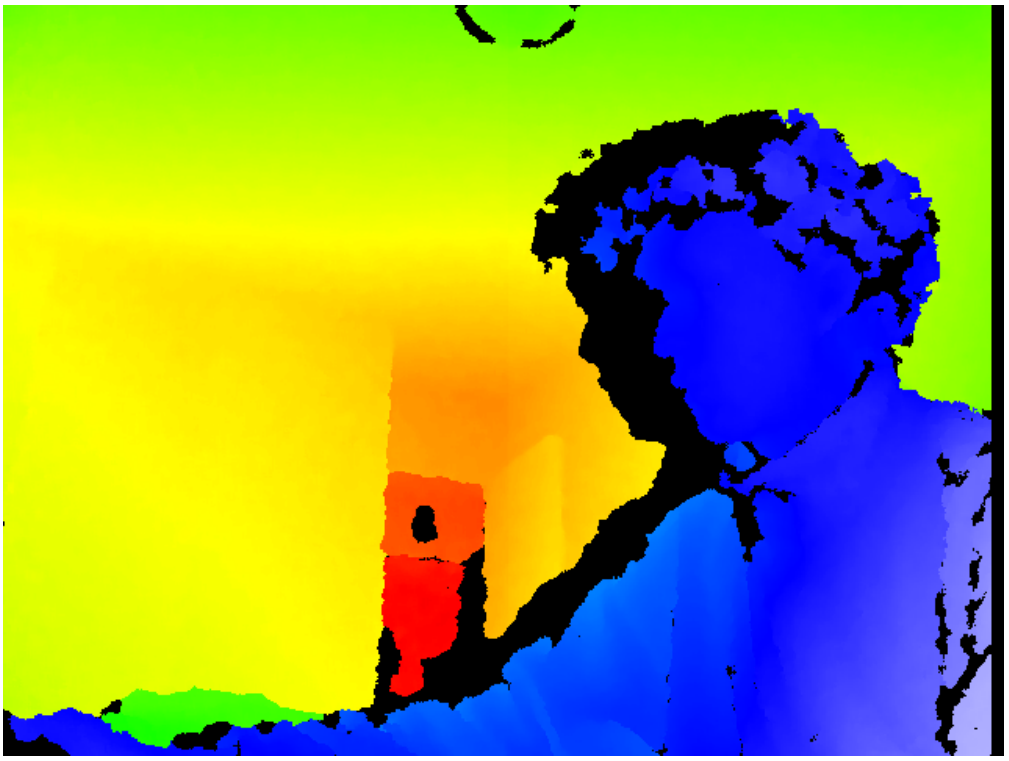


FIGURE 7.6.: Programme d'exemple d'OpenKinect

Sur cette image, on voit que plus la couleur tend vers le rouge, plus l'objet est loin, tandis que le bleu montre les objets près. Ceci serait suffisant pour réaliser une cartographie de la pièce par exemple, mais ne nous permet pas de détecter le joueur (et surtout son squelette).

7.2.2.2. OpenNI

OpenNI (Open Natural Interaction) est un projet de plus grande envergure visant à unifier le monde des capteurs 3D dans une unique interface. Ainsi, il est possible d'utiliser aussi bien la Kinect que d'autres capteurs du même genre. Cependant, suite au rachat de *PrimeSense* (la société qui a conçu cette librairie) par *Apple*, le projet n'est plus officiellement supporté et il a été supprimé de tous les sites de téléchargement.

Mais comme le code source était public, il a été conservé sur des sites de partage de code source (comme *GitHub*). Nous avons donc pu le récupérer et l'utiliser pour VRController. Il faut cependant savoir que la fonctionnalité qui permet de récupérer le squelette du joueur est fournie par une sous-librairie (*NiTE*), qui elle, est propriétaire (son code source n'est pas public). Nous avons cependant réussi à la retrouver et à l'utiliser.

FIGURE 7.7.: Logo de la librairie OpenNI

Il faut cependant noter qu'au cours du développement de notre logiciel et de la découverte de la librairie OpenNI, nous avons dû faire face à plusieurs problèmes :

— La librairie ne fonctionne pas sur les processeurs *AMD*. Ceci est dû à un bug de OpenNI (qui essaye toujours d'utiliser un jeu d'instruction spécial, seulement défini par *Intel*, le jeu d'instructions *SSE3*)
— Lorsque nous avons trouvé la sous-librairie chargée de la reconnaissance du squelette, nous n'avons pas pu trouver les versions 32 bits. Notre programme se limitera donc aux ordinateurs 64 bits (avec un processeur *Intel*).

7.2.2.3. Fonctionnement de la Kinect

La caméra Kinect était initialement vendue avec la console de jeu *XBox 360* et avait justement pour but de détecter les mouvements des joueurs. Celle-ci est capable de générer une « carte de profondeur », qui sera justement utilisée par OpenNI pour détecter le squelette du joueur.

« Carte de profondeur » Dans le monde de l'imagerie informatique, on peut représenter une image comme une grille de pixels, c'est à dire une grille de petits carreaux ayant chacun sa propre couleur. Chaque pixel est également représenté à l'aide de 3 composantes colorimétriques : le rouge, le vert, le bleu. On appelle ce codage le codage *RGB* (*Red Green Blue*) ou *RVB* en français (*Rouge Vert Bleu*).

bigskip Ainsi, en combinant ces composantes (qui sont en réalité des nombres stockés sur 1 octet chacun, c'est a dire entre 0 et 255), on peut obtenir une couleur.

Par exemple, le rouge peut s'écrire $(255;0;0)$, le noir $(0;0;0)$ ou le jaune $(255;255;0)$.

Mais dans une « carte de profondeur », qui n'est en fait rien d'autre qu'une image, les pixels sont stockés différemment. Pour chaque pixel n'est utilisé qu'un seul nombre stocké sur 16 bits (soit deux octets), compris entre 0 et 65535. Cette valeur représente la distance en *mm* qu'il y a entre la caméra, et le point représenté par le pixel correspondant. Ainsi, si un point est situé à **2,49 m** de la caméra, il serait représenté par un pixel d'une valeur de **2490**. A noter cependant que chaque pixel ayant une valeur nulle signifie que la caméra n'a pas pu déterminer à quelle distance est le point.

Comme chaque pixel n'est représenté qu'avec un seul nombre, la « carte de profondeur » est souvent affichée comme une image en nuance de gris, où chaque valeur représente cette nuance.

Génération de la « carte de profondeur » La caméra Kinect n'est pas une simple caméra, elle intègre plusieurs capteurs ainsi qu'une puce afin de pouvoir calculer la distance de chaque point vu par la caméra.

En effet celle-ci est en réalité un assemblage de plusieurs caméras. En plus de la caméra ordinaire, elle possède aussi une caméra infrarouge ainsi qu'un émetteur infrarouge.

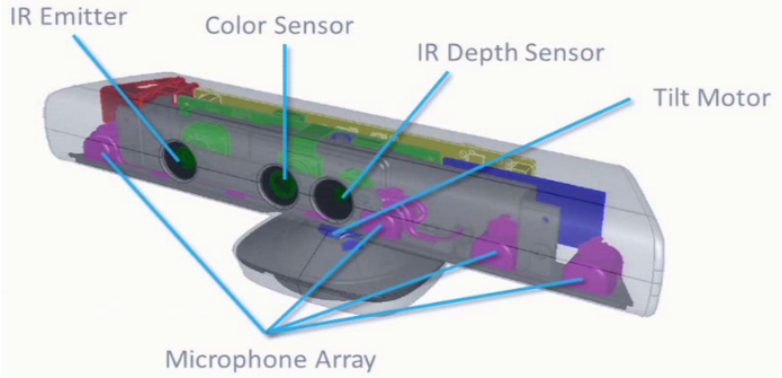


FIGURE 7.8.: Différents capteurs de la Kinect

L'émetteur est chargé de projeter un motif constitué de milliers de petites taches, que la caméra infrarouge sera capable de filmer (ce motif est invisible à l'œil nu car les infrarouges ne font pas partie du domaine du visible). Une fois le motif filmé, la Kinect est capable d'en analyser les déformations et donc de définir si un point est plus proche ou plus loin qu'un autre.

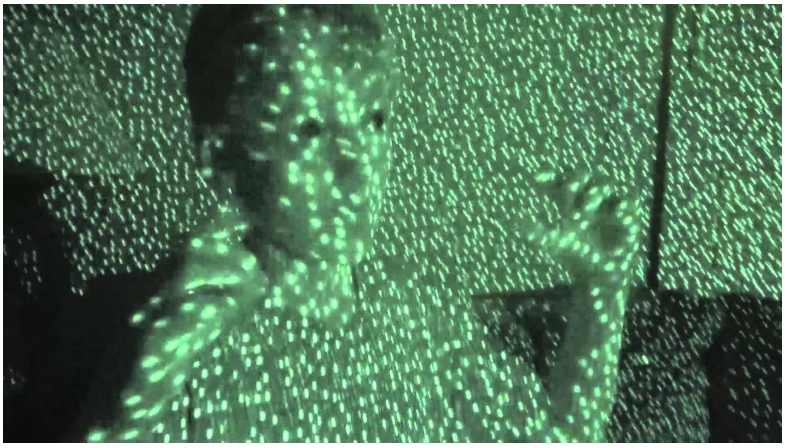


FIGURE 7.9.: Motif infrarouge de la Kinect

Sur cette image, on voit bien que les taches sur son visage sont plus larges que celles sur le mur du fond. De cette déformation, la caméra Kinect arrive à calculer une distance pour chaque point de l'image de manière relativement précise (au mm près).

Une fois l'analyse faite, la caméra est en mesure de générer la « carte de profondeur » que nous utilisons.

7.3. Solution retenue

Nous avons donc décidé d'utiliser la librairie **OpenNI** pour la gestion de la Kinect et la librairie **BlueZ** pour le Bluetooth. Le programme utilise également d'autres librairies qui nous permettent différentes choses. En voici un résumé :

— **OpenNI :** gestion de la Kinect
— **BlueZ :** gestion du Bluetooth
— **Qt :** permet de créer des interfaces graphiques (boutons, fenêtres, ...)
— **OpenCV :** permet d'afficher le retour vidéo de la Kinect en temps réel

Le logiciel a donc été réalisé pour un environnement Linux et est écrit en C++, un langage de programmation orienté objet.

A titre d'information, VRController possède **3511 lignes de codes exclusives**. Cela signifie que dans ce nombre ne sont comptées uniquement que les lignes que nous avons écrites nous-mêmes. Les librairies tierces ne figurent pas dans ce total, ni les lignes vides ainsi que les commentaires. Pour réaliser ce calcul, nous avons utilisé un utilitaire appelé *cloc*, disponible gratuitement sur le site SourceForge.

FIGURE 7.10.: Logo de la librairie Qt

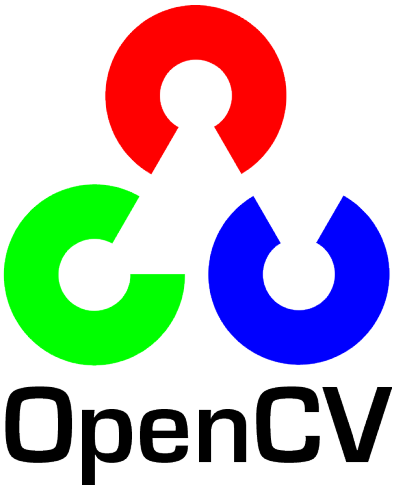


FIGURE 7.11.: Logo de la librairie OpenCV

7.3.1. Diagramme FAST

Pour résumer l'ensemble des fonctions du logiciel dans sa version définitive, voici son diagramme FAST :

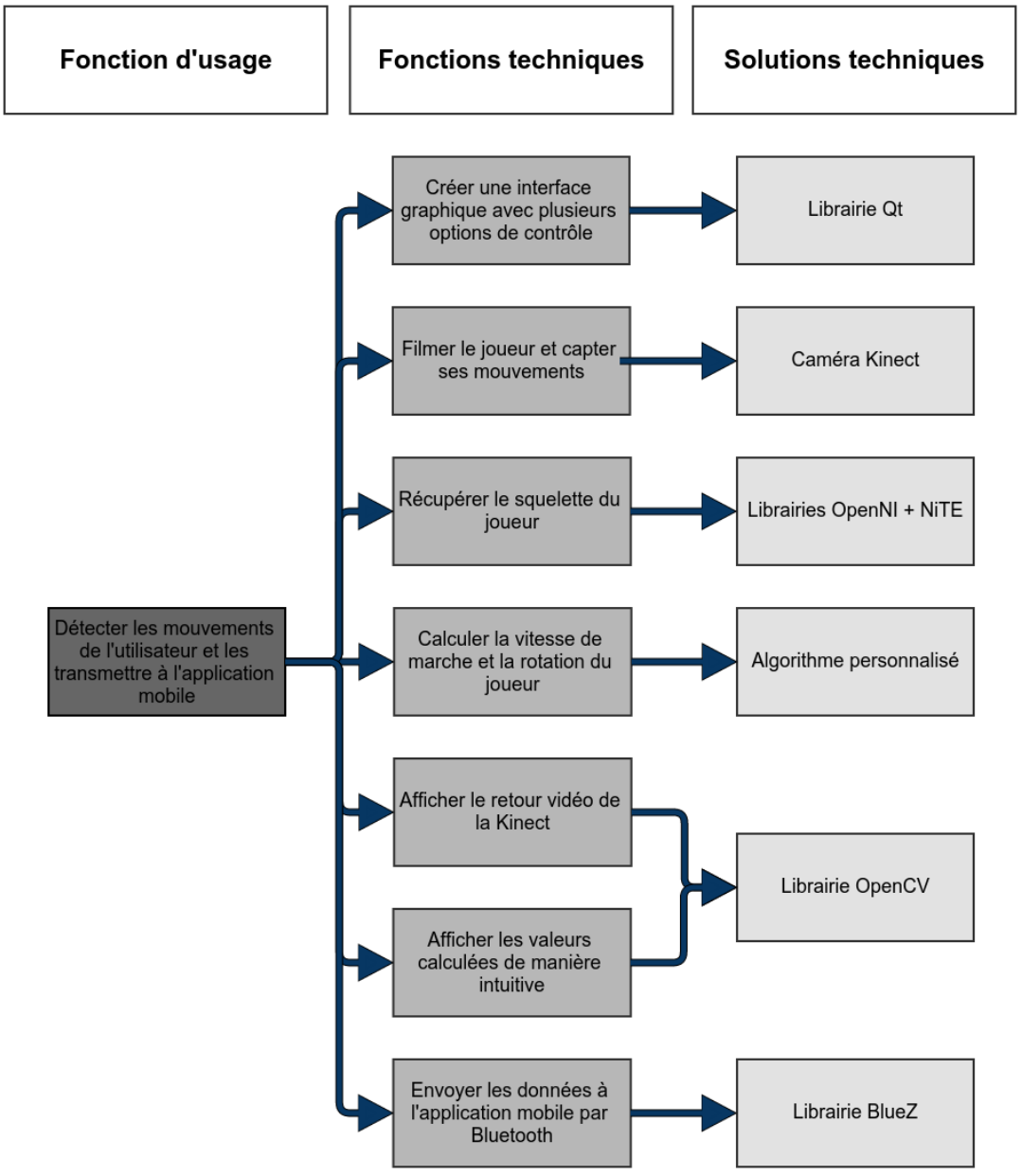


FIGURE 7.12.: Diagramme FAST de VRController

7.4. Présentation technique du logiciel

Passons maintenant à une présentation plus technique du logiciel VRController en détaillant chaque élément du programme.

7.4.1. Multithreading

Dans un programme basique, chaque instruction est exécutée l'une après l'autre. Si une instruction est très longue ou prend plus de temps que prévu, alors les suivantes sont bloquées, tout comme l'ensemble du programme.

Le multithreading est une technique qui permet d'exécuter plusieurs instructions en même temps, de manière parallèle, un peu à la manière de ce que fait le système d'exploitation quand vous écoutez de la musique et surfez sur *Facebook* en même temps. Cela permet de créer des applications dynamiques, capables de faire plusieurs choses en même temps.

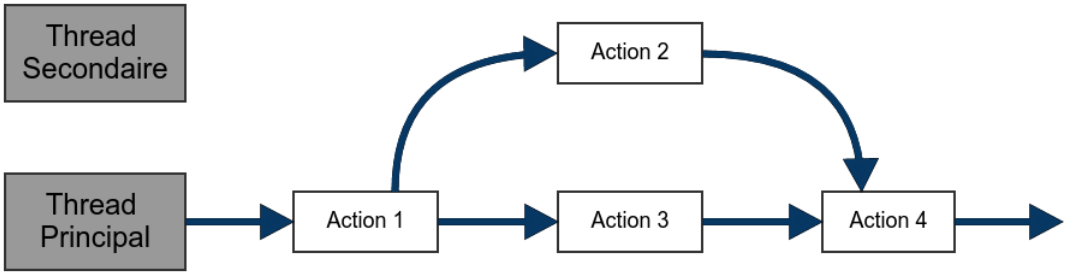


FIGURE 7.13.: Schéma d'explication du multithreading

Comme vous pouvez le voir sur le schéma, le programme se lance sur le Thread Principal et commence à exécuter sa première action. Si l'action est une action gourmande en ressources ou en temps, le développeur peut choisir de la faire s'exécuter dans un Thread Secondaire. Ainsi les actions 2 et 3 pourront se faire en même temps. On peut voir aussi qu'une fois l'action 2 terminée, son résultat est renvoyé vers le Thread principal.

Dans VRController, cette technique est utilisée lors de deux moments clés :

— Lors de l'attente d'une connexion Bluetooth par l'application mobile
— Pour récupérer en continu les informations en provenance de la Kinect

7.4.2. Menu Principal

Lorsque vous lancez le logiciel VRController, vous arrivez sur le menu principal où sont présentes différentes options :

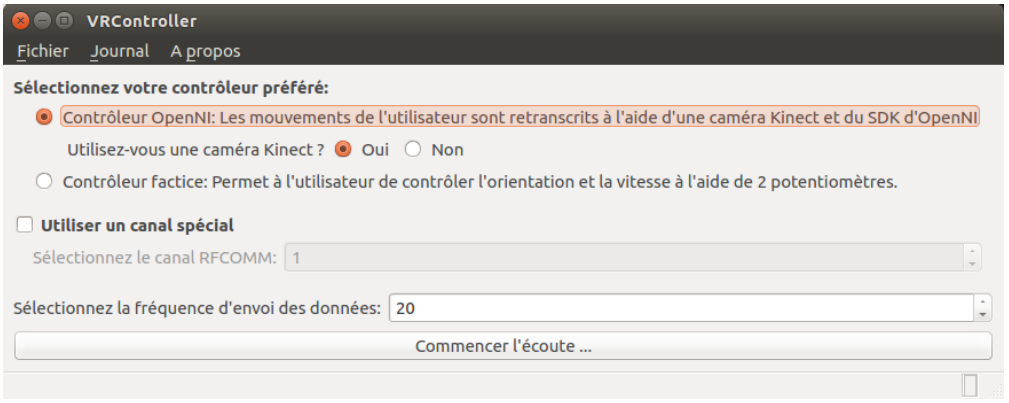


FIGURE 7.14.: Menu principal de VRController

Cette fenêtre et l'ensemble des composants utilisent la librairie *Qt*, qui permet de créer des interfaces graphiques. Avec cette librairie, vous pouvez créer une fenêtre, y ajoutez des Widgets (c'est-à-dire des composants, que ce soit des boutons ou une zone de saisie) et les arrangez comme bon vous semble.

Voici par exemple un morceau de code qui permet d'ajouter la zone de saisie pour la fréquence d'envoi des données Bluetooth :

```
_frequencyBox = new QSpinBox(this);
_frequencyBox->setRange(1, 100);
formLayout->addRow(tr("Set the frequency of data send:"), _frequencyBox);
```

FIGURE 7.15.: Code d'ajout d'une boite de sélection numérique

Sur ce menu, vous pouvez voir différentes choses :

— Le choix du contrôleur que vous voulez utiliser. Ce choix sera détaillé plus tard dans ce rapport.
— Des options pour contrôler le Bluetooth :
  — Le canal à utiliser (défini automatiquement par défaut)
  — Le nombre de données à envoyer par seconde (plus il y en a, moins le mouvement est saccadé, mais plus les ressources de l'ordinateur sont sollicitées)
— Un bouton pour commencer l'écoute. Ce bouton sert à lancer la partie Bluetooth du programme.

Il existe également une option (à l'intérieur de la barre de menu) qui permet d'afficher le journal des événements :

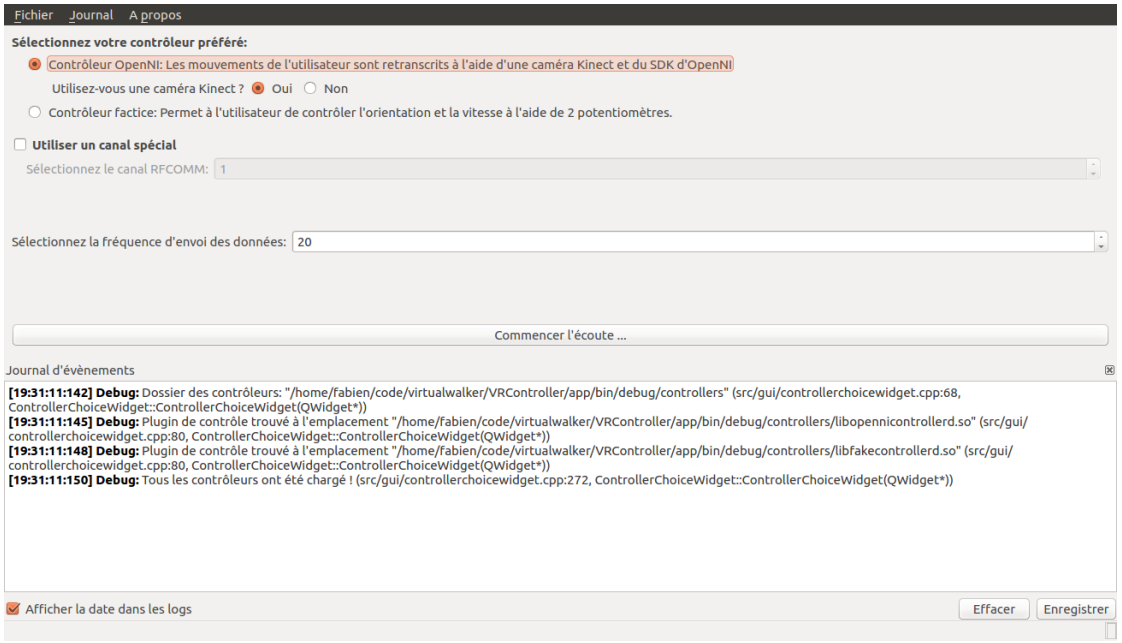


FIGURE 7.16.: Menu principal de VRController avec Journal des Événements

Dans ce journal sont consignées toutes les actions internes qui sont exécutées par le programme. Cela est utile lors de la phase de création pour être tenu au courant de ce que le logiciel fait (ou ne fait pas) et aide à déceler les problèmes éventuels.

7.4.3. Gestion du Bluetooth

Lorsque vous cliquez sur le bouton « Commencer l'écoute », le gestionnaire Bluetooth se met en route. Ce gestionnaire est une classe que nous avons spécialement créée. Celle-ci est complètement indépendante et peut-être réutilisée dans n'importe quel autre programme.

Avant de pouvoir envoyer des messages vers l'application mobile, le gestionnaire doit passer par plusieurs états :

```
enum class State
{
    // Default state
    NO_STATE = 0,

    CONNECTED_TO_SOCKET = 1,
    BOUND_TO_SOCKET = 2,
    LISTENING = 3,
    CONNECTED_TO_CLIENT = 4
};
```

FIGURE 7.17.: Enumération des différents état du gestionnaire Bluetooth

L'état *NO_STATE* est utilisé lorsque qu'une erreur se produit.

L'état *CONNECTED_TO_SOCKET* est utilisé lorsque le gestionnaire arrive à se connecter à la carte Bluetooth de l'ordinateur. Ceci s'effectue grâce à la commande :

```
_socket = socket(AF_BLUETOOTH, SOCK_STREAM, BTPROTO_RFCOMM);
```

FIGURE 7.18.: Connexion au socket Bluetooth local

On peut voir que les arguments de cette fonction signifient que l'on veut établir un pont pour le Bluetooth en utilisant le *protocole RFCOMM*. Ce protocole est le protocole de base du Bluetooth et est l'équivalent du *TCP* pour les réseaux IP.

Le second état, *BOUND_TO_SOCKET* est défini une fois que ce pont est opérationnel (quand le système d'exploitation aura accepté notre demande).

Puis vient l'état *LISTENING*. Celui-ci est appelé une fois que le gestionnaire a lancé une écoute (il est en attente de connexion). Cependant, l'écoute ne peut se faire que sur un canal précis, celui précisé par l'utilisateur dans le menu principal. Pour que le téléphone puisse connaître le canal à utiliser, nous avons recours a un autre protocole, le *SDP (Search Device Protocol)*.

Ce protocole va lancé un « mini-serveur », qui aura pour but d'indiquer à tous les périphériques le canal utilisé par le vrai serveur. Pour éviter que tous les périphériques s'y connectent, il faut avoir connaissance de l'ID du serveur. Cet ID est donc la seule information prédéfinie à la fois dans VRController et RandCity.

Une fois ce protocole lancé, le gestionnaire va démarrer un nouveau Thread, qui aura pour but d'attendre qu'un périphérique se connecte.

```
_acceptThread = std::thread(&BluetoothManager::acceptConnection, this,
                            _socket, (struct sockaddr *)&_remoteSockAddr, &_remoteSockLength,
                            [this](int clientID) {
    _client = clientID;

    if(_client < 0)
        appendError(Error::CLIENT_CONNECTION);
    else
    {
        setState(State::CONNECTED_TO_CLIENT);

        // Close the SDP service when connected
        sdp_close(_sdpSession);
    }
});
```

FIGURE 7.19.: Thread d'écoute Bluetooth

On peut voir que lorsque le Thread se termine, soit il y a eu une erreur, soit on passe à l'état *CONNECTED_TO_CLIENT*. Dans cet état, nous sommes maintenant capables d'envoyer des messages. La fréquence d'envoi des messages est définie par l'utilisateur dans le menu principal.

7.4.3.1. Envoi de messages

Pour que le logiciel et l'application mobile se comprennent, il faut que les messages soient toujours constitués de la même manière. Voici donc le datagramme que nous utilisons pour la transmission de messages.

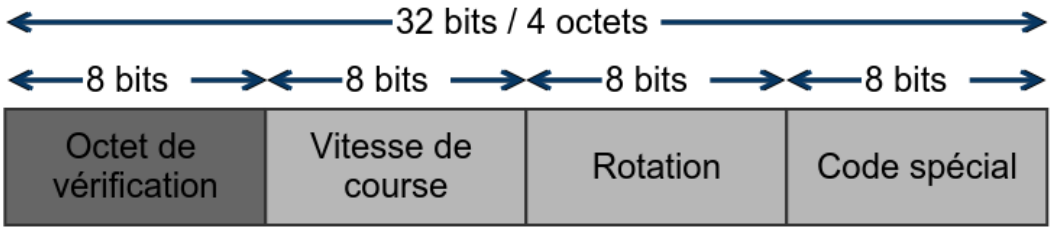


FIGURE 7.20.: Datagramme Bluetooth

Vous pouvez voir sur ce schéma que les datagrammes envoyés sont très courts. En effet, ils ne sont constitués que de 4 octets dont voici leur signification :

- **Octet de vérification :** toujours défini sur une valeur de 255 (soit 11111111_2). Il permet au jeu de savoir où est le début du message. Cet octet est le seul à pouvoir prendre cette valeur.
- **Vitesse de course :** nombre entre 0 et 254 (entre 00000000_2 et 11111110_2). Indique la vitesse de déplacement du joueur.
- **Rotation :** nombre entre 0 et 254. Indique la rotation du joueur. A noter que la rotation est normalement un nombre entre 0 et 360. Pour pouvoir la transmettre avec un seul octet, nous lui appliquons le ratio suivant : $\frac{255}{360}$. Avec ce ratio, nous avons une perte de 30% environ. Cela peut sembler important, mais il reste quand même 255 orientations possibles, ce qui est largement suffisant face à la précision de notre algorithme de détection. Ce ratio est également appliqué lors de la réception par le jeu, mais à l'inverse.
- **Code spécial :** permet d'envoyer des instructions spécifiques au jeu (tout autre valeur est ignorée) :
  - **1 :** envoyé pendant les 3 premières secondes. Permet d'initialiser la rotation. Cela permet d'éviter les décalages éventuels entre l'orientation du téléphone et celle calculée par la Kinect. Il faut que le joueur bouge le moins possible pendant ces 3 secondes pour obtenir de meilleurs résultats.
  - **2 :** met à jour l'orientation de l'application mobile en fonction de celle envoyée. Ce code est transmis lorsque le joueur a cessé de courir pendant plus d'une seconde : il permet de re-synchroniser les appareils.
  - **3 :** envoyé au bout de 3 secondes, pour dire au téléphone que l'initialisation est terminée.

7.4.4. Contrôleur factice

Au sein du menu principal vous pouvez choisir quel contrôleur vous voulez utiliser. Si vous choisissez le contrôleur factice, la détection de mouvements ne sera pas activée. À la place, vous aurez deux potentiomètres qui vous permettront de contrôler la vitesse et l'orientation.

FIGURE 7.21.: Potentiomètres du contrôleur factice

Ce mode n'est pas très utile pour les joueurs mais il a été laissé car il est très pratique pour tester l'envoi des données par Bluetooth. Il a été aussi le premier élément créé du logiciel, bien avant la partie de reconnaissance des mouvements.

7.4.5. Contrôleur OpenNI

Ce contrôleur est l'élément principal du programme VRController. Il va gérer à lui seul tout le système de reconnaissance des mouvements. Mais avant de parler des détails techniques, parlons un peu plus de la librairie OpenNI.

En effet, cette librairie permet de récupérer deux éléments importants :

— **La carte de profondeur :** surtout utilisée pour avoir un retour vidéo à l'écran
— **La liste des articulations du joueur :** nous permet de faire l'ensemble des calculs. Chaque articulation est un point qui possède des coordonnées $(X;Y;Z)$ dans l'espace mais aussi des coordonnées $(X;Y)$ correspondantes dans l'image.

OpenNI est capable de récupérer les articulations suivantes :

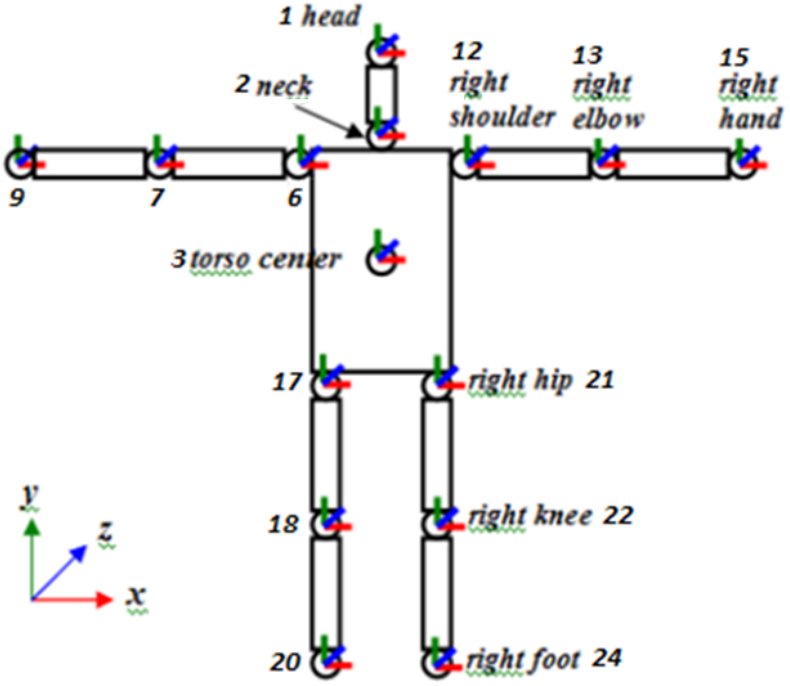


FIGURE 7.22.: Articulations d'OpenNI

Dans le cadre de notre programme, nous nous intéressons surtout à la partie inférieure du corps, c'est pourquoi nous ne récupérons que les articulations suivantes :

— Pieds droit/gauche (24-20)
— Genoux droit/gauche (22-18)
— Hanches droite/gauche (21-17)
— Buste (3)
— Épaules droite/gauche (12-2)

7.4.5.1. Initialisation et récupération des informations de la Kinect

L'initialisation de OpenNI se fait dans un Thread séparé. En effet, une fois lancé, le programme va récupérer en continu des informations depuis la Kinect, et il fallait que cette tâche s'effectue en parallèle pour ne pas bloquer le reste du programme, comme l'affichage ou le Bluetooth.

L'initialisation quand à elle, consiste à expliquer à la librairie OpenNI ce qu'elle doit récupérer, et comment elle doit le faire. Lorsque cela est terminé, le Thread entre dans une boucle infinie. À chaque itération, le programme va chercher un utilisateur dans le champ de vision de la Kinect, le traquer et récupérer les coordonnées des ses articulations.

```
user.leftPart.hip = createJoint(XN_SKEL_LEFT_HIP, user.id);
user.leftPart.knee = createJoint(XN_SKEL_LEFT_KNEE, user.id);
user.leftPart.foot = createJoint(XN_SKEL_LEFT_FOOT, user.id);
user.leftPart.shoulder = createJoint(XN_SKEL_LEFT_SHOULDER, user.id);
user.rightPart.hip = createJoint(XN_SKEL_RIGHT_HIP, user.id);
user.rightPart.knee = createJoint(XN_SKEL_RIGHT_KNEE, user.id);
user.rightPart.foot = createJoint(XN_SKEL_RIGHT_FOOT, user.id);
user.rightPart.shoulder = createJoint(XN_SKEL_RIGHT_SHOULDER, user.id);

user.torsoJoint = createJoint(XN_SKEL_TORSO, user.id);

user.previousLeftPart = previousUser.leftPart;
user.previousRightPart = previousUser.rightPart;

OpenNIUtil::rotationForUser(_frequency, previousUser.rotation, &user);

// Only compute the walk speed if we are not in the first frame
if(!firstLoop)
{
    user.walkSpeed = OpenNIUtil::walkSpeedForUser(_frequency, user, previousUser.timestamp, previousUser.walkSpeed);
    if(previousUser.walkSpeed != -1 && previousUser.walkSpeed <= MIN_COMPUTED_WALKSPEED)
        user.numberOfFramesWithoutMove = previousUser.numberOfFramesWithoutMove + 1;
    else
        user.numberOfFramesWithoutMove = 0;
}
```

FIGURE 7.23.: Boucle de récupération des informations d'OpenNI

Vous pouvez voir sur cette image que nous récupérons seulement les articulations qui nous intéressent, puis, que nous appelons deux fonctions essentielles, l'une pour le calcul de l'orientation et l'autre pour le calcul de la vitesse de marche.

7.4.5.2. Calcul de l'orientation et de la vitesse de marche

Orientation Pour calculer l'orientation du joueur, nous utilisons des relations trigonométriques. En effet, à chaque image, nous sommes dans la configuration suivante (en vue du dessus) :

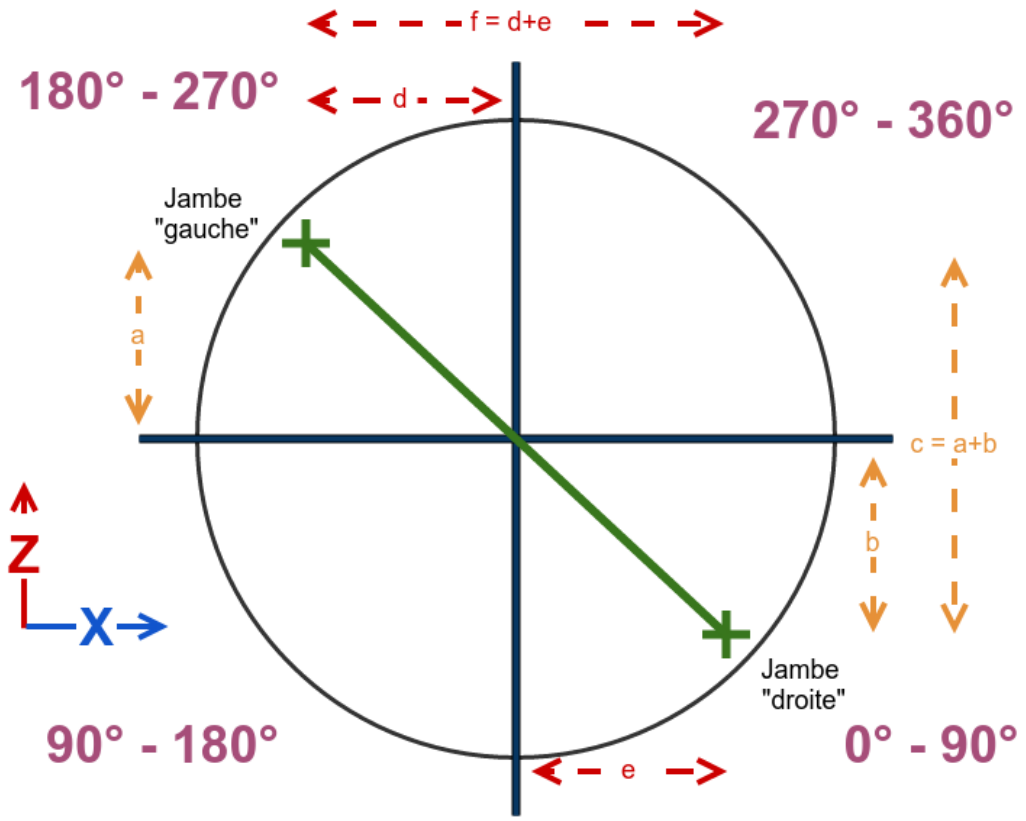


FIGURE 7.24.: Schéma du calcul de l'orientation du joueur

Sur ce schéma vous pouvez voir que l'utilisateur a légèrement pivoté vers l'avant. Pour calculer son angle de rotation, nous utilisons la formule suivante :

$$\arctan\left(\frac{c}{f}\right) = \arctan\left(\frac{b}{e}\right) = \arctan\left(\frac{a}{d}\right)$$

La première est considérée comme plus précise car elle « mélange » la partie droite et la partie gauche de l'utilisateur.

L'équivalent en C++ donne :

```
const float angle = std::atan(std::abs(rightJoint.info.position.Z - leftJoint.info.position.Z)
                              / std::abs(rightJoint.info.position.X - leftJoint.info.position.X)) * RAD2DEG;
```

FIGURE 7.25.: Calcul de l'orientation du joueur

Cependant, ce calcul ne nous donne qu'une valeur entre 0°et 90°. Il nous faut donc la ramener entre 0°et 360° en sachant dans quel quadrant se situe chaque jambe.

Au début, nous n'appliquions cette formule qu'entre les points des hanches. Mais nous nous sommes vite rendus compte que cela n'était pas assez précis. Cette formule est donc maintenant appliquée sur plusieurs ensembles, et une moyenne est faite de l'ensemble des valeurs. Les ensembles étudiés sont :

— Hanche droite / Hanche gauche
— Hanche droite / Buste
— Buste / Hanche gauche
— Épaule droite / Épaule gauche

Vitesse de marche La vitesse de marche est l'autre valeur importante calculée par notre programme. Ce calcul est beaucoup plus simple que le précédent, car il utilise de simples relations dans le triangle rectangle :

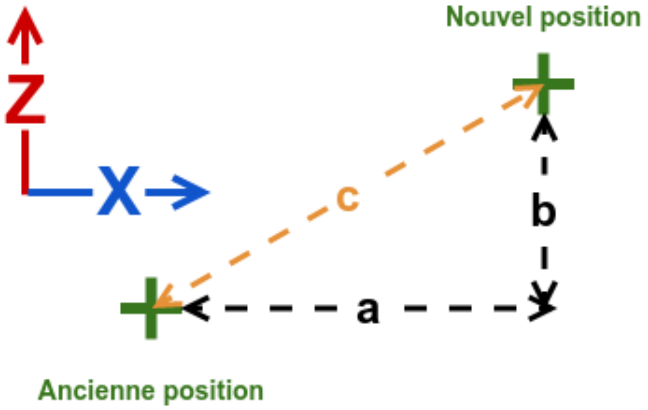


FIGURE 7.26.: Schéma du calcul de la vitesse de marche du joueur

Sur ce schéma, la distance qui nous intéresse est la distance c, qui représente la distance parcourue par le pied du joueur. Pour la calculer, il suffit de faire :

$$c = \sqrt{a^2 + b^2}$$

Cette opération est effectuée pour chaque pieds, puis une moyenne est faite. En C++, cela donne :

```
const float rightDiff = std::sqrt(std::pow(rdx, 2.0) + std::pow(rdz, 2.0));
const float leftDiff = std::sqrt(std::pow(ldx, 2.0) + std::pow(ldz, 2.0));

// Compute the average of diff (in mm)
const float diff = (rightDiff + leftDiff) / 2.0;
```

FIGURE 7.27.: Calcul de la distance des pieds entre deux frames

Une fois que nous avons cette distance, il suffit de la diviser par le temps écoulé entre les deux images, et nous avons notre vitesse de marche.

7.4.5.3. Affichage des informations

Maintenant que nous avons vu en détails comment calculer les informations nécessaires, nous pouvons voir comment les afficher. Ainsi, le module d'affichage va récupérer les informations depuis le Thread dédié à *OpenNI* à une certaine fréquence. Cette fréquence est la même que celle d'envoi des données Bluetooth.

Pour pouvoir réaliser cet affichage, nous utilisons la librairie *OpenCV* qui est spécialisée dans le traitement et la modification des images.

Voici une capture d'écran du logiciel lorsque personne n'est visible par la Kinect :

FIGURE 7.28.: Module d'affichage de la Kinect

Cet écran est divisé en deux parties : à gauche, le retour vidéo et à droite, l'affichage des informations. On peut aisément deviner que la rotation sera affichée à l'intérieur du cercle et que la vitesse de marche sera affichée au dessus du curseur, en bas à droite.

Lorsque qu'un utilisateur entre dans le champ de vision, son squelette s'affiche :

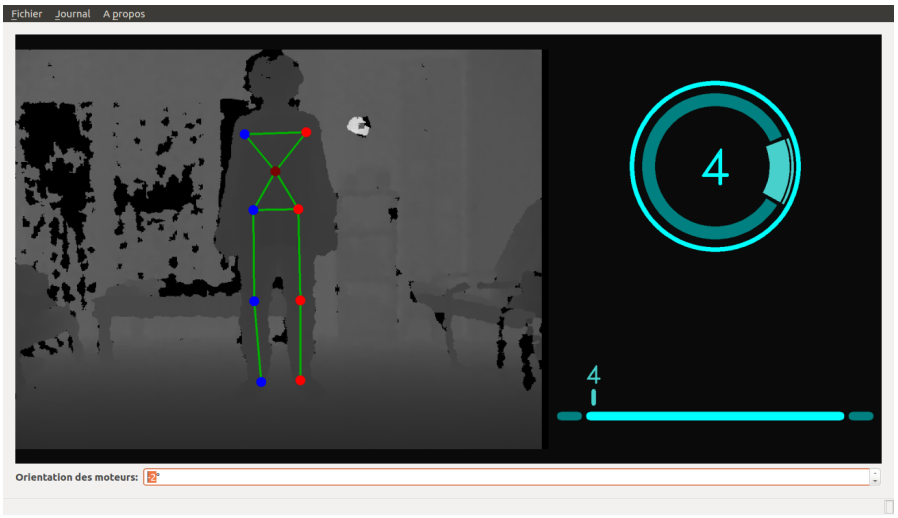


FIGURE 7.29.: Module d'affichage de la Kinect avec joueur

On peut également observer que sa rotation est quasi-nulle (ce qui est normal puique le joueur est de face), tout comme sa vitesse. Lorsque que le joueur se tourne, sa rotation est mise à jour :

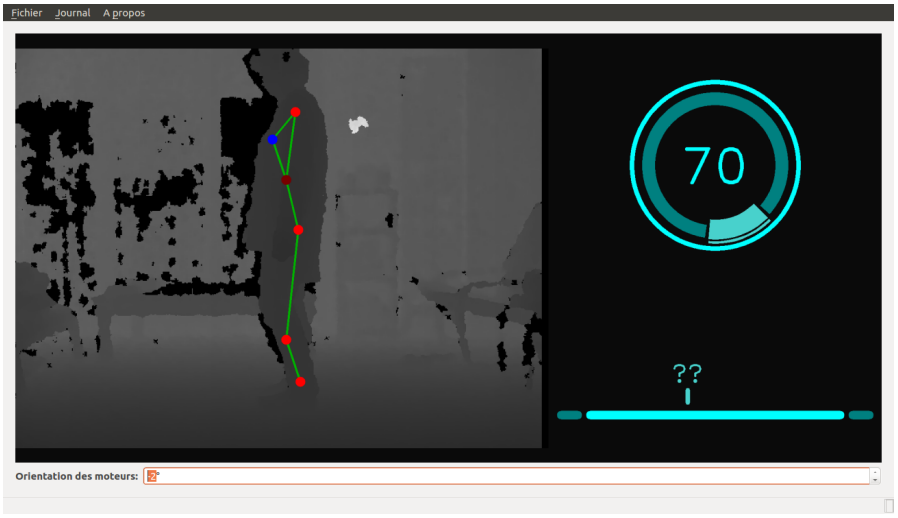


FIGURE 7.30.: Module d'affichage de la Kinect avec joueur de profil

Tout comme sa vitesse de marche lorsque qu'il commence à courir :

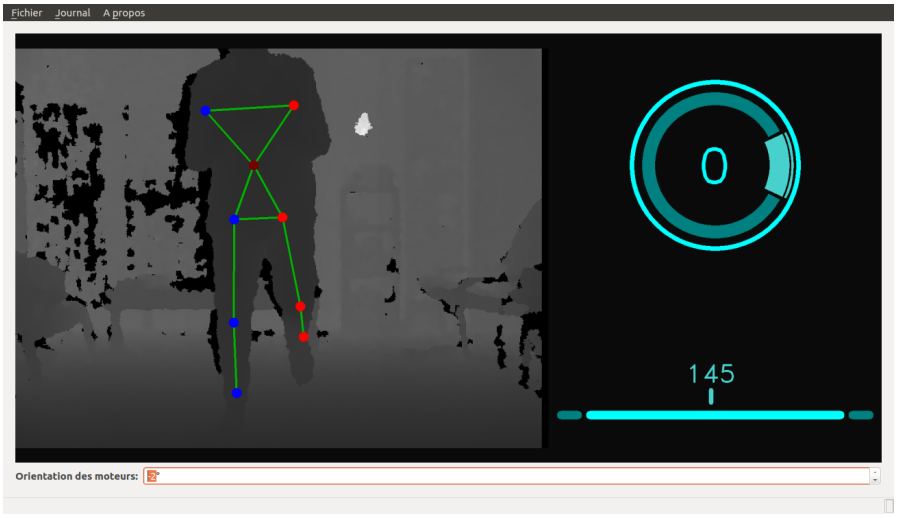


FIGURE 7.31.: Module d'affichage de la Kinect avec joueur qui court

8. Application pour smartphone (RandCity)

FIGURE 8.1.: Logo du l'application RandCity

8.1. Objectifs et cahier des charges

Afin de permettre l'immersion visuelle du joueur, le jeu qui seras lancé sur le téléphone contenu dans le casque devra afficher un environnement 3D. Il devra aussi séparer l'écran en deux afin que l'utilisateur voit l'image en 3 dimensions.

De plus, cette application doit pouvoir être exécutée dans un environnement *Android*. Si nous avons choisi Android plutôt qu'*iOS*, ce n'est seulement que parce que l'ensemble des membres de notre projet possédait des téléphone Android. Nous avons donc utilisé la seule plate-forme à notre disposition.

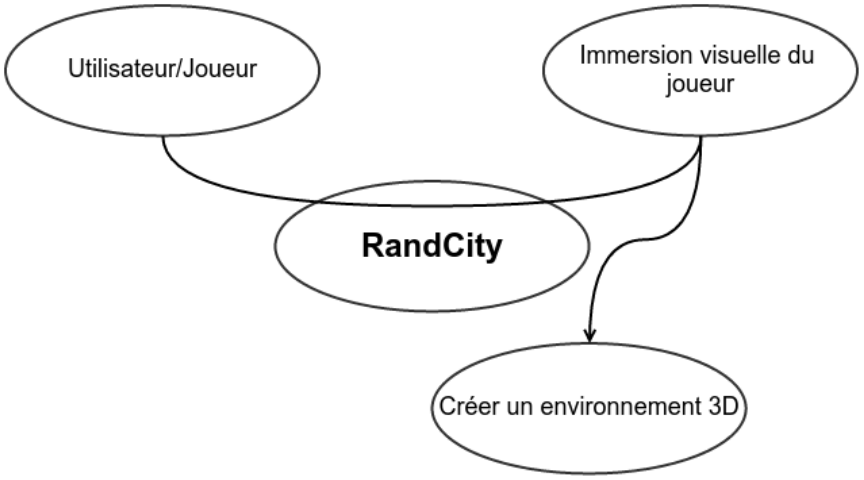


FIGURE 8.2.: Diagramme « Bête à cornes » de l'application RandCity

De plus, nous avons décidé que l'environnement 3D que devait générer l'application serait un environnement urbain. En effet, aucun de nos membres n'ayant de réelles compétences en infographie, l'environnement urbain nous a permis de rester plus sobres quand au design des bâtiments.

8.1.1. Diagramme FAST

Le choix des solutions à utiliser était beaucoup plus limité que pour la création du logiciel VRController. Nous avons donc pu directement créer le diagramme FAST suivant :

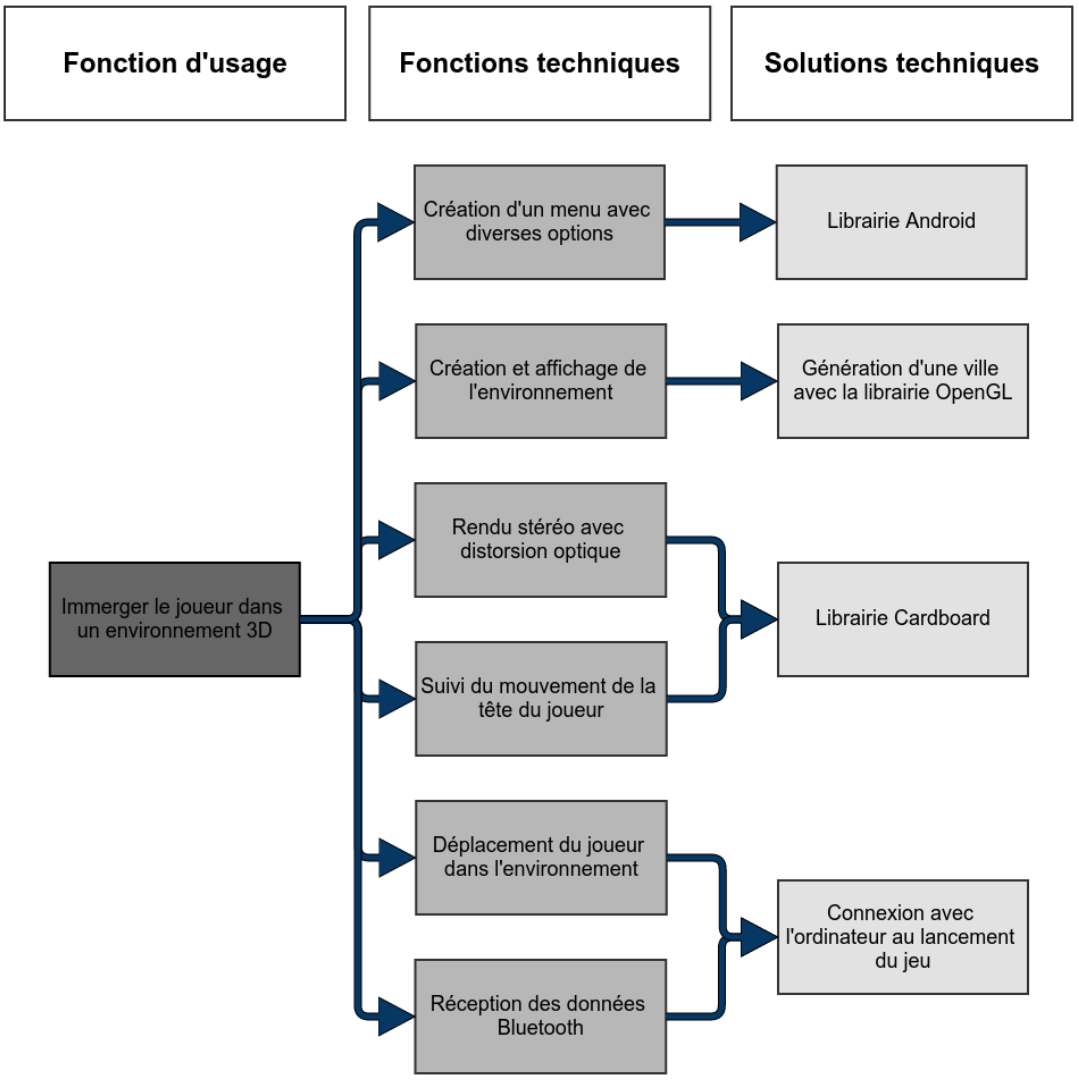


FIGURE 8.3.: Diagramme FAST de l'application RandCity

8.2. Présentation des outils utilisés

Pour pouvoir réaliser l'application RandCity, nous avons donc eu recours à diverses librairies externes :

— **Bibliothèque Android :** obligatoire pour programmer sur Android. Donne accès à un ensemble d'éléments pour créer une application, que ce soit au niveau graphique ou interne. Fournit également un logiciel adapté à la programmation Android.
— **OpenGL ES 2 :** Sous ensemble de la bibliothèque Android, OpenGL est spécialisé dans l'imagerie en 3 dimensions. OpenGL ne fournit que des fonctions de bases pour l'affichage de formes simples à l'écran. Cette librairie est aujourd'hui utilisée par la majorité des jeux, qu'ils soient sur mobile, ordinateur ou console.
— **Librairie Cardboard :** Nous avons vu précédemment ce qu'était le Cardboard. En même temps que son masque, Google a aussi publié une librairie Android qui rend la Réalité Virtuelle accessible.

8.2.1. Bibliothèque Android

La bibliothèque Android est un ensemble d'outils qui permettent la création d'applications et la programmation pour cette plate-forme. Deux outils essentiels composent cette bibliothèque :

— **Le logiciel Android Studio :** l'IDE recommandé par Google (Integrated Development Environement). Il permet de facilement programmer pour cette plate-forme, permet d'avoir le retour de tout ce qui se passe sur le téléphone, et permet même de l'émuler si vous n'en avez pas.

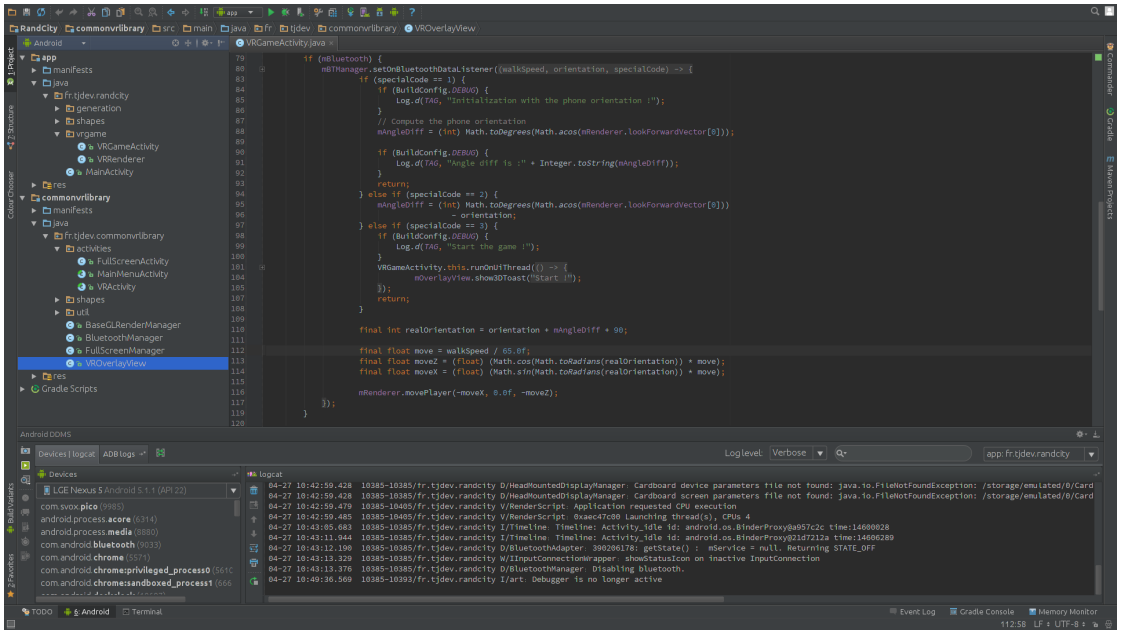


FIGURE 8.4.: Capture d'écran du logiciel Android Studio

— **L'API Android :** ensemble de fonctions, d'objets, d'utilitaires et de concepts permettant de créer une application.

La programmation Android se fait à l'aide du langage de programmation Java, mais la structure du programme est très différente. En effet, sous Android, il y a le concept des « Activités ». Une Activité peut être vue comme un « écran »ou une « fenêtre » : lorsque vous lancez une application, vous apparaissez sur l'activité principale. Si vous appuyez sur un bouton, une nouvelle activité est lancée et va afficher la météo par exemple.

Chaque activité possède la même structure, avec certaines fonctions qui sont appelées à un moment bien précis :

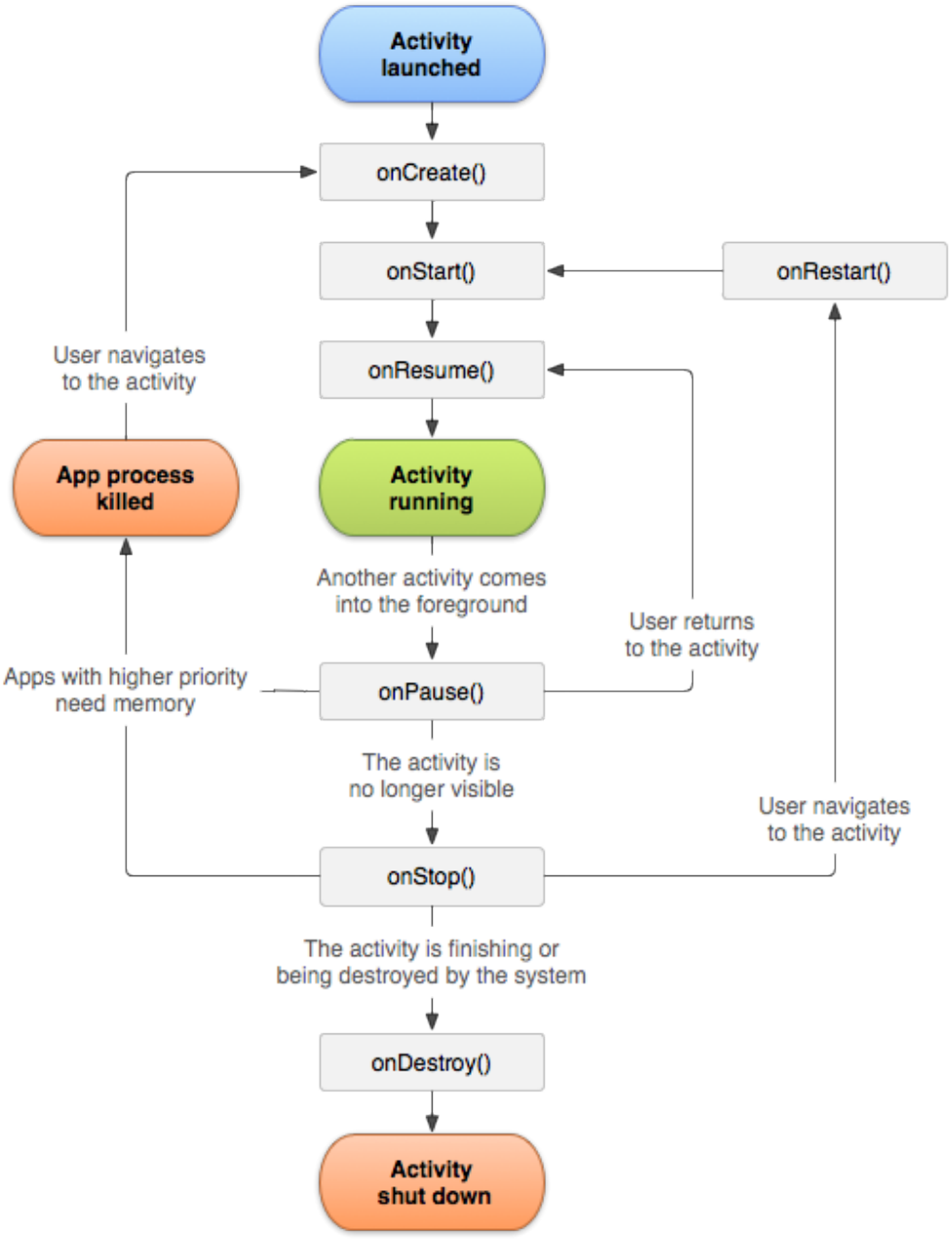


FIGURE 8.5.: Schéma d'une activité Android

8.2.2. OpenGL ES 2

OpenGL ES 2 n'est pas vraiment une librairie, mais plutôt un standard, que les autres librairies doivent supporter. Ainsi la bibliothèque Android supporte ce standard dans une « Activité »spéciale.

OpenGL permet l'affichage d'un environnement 3D. La version *ES 2* est juste une variante destinée aux systèmes embarqués comme les téléphones. Certaines fonctionnalités poussées ont été enlevées, mais cela ne nous a pas affectés à notre niveau.

On pourrait qualifier OpenGL d'une librairie bas niveau, au sens où elle se rapproche du strict minimum. Par exemple, OpenGL ne connaît que les concepts de points dans l'espace et de triangles. Si vous voulez afficher un cube, il vous faut créer 2 triangles par face, soit 12 en tout.
Elle ne connaît pas non plus le principe de « collisions ». Si vous ne faites rien, vous pouvez très bien passer à travers les murs.

OpenGL n'est donc pas un moteur de jeu, comme *Unity* ou *Unreal Engine*, mais simplement un outil pour afficher des formes en 3 dimensions. Sachez également que les moteurs comme *Unity* se basent sur OpenGL.

FIGURE 8.6.: Logo d'OpenGL ES 2

8.2.2.1. Différents espaces de coordonnées

Lorsque vous voulez afficher quelque chose avec OpenGL, il faut faire attention aux différents espaces de coordonnées.

Prenons l'exemple d'un cube. Lorsque vous définissez sa forme, vous devez définir chaque triangle de chaque face (2 par face) à l'aide de 3 points (de coordonnées $(x;y;z)$). Vous avez donc 36 points à spécifier (8 faces de 2 triangles de 3 points chacun).

```
static public final float[] positionData = {
        // Front face
        -1.0f, 1.0f, 1.0f,
        -1.0f, -1.0f, 1.0f,
        1.0f, 1.0f, 1.0f,
        -1.0f, -1.0f, 1.0f,
        1.0f, -1.0f, 1.0f,
        1.0f, 1.0f, 1.0f,

        // Right face
        1.0f, 1.0f, 1.0f,
        1.0f, -1.0f, 1.0f,
        1.0f, 1.0f, -1.0f,
        1.0f, -1.0f, 1.0f,
        1.0f, -1.0f, -1.0f,
        1.0f, 1.0f, -1.0f,

        // Back face
        1.0f, 1.0f, -1.0f,
        1.0f, -1.0f, -1.0f,
        -1.0f, 1.0f, -1.0f,
        1.0f, -1.0f, -1.0f,
        -1.0f, -1.0f, -1.0f,
        -1.0f, 1.0f, -1.0f,

        // Left face
        -1.0f, 1.0f, -1.0f,
        -1.0f, -1.0f, -1.0f,
        -1.0f, 1.0f, 1.0f,
        -1.0f, -1.0f, -1.0f,
        -1.0f, -1.0f, 1.0f,
        -1.0f, 1.0f, 1.0f,

        // Top face
        -1.0f, 1.0f, -1.0f,
        -1.0f, 1.0f, 1.0f,
        1.0f, 1.0f, -1.0f,
        -1.0f, 1.0f, 1.0f,
        1.0f, 1.0f, 1.0f,
        1.0f, 1.0f, -1.0f,

        // Bottom face
        1.0f, -1.0f, -1.0f,
        1.0f, -1.0f, 1.0f,
        -1.0f, -1.0f, -1.0f,
        1.0f, -1.0f, 1.0f,
        -1.0f, -1.0f, 1.0f,
        -1.0f, -1.0f, -1.0f
};
```

FIGURE 8.7.: Définition d'un cube avec OpenGL

Sur cette image, on a créé un cube de largeur 2, centré sur le point $(0;0;0)$. Nous avons donc utilisé **l'espace de coordonnées du modèle**. Si nous voulons déplacer ce cube, nous n'allons pas modifier ces valeurs, mais nous allons déplacer ce cube à l'intérieur de **l'espace de coordonnées du monde**.

L'espace du monde contient l'ensemble des éléments de notre environnement. Mais lorsque nous regardons la scène, nous ne pouvons pas tout voir, ce qui est derrière nous, par exemple, est caché. L'espace que nous voyons est **l'espace de coordonnées de la vue**. OpenGL applique une dernière modification, afin de projeter les éléments d'un espace 3D vers un espace 2D (l'écran) : ce dernier s'appelle **l'espace de coordonnées de projection**.

Quand nous regardons l'écran, nous voyons donc l'espace de projection. Les méthodes utilisées pour transformer les deux derniers espaces sont extrêmement compliquées, et OpenGL les prend en charge.

Au niveau des espaces, nous ne soucions que de l'espace du modèle et de l'espace du monde.

8.2.2.2. Système de matrices

Pour pouvoir gérer les différentes informations que nous lui envoyons, OpenGL doit utiliser un système standard afin d'appliquer tous les calculs dont il a besoin.

En temps normal, lorsque nous écrivons les coordonnées d'un point, nous utilisons la notation : $(x;y;z)$. Mais pour avoir un système uniforme, OpenGL utilise exclusivement la notation matricielle. Ainsi, les coordonnées d'un point deviennent :

$$P = \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

La variable w peut prendre deux valeurs : **1** si ce sont les coordonnées d'un point et **0** si c'est un vecteur.

Ce quatrième élément est essentiel lorsque OpenGL va appliquer des transformations. Ces transformations, c'est-à-dire lorsque l'on va modifier les coordonnées d'un point, sont toutes exprimées à l'aide d'une matrice carrée de taille 4 :

$$T = \begin{pmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{pmatrix}$$

Pour pouvoir appliquer cette transformation , OpenGL va tout simplement multiplier la matrice T avec celle du point P :

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{pmatrix} \times \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} m_0 \cdot x + m_4 \cdot y + m_8 \cdot z + m_{12} \cdot w \\ m_1 \cdot x + m_5 \cdot y + m_9 \cdot z + m_{13} \cdot w \\ m_2 \cdot x + m_6 \cdot y + m_{10} \cdot z + m_{14} \cdot w \\ m_3 \cdot x + m_7 \cdot y + m_{11} \cdot z + m_{15} \cdot w \end{pmatrix}$$

Pour expliquer on peut dire la chose suivante :

— m_0 est la proportion de x dans x'
— m_1 est la proportion de x dans y'
— m_2 est la proportion de x dans z'
— ...
— m_9 est la proportion de z dans y'
— m_{10} est la proportion de z dans z'
— ...

Cela peut ne pas sembler évident, mais c'est essentiel de le comprendre pour comprendre le fonctionnement d'OpenGL. En pratique, la matrice T n'est jamais utilisée telle-quelle car il existe plusieurs formes particulières, selon que l'on veuille faire une translation, une mise à l'échelle ou une rotation.

Ces formes particulières sont les suivantes :

— Translation de paramètres $(x;y;z)$:

$$T = \begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

— Redimensionnement de paramètres $(a;b;c)$:

$$T = \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

— Rotation d'angle θ autour de l'axe X :

$$T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

— Rotation d'angle θ autour de l'axe Y :

$$T = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

— Rotation d'angle θ autour de l'axe Z :

$$T = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

8.2.3. Librairie Cardboard

La librairie Cardboard est une librairie permettant de transformer une application affichant un environnement 3D, en une application prête pour la réalité virtuelle. En effet cette librairie permet de diviser automatiquement l'écran en deux parties (une partie pour chaque œil) et d'y appliquer les bonnes transformations.

Une fois encore, le système permettant de réaliser cette division n'était pas à notre portée, nous avons donc décider d'utiliser une librairie existante.

FIGURE 8.8.: Application d'exemple utilisant la librairie Cardboard

Cette librairie permet également de gérer l'accéléromètre et le gyroscope du téléphone. Cela signifie que si vous levez le téléphone, vous allez aussi lever la tête dans l'environnement 3D. Celui-ci s'adapte donc à tous les mouvements du joueur.

8.3. Présentation technique de l'Application

Après avoir présenté l'ensemble des éléments tiers que nous allons utiliser, passons maintenant à une description plus technique et détaillée de notre projet.

À l'instar de VRController, nous avons également réalisé un calcul du nombre de lignes de codes : nous avons obtenu le résultat de **2641 lignes de code**.

8.3.1. Menu principal

Pour la création du Menu principal, nous avons utilisé le concept d'activité de l'API Android. Nous avons voulu rester relativement sobres au niveau de son agencement.

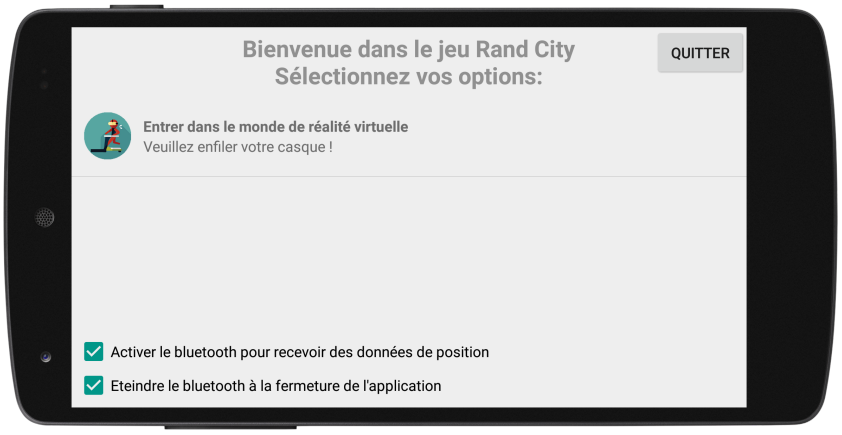


FIGURE 8.9.: Menu principal de RandCity

Vous pouvez voir qu'il y a une option pour activer le Bluetooth. Cependant, si vous la laissez décochée, le jeu perd de son intérêt puisque vous ne pouvez plus vous déplacer dans la ville.

Il existe une autre version du menu, utilisée lors des phases de débogages. IL permet de lancer un environnement de jeu utile pour les tests. Cet environnement sera plus détaillé par la suite.

FIGURE 8.10.: Menu principal de RandCity (mode Debug)

8.3.2. Recherche du serveur Bluetooth

Dés que vous cliquez sur l'un des modes de jeu du menu principal, un nouveau Thread est lancé pour pouvoir accueillir le gestionnaire Bluetooth.

Lors de son lancement, le gestionnaire va vérifier si le Bluetooth est bien activé. Si ce n'est pas le cas, il va demander à l'utilisateur de le faire :

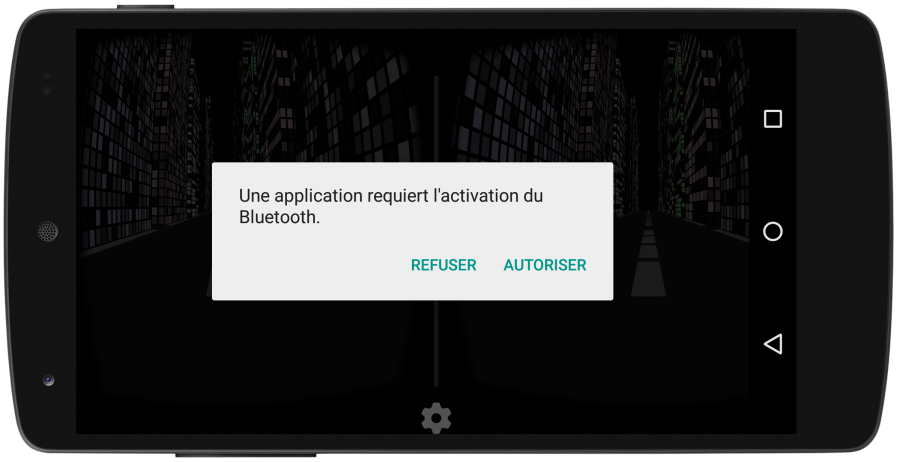


FIGURE 8.11.: Demande d'activation du Bluetooth

Le morceau de code qui réalise cette opération est le suivant :

```
// Enable bluetooth
if (!mBluetoothAdapter.isEnabled()) {
    Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    mParentActivity.startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
    Log.d(TAG, "Enabling Bluetooth ...");
} else {
    // The bluetooth is already on.
    Log.d(TAG, "Bluetooth is already enabled.");
    searchForDevices();
}
```

FIGURE 8.12.: Morceau de code pour la demande d'activation du Bluetooth

Une fois le Bluetooth activé, l'application va commencer à rechercher les différents serveurs (applications VRController) à portée de communication. Pour les sélectionner, elle utilise deux éléments :

— **L'UUID :** un ID unique que seules les applications VRController et RandCity connaissent. Il sert à vérifier que l'on se connecte à la bonne application.
— **L'Adresse MAC :** c'est l'adresse unique de la carte Bluetooth de l'ordinateur qui possède VRController. Elle est stockée dans un fichier texte. L'application n'essaiera de se connecter au serveur que si l'adresse MAC correspond.

A chaque fois qu'un périphérique est trouvé, RandCity va essayer de s'y connecter si l'adresse MAC est la même que celle définie. Cette étape est réalisée par la ligne suivante :

```
socket = device.createInsecureRfcommSocketToServiceRecord(mUUID);
```

FIGURE 8.13.: Tentative de connexion au serveur Bluetooth

Une fois connecté, si aucune erreur ne s'est produite, le gestionnaire va entrer dans une boucle de lecture infinie, afin de recevoir les données en provenance de VRController.

8.3.3. Environnement de jeu

Si vous avez activé le Bluetooth, vous devez attendre que le jeu soit connecté. Une fois fait, vous pouvez profiter pleinement de l'environnement de jeu.

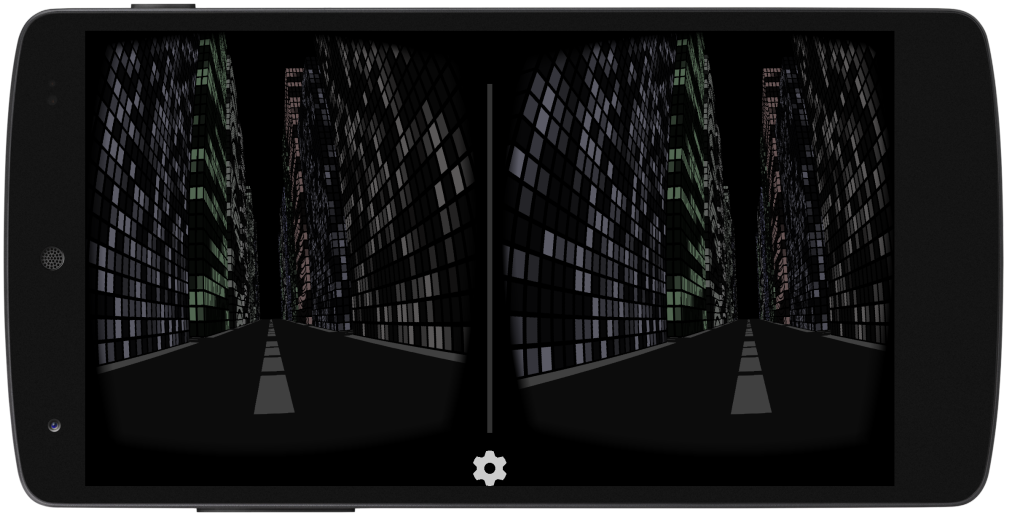


FIGURE 8.14.: Environnement de RandCity (VR)

Vous pouvez voir ici que vous êtes sur une route entouré d'immeubles ayant des façades toutes différentes à l'allure moderne.

Si vous avez activé le mode « Debug », vous n'aurez pas l'effet de double vision et vous aurez un petit bouton vous permettant d'avancer. Ce mode permet de tester les mécaniques du jeu. Certaines captures d'écran seront réalisées dans ce mode pour une meilleure visibilité.

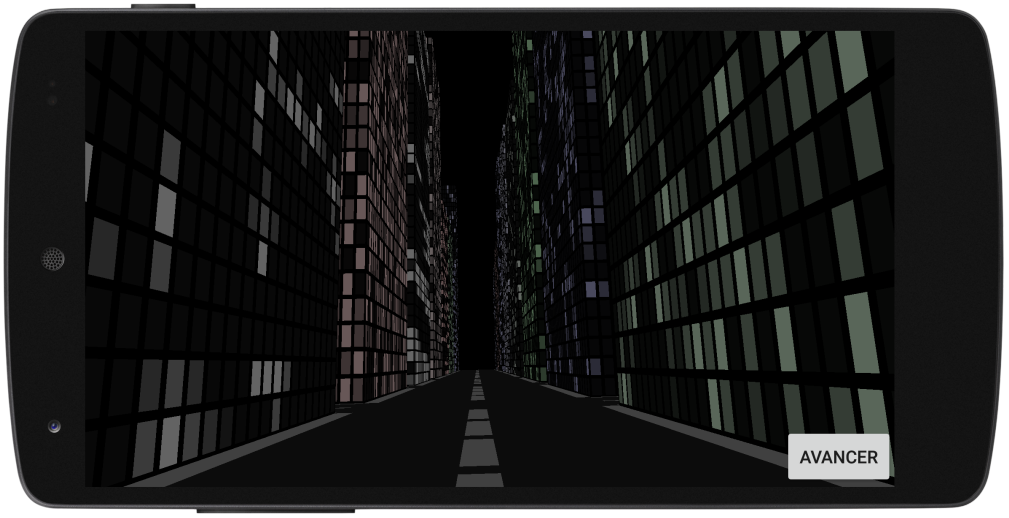


FIGURE 8.15.: Environnement de RandCity

Ce mode a aussi été le premier à être réalisé. En effet, lors des prémices de RandCity, la vue stéréo n'existait pas encore et nous affichions tout un tas d'informations à l'écran. Les déplacements se faisaient exclusivement à l'aide de boutons, même pour la rotation de la tête.

FIGURE 8.16.: Première version de RandCity

8.3.3.1. Génération aléatoire

Comme vous le savez, la ville que vous pouvez explorer dans RandCity est générée aléatoirement. Voici les divers éléments issus de facteurs aléatoires :

Le trésor Ce que nous appelons le trésor possède une position aléatoire dans la ville. Si vous le trouvez vous verrez des escaliers descendre suivi d'un long tunnel de plus en plus sombre. En réalité, il n'y a rien au bout, mais si vous vous avancez suffisamment loin, un message s'affichera disant que vous avez trouvé la sortie et avez fini le jeu.

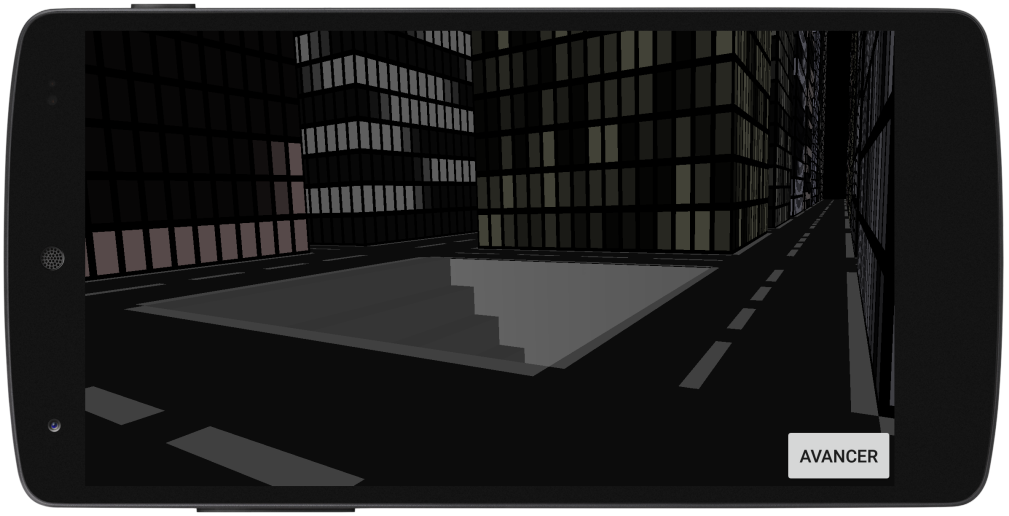


FIGURE 8.17.: Escaliers du « Trésor »

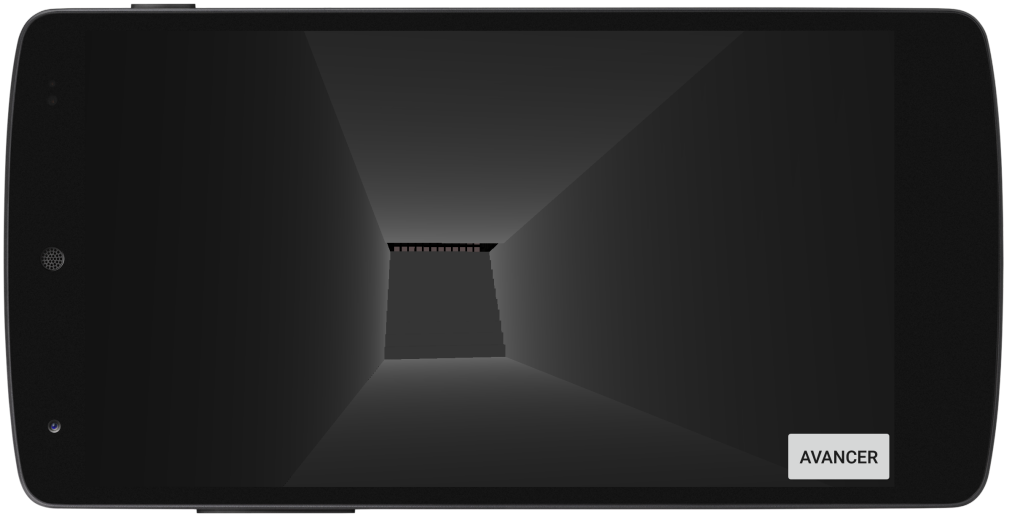


FIGURE 8.18.: Couloir du « Trésor »

Les bâtiments Pour les bâtiments, plusieurs éléments sont aléatoires. Tout d'abord, la hauteur et leur couleur. Pour ces éléments, rien de plus simple, il suffit de tirer quatre nombres au hasard (1 pour la hauteur et 3 pour la couleur). Une fonction existe dans la librairie Android et nous l'utilisons :

```
build.height = (float)rand.intBetween(GenUtil.BUILD_MIN_HEIGHT, GenUtil.BUILD_MAX_HEIGHT);
build.color = new float[] {
        rand.intBetween(0, 100) / 100.0f,
        rand.intBetween(0, 100) / 100.0f,
        rand.intBetween(0, 100) / 100.0f,
        1.0f
};
```

FIGURE 8.19.: Génération aléatoire de la couleur et de la hauteur des bâtiments

Mais l'élément le plus visible est sûrement les textures des bâtiments. En effet, vous avez sûrement remarqué que les fenêtres ne sont pas toujours placées au même endroit et ont des formes différentes. Pour générer les fenêtres, nous utilisons deux algorithmes distincts.

Le premier peut se résumer avec ce schéma où chaque rectangle noir correspond à une étape :

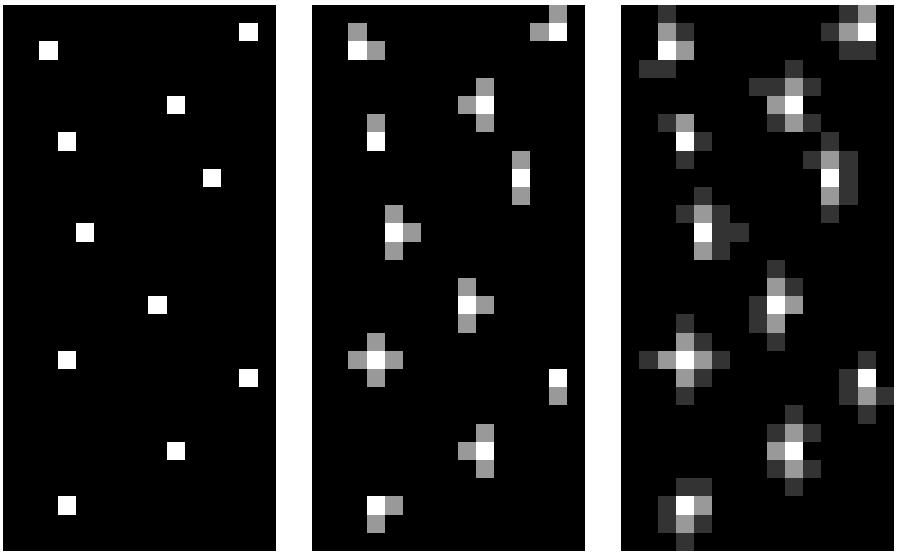


FIGURE 8.20.: Génération aléatoire des textures (1)

Au départ, on ajoute quelques fenêtres aléatoirement sur la texture. Puis, à l'étape d'après, on ajoute un dégradé sur les cases voisines, et ainsi de suite. Pour un meilleur rendu, nous répétons cette étape 5 fois. Bien sûr, le taux d'apparition des fenêtres ainsi que celui des dégradés est lui même aléatoire.

Le deuxième algorithme est sûrement plus réaliste. Le voici :

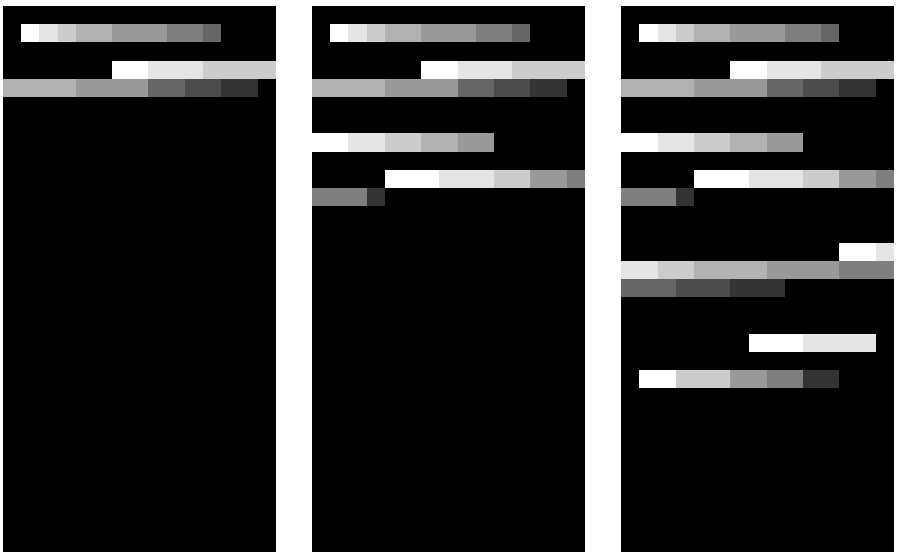


FIGURE 8.21.: Génération aléatoire des textures (2)

Dans cet algorithme, chaque étape correspond à une rangé éclairée. En effet, à chaque case, il y a une certaines chance que la fenêtre devienne blanche. Si c'est le cas, il y a à nouveau une certaine chance, mais plus petite, qu'elle se répète ou qu'elle se dégrade légèrement. Et ainsi de suite. Cela permet de donner cet effet de bandes de plus en plus sombre.

8.3.3.2. Effet de brouillard

Dans un souci d'immersion, nous avons voulu donner un côté sombre à notre jeu. Pour ce faire, nous avons ajouté un effet de brouillard :

FIGURE 8.22.: Effet de brouillard

Afin de pouvoir avoir ce rendu, nous avons étudié les 3 fonctions principalement utilisées pour créer le brouillard :

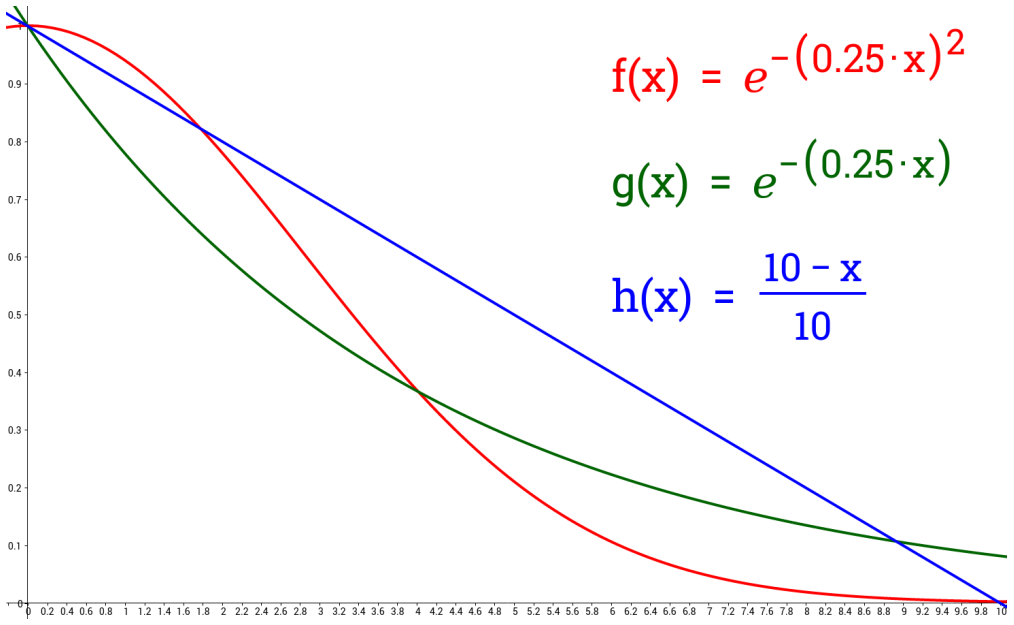


FIGURE 8.23.: Fonctions de l'effet de brouillard

Sur ce graphique, on peut observer le pourcentage de la couleur originale en fonction de la distance avec le joueur. Nous avons donc directement exclu la fonction linéaire $h(x)$ car peu réaliste. Nous avons finalement choisi la fonction $f(x)$ car elle permet de n'avoir aucun brouillard proche du joueur, mais descend à la fois très vite avec la distance.

La formule du brouillard que nous utilisons est donc :

$$fog = \exp(-(densite * z)^2)$$

or :

$$\exp x = 2^{\frac{x}{\log 2}} = 2^{x \cdot \frac{1}{\log 2}} \approx 2^{1.442695 \cdot x}$$

On obtient donc la formule :

$$fog = 2^{(-1.442695 \cdot densite^2 \cdot z^2)}$$

Cette formule est finalement appliquée pour chaque pixel de l'écran avant son envoi vers OpenGL.

8.3.4. Déplacements dans l'environnement

Maintenant, revenons à la partie Bluetooth. A chaque fois que des données sont envoyées par le logiciel de détection, le jeu doit immédiatement les appliquer à l'environnement 3D.

```
final int realOrientation = orientation + mAngleDiff;

final float move = walkSpeed / 65.0f;
final float moveZ = (float) (Math.cos(Math.toRadians(realOrientation)) * move);
final float moveX = (float) (Math.sin(Math.toRadians(realOrientation)) * move);

mRenderer.movePlayer(-moveX, 0.0f, -moveZ);
```

FIGURE 8.24.: Application de la vitesse et de l'orientation sur le joueur

Sur ce morceau de code, l'orientation est redéfinie en fonction du décalage qu'il existe entre le jeu et la Kinect (défini lors de l'envoi du code spécial 1). Puis, on applique la vitesse de marche sur les deux composantes X et Z avec de simples fonctions trigonométriques. Si vous vous demandez pourquoi on divise la vitesse par 65, c'est simplement que nous nous sommes rendus compte, que si nous ne le faisions pas, la vitesse de déplacement dans la ville était trop élevée et donc non-réaliste.

9. Distribution sous licence OpenSource

Dés le début, nous avons voulu partager notre projet avec la communauté, que ce soit celle des joueurs ou celle de l'informatique. Ainsi nous avons décidé de partager l'ensemble de notre code sous une licence libre, une licence OpenSource.

Qu'est-ce qu'une license OpenSource

Tout d'abord, une licence est un texte juridique qui protège l'auteur et son œuvre avec différents critères fixés par ce même texte. En informatique, une licence est dite OpenSource si elle permet à n'importe qui d'utiliser, de modifier ou d'intégrer un programme (ou une partie de ce programme) dans une œuvre plus large.

Cependant, la personne qui réutilise le code ne peut pas faire ce qu'elle veut avec, et doit respecter les termes de la licence.

Dans notre cas, nous avons choisi de distribuer le code sous licence *GNU GPL v3*. Cette licence permet à quiconque de réutiliser tout ou partie du programme tant que les conditions suivantes sont respectées :

— Le code source doit toujours être mis a disposition du public
— Le code modifié doit être lui aussi distribué sous licence *GNU GPL v3*
— Tout logiciel intégrant du code sous licence *GPL* doit être distribué sous la même licence
— L'auteur original doit toujours être cité

Dans cette optique de distribution OpenSource, nous avons hébergé notre code sur un site spécialisé dans le partage de code source appelé *GitHub*. Le code est donc disponible à l'adresse `https://github.com/VirtualWalker`.

Vous avez également remarqué que l'ensemble du code a été écrit en Anglais. Comme décrit en introduction de la partie informatique, ceci est dû à la prédominance de l'anglais dans le domaine de l'informatique. Des gros projets même créés par des français (comme le moteur de jeu *SFML*) sont écrits en anglais.

Cinquième partie .

Conclusion

Le projet VirtualWalker, que nous avons réalisé au cours du PPE de Terminale, était très ambitieux. Dés le début nous avons eu une multitude d'idées afin d'arriver à notre objectif : *Améliorer l'immersion procurée par la réalité virtuelle*.

Ce projet nous a permis d'explorer plusieurs facettes de l'ingénierie, en allant de la mécanique à l'informatique, en passant par la physique et l'optique. Il nous a également permis d'aborder les contraintes d'un projet en groupe, et la nécessité de définir précisément le cahier des charges et les diagrammes divers (Bête à cornes, SADT, FAST)

Aujourd'hui, nous pouvons considéré que le projet est terminé et que nous avons réalisé un prototype plus que fonctionnel (le même prototype que nous avons amené aux Olympiades de SI). Nous restons tout de même conscients que des améliorations auraient été possibles avec un peu plus de temps et plus de connaissances préalables.

Cependant, même terminé, notre projet conserve toujours son fort potentiel que nous avons expliqué en introduction de ce rapport. Dans cet objectif, et afin que de futures personnes puissent profiter de notre travail, nous avons mis en ligne un site Internet ayant pour but de présenter notre projet.
Celui-ci est disponible à l'adresse `http://virtualwalker.github.io`.

A. Synthèses personnelles

A.1. Fabien CAYLUS

Quelle a été ma place dans ce Projet ?

Dans ce projet, j'étais la personne chargée de toute la partie informatique, ayant déjà des connaissances préalables dans ce domaine. J'ai donc réalisé les deux programmes importants : *VRController* et *RandCity*.

Qu'est ce que je pense de ce Projet ? Et des conditions dans lesquelles il s'est déroulé ?

A titre personnel, je suis plus que satisfait de ce que nous avons réalisé. Je dois avouer qu'au début j'étais sceptique sur le fait que nous allions y arriver, surtout sur la partie informatique. Mais nous avons réussi. Nous avons du collaborer très étroitement tous ensemble. Les professeurs, surtout M. Desautel et M. Sahakian, nous ont souvent conseillé quant à la démarche à suivre, et cela nous a grandement aidé.

Qu'ai-je appris lors de ce Projet ?

Lors de ce projet, je pense avoir appris à collaborer et à m'intégrer dans une équipe. J'ai aussi découvert la démarche d'ingénieur et la rigueur qui nous est demandée. J'ai également appris beaucoup de choses dans le domaine de l'informatique, et maîtrisé des bibliothèques que je ne connaissais pas encore il y a quelques mois.

A.2. Thibault GRIMALDI

Quelle a été ma place dans ce Projet ?

Ce projet m'a beaucoup plu, je me suis donc beaucoup investi dedans. J'ai donc eu une des quatre parties à réaliser, celle du support qui comportait plusieurs choses :

— Recherche d'une solution technique
— Modélisation et simulation via un outil de *CAO*
— Réalisation
— Recherche d'un agent glissant

Qu'est ce que je pense de ce Projet ? Et des conditions dans lesquelles il s'est déroulé ?

Je pense que ce que nous avons produit est très satisfaisant. En effet nous sommes arrivés à un projet qui fonctionne et qui semble plaire aux utilisateurs. J'ai beaucoup aimé le travail en équipe mais aussi la présence des deux professeurs chargés de nous encadrer, qui répondaient à nos questions et possédaient beaucoup d'humour. On peut dire dans l'ensemble que ce Projet s'est très bien déroulé.

Qu'ai-je appris lors de ce Projet ?

Ce projet m'a appris de nouvelles choses dans toutes les disciplines abordées. Celui-ci m'a aussi ouvert les yeux sur les manières dont procèdent les grandes industries lors de la conception d'un nouveau produit. Mais il m'a surtout appris à mieux travailler en équipe.

A.3. Taki HAZAM

Quelle a été ma place dans ce Projet ?

Comme les autres, j'ai tout de suite été content de participer à ce projet. Initialement, je devais travailler sur la partie informatique, et plus particulièrement sur le logiciel de détection de mouvements.

Qu'est ce que je pense de ce Projet ? Et des conditions dans lesquelles il s'est déroulé ?

Je pense que ce projet était complexe mais réalisable pour notre équipe de 4 personnes.

Personnellement, j'ai eu beaucoup de difficulté à réaliser la partie qui m'était destinée car je n'avais aucune connaissance préalable en informatique et en programmation. J'ai donc du me résoudre à laisser la main à Fabien, qui avait de meilleures capacités que moi en programmation.

Qu'ai-je appris lors de ce Projet ?

Malgré tout, j'ai quand même appris à utiliser le langage de programmation *C++*, qui est un langage bien plus complexe que le *Python* ou le *HTML* que nous apprenons en cours d'ISN. J'ai aussi appris le fonctionnement de la Kinect et à utiliser son retour vidéo par exemple.

De manière plus générale, j'ai pu réaliser, à travers ce projet, l'importance de chaque membre d'un groupe, et leur implication dans le résultat final.

A.4. Mathieu MERTINY

Quelle a été ma place dans ce Projet ?

J'ai beaucoup aimé ce projet car il traite des sujets qui me passionnent. Je me suis donc beaucoup investi et j'ai eu plusieurs parties à réaliser concernant le casque de réalité virtuelle :

— Recherche d'une solution technique
— Calculs Optiques
— Conception du masque via un outil de *CAO*
— Réalisation et assemblage du masque

Qu'est ce que je pense de ce Projet ? Et des conditions dans lesquelles il s'est déroulé ?

J'ai été grandement étonné tout au long du projet par le potentiel que pouvais avoir la réalité virtuelle. Je ne pensais clairement pas au début du projet arriver à une telle immersion dans une autre dimension. J'ai beaucoup aimé le travail en équipe, la répartition des rôles était à la hauteur de nos capacités, mais j'ai apprécié aussi la présence des deux professeurs chargés de nous encadrer, qui répondaient à nos questions et nous permettaient d'approfondir nos travaux.

L'ensemble de notre projet s'est bien déroulé.

Qu'ai-je appris lors de ce Projet ?

Ce projet m'a permis de mieux maîtriser les différents logiciels utilisés, et m'a aussi apporté de nombreuses connaissances sur les disciplines traitées. Ce projet m'a aussi permis de découvrir les manières dont procèdent les grandes industries pour créer et optimiser au maximum leurs produits. Le travail tout au long de l'année avec des équipiers studieux m'a surtout appris à mieux travailler et gérer mon travail en équipe.

B. Coût du projet

Pour réaliser ce projet et le prototype fonctionnel, nous avons essayé de réduire au maximum les dépenses. En voici un résumé :

- **Support du joueur :**
  - **Métaux et bois :** issus de la récupération, gratuit
  - **Peinture noire :** 17 €
  - **Paraffine et essence de térébenthine :** 6.5 €
- **Casque :**
  - **Impression en 3D :** 75 €
  - **Jeu de lentilles :** 12 €
  - **Sangles :** issues de la récupération, gratuit
  - **Mousses :** issues de la récupération, gratuit
- **Logiciel de détection :**
  - **Caméra Kinect :** prêtée par le Lycée, gratuit

Le projet nous est donc revenu à 110.5 €, dont 67% sont des frais d'impression 3D.