Sorbonne Université
Master ANDROIDE

Simulation de mouvements collectifs en réalité virtuelle

UE de projet M1

Kevin Ding – Thomas ROIG – Nicolas loayza

2024

Table des matières

Chapitre 1

Introduction

L'objectif de ce projet est d'implémenter une simulation de Boids interactive en Réalité Virtuelle, à l'aide d'un casque VR Meta Quest, et réalisée en C# avec la plate-forme de développement Unity.

Le but de cette simulation est d'introduire la dynamique collective ainsi que le fonctionnements des Boids à un.e utilisateur.ice débutant.e.

Pour cela on permet à l'utilisateur de déplacer des cubes qui serviront d'obstacles pour les Boids afin d'observer l'évolution de leur comportement.

Notre encadrant, M. Nicolas Bredeche, a précisé que l'expérience à développer doit pouvoir être compréhensible par des néophytes, ainsi l'accent est mis sur l'accessibilité.

Ce projet est accessible à travers ce lien. Il nécessite d'avoir installé Unity au préalable.

Chapitre 2

État de l'art

2.1 Les Boids

Les objets "Bird-oid" (objets ayant la forme d'un oiseau), généralisés en "boids", ont été créés en 1986 par l'informaticien Craig W. Reynolds. Reynolds a présenté les Boids l'année suivante lors de la conférence *SIGGRAPH 87* [1], accompagnés d'un court-métrage [2], ainsi que du document de recherche intitulé *"Flocks, Herds, and Schools : A Distributed Behavioral Model"* [3], dans lequel il expose son travail sur les Boids et explicite les règles qui gouvernent les mouvements de ces objets capables de produire des simulations réalistes sur les mouvements de foules. Ce document, étant fortement innovant à l'époque, reste pertinent même de nos jours et forme ainsi la base de notre projet, ayant été l'inspiration de la plupart des ressources que nous avons utilisées. Avec plus de 30 ans de recherche non seulement sur les Boids, mais aussi sur la vie artificielle dans son ensemble, de nombreuses simulations 2D [4] ainsi que 3D [5] existent déjà. Cependant, les implémentations en Réalité Virtuelle restent plus rares et donc plus intéressantes.

2.2 Réalité Virtuelle

Le domaine de la réalité virtuelle, tel que nous le connaissons aujourd'hui, existait déjà dans les années 1980, même avant l'apparition des Boids [6]. Cependant, ce n'est que dans la dernière décennie que la technologie est devenue plus accessible en termes de prix et de puissance, notamment avec l'émergence de casques comme le Meta Quest 2 et 3. Ces casques peuvent être utilisés de manière indépendante, sans avoir besoin d'un ordinateur puissant et surtout sans nécessiter l'installation de caméras pour détecter les mouvements des manettes. Ainsi, ce domaine devient de plus en plus intéressant pour le développement logiciel et la recherche.

Il n'est donc pas difficile d'imaginer les avantages que ce support peut apporter aux simulations de Boids de Reynolds :

— Immersivité : l'utilisateur peut observer et étudier de près les mouvements qui se déroulent tout autour de lui.
— Interactivité : l'utilisateur est capable d'interagir avec l'environnement et les Boids de manière directe et intuitive.
— Réalisme : il est possible de créer des Boids qui ressemblent davantage à des poissons ou des oiseaux (en ajoutant par exemple des ailes qui bougeront de façon dynamique).

Chapitre 3

Contribution

Dans cette partie nous allons expliquer en détail les systèmes que nous avions utilisé ainsi que leur implémentation.

3.1 Prise en Main de Unity

Aucun membre du groupe n'avait d'expérience sur Unity, cela représente donc une opportunité pour apprendre à manipuler une nouvelle plate-forme de développement ainsi que pour se familiariser avec C#. Nous avons utilisé la version 2022.3.17f1 pour notre projet.

3.2 Implémentation des Boids

L'étude des Boids et de leurs déplacements n'est pas un sujet récent. De nombreuses ressources, expliquant les différentes règles qui gouvernent les mouvements des Boids, existent en ligne. Par exemple, le chapitre 5 de *Nature of Code* [7] nous a été utile pour comprendre le fonctionnement précis. Ainsi que le site personnel de Reynolds [8], où il explique de façon concise les Boids.

3.2.1 Propriétés

Un Boid dans les recherches et dans notre projet est représenté par un triangle. Cela permet de savoir la direction dans laquelle ils se dirigent et ce qui est dans leur champ de vision.

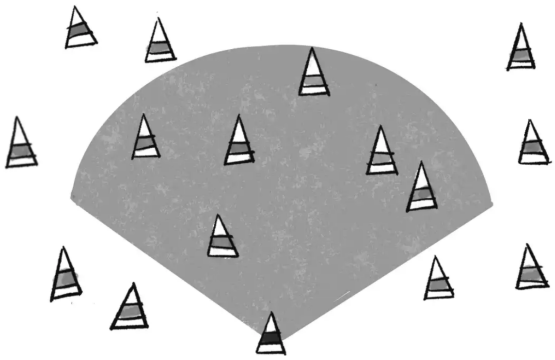

FIGURE 3.1 – Représentation d'un Boid [7]

L'angle du champ de vision d'un Boid est un paramètre qui modifie son comportement, il doit donc être choisi sciemment. Cet angle varie habituellement entre 90 degrés et 360 degrés (sans angle mort). Dans notre projet, les Boids ont un champ de vision de 270 degrés.

Les Boids se déplacent sur une arène. Comme on ne cherche pas à représenter des oiseaux, les boids implémentés ne peuvent pas voler. Ils doivent implémenter les comportements suivant : éviter les murs et les trois règles de : séparation, cohésion et d'attraction.

3.2.2 Approches envisagées

Au cours de ce projet, nous avons pu expérimenter plusieurs approches afin de réaliser les divers comportements des Boids. Pour pouvoir appliquer les divers comportements de chaque Boids il faut d'abord qu'ils puissent percevoir leur environnement. Pour cela nous avons d'abord tenté d'utiliser un collider spécifique à Unity, les *SphereCollider*. Les *SphereCollider* disposent de méthodes pouvant détecter les collisions des objets qui se trouvent dans leurs rayons.

Nous n'avons pas utilisé cette approche car cette dernière était moins intuitive pour l'implémentation de certains comportements. De plus si nous souhaitons que nos Boids soient dotés d'un angle mort, nous ne pouvons pas utiliser cette approche.

L'approche qui a été sélectionnée pour la réalisation des boids ici est l'utilisation des Raycast. Les Raycast sont des rayons qui permettent de récupérer les informations des objets avec lesquels ils entrent en collision. Comme il est possible de spécifier une distance maximale, nous utiliserons des raycasts de longueurs différentes pour chaque des comportements.

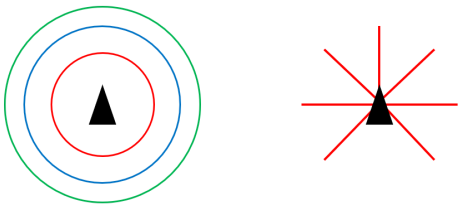

FIGURE 3.2 – Approches par SphereCollider (*à gauche*) et Raycast (*à droite*)

Pour s'assurer que les Boids ne puissent détecter que leurs semblables et les murs, nous avons utilisé le système de layers de Unity qui permet de spécifier sur quelle couche se trouvent les objets à détecter. Ainsi nous utilisons les layers suivants :

— un layer *SOL* qui correspond au sol de l'arène.
— un layer *MUR* qui permet de ne percevoir que les murs
— un layer *BOID* qui permet aux Boids de se détecter les uns les autres

Ces layers sont utilisés dans chaque comportement afin d'appliquer le bon masque aux raycasts, permettant ainsi de ne détecter que ce qui nous intéresse. Chaque comportement des boids vont utiliser des raycasts qui détecteront les collisions toutes droites, en diagonale avant gauche et droite, à gauche et à droite pour le comportement d'évitement des murs. Pour les autres comportements, on ajoute deux rayons pour détecter les collisions sur les deux diagonales arrières.

3.2.3 Architecture principale des Boids

Les Boids que nous avons implémentés sont composés des éléments suivants :

— le corps du Boid même
— un élément nommé *ground collider* qui vérifie si le Boid est en contact avec le sol
— des éléments *sphere visualizers* qui ont servi pour le débug.

Pour permettre d'éviter que les Boids se retrouvent bloqués s'ils sont en hauteur, nous appliquons nous même la gravité sur les Boids. Afin de simplifier les interactions nous avons choisi d'ignorer les collisions physiques entre Boids, ce qui fait qu'ils se traversent. Chaque Boid est composé des attributs suivants :

— *speed* qui permet de régler la vitesse de déplacement du Boid
— *withAvoid, withCohesion, withGoto*, des booléens qui permettent d'activer ou désactiver chaque comportements des Boids.
— *wallRay, avoidRay, CohesionRay, AttractionRay*, des flottants qui déterminent la distance maximale liée à chaque comportement.
— *filter*, un flottant spécifiant la valeur maximale de rotation que le Boid peut effectuer au cours d'un appel.

La méthode *update()* des Boids, appelé à chaque frame, suit le cheminement suivant :

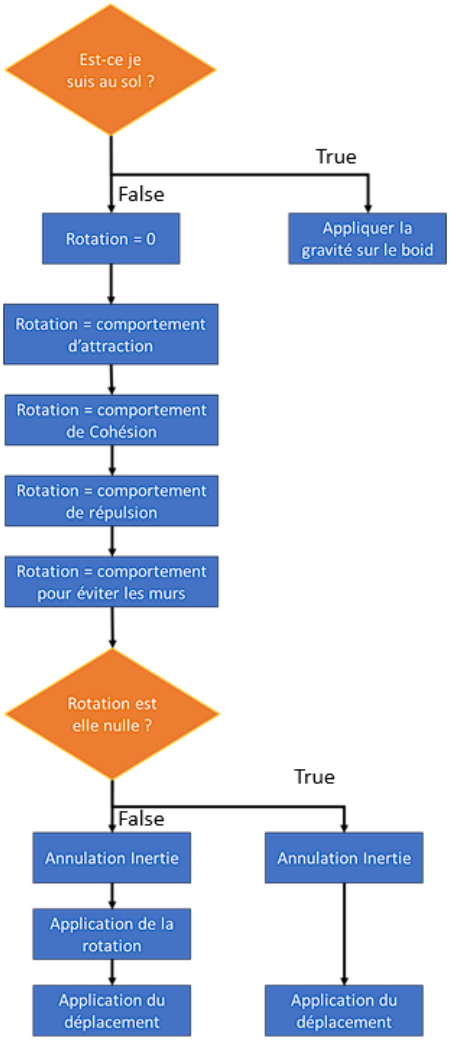

FIGURE 3.3 – Schéma de l'algorithme général du boid

On retrouve donc une architecture de subsomption sur les différents comportements des Boids. L'ordre de priorité est le suivant :

1. Éviter les murs
2. Éviter les autres Boids
3. S'aligner avec les autres Boids
4. Se rapprocher des autres Boids

Nous supprimons aussi l'inertie afin d'avoir des déplacements et rotations applicables avec des forces ce qui évite les conflits de coordonnées qui peuvent être engendrés par l'utilisation d'une simple translation.

Comportements des boids

Les comportements des Boids prennent tous en argument :

— un attribut *rotation* qui correspond à la valeur de rotation trouvée avant le comportement. Cette valeur est mise à 0 dans le cadre du premier comportement.

— un ou deux attributs *minRay, maxRay* qui sont des flottants permettant de définir les distances minimales et maximales des comportements lorsque cela est nécessaire.

Ensuite les comportement suivent les phases suivantes :

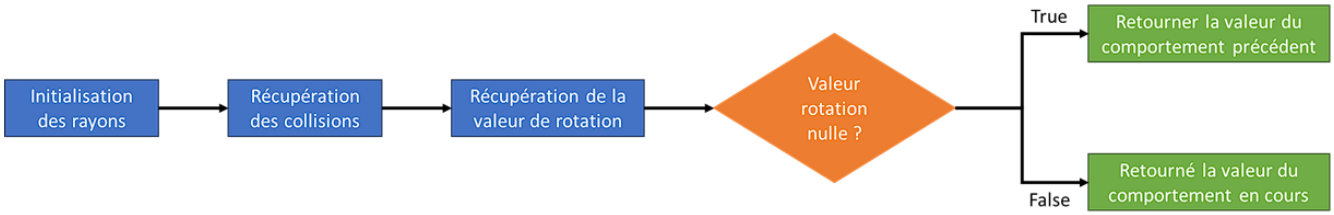

FIGURE 3.4 – Schéma des principales phases des comportements des Boids

Chaque comportement se termine en vérifiant si la valeur de rotation doit être écrasée pour garantir la priorité des comportements.

3.2.4 Evitement des murs

Les Boids doivent chercher à éviter les obstacles comme les murs. Pour cela on utilise les raycasts en masquant toutes les collisions sauf celles des murs. A chaque collision détectée on retourne une valeur d'angle associée. L'ordre de priorité de détection des collisions est le suivant :

1. collision en face de soi
2. collision diagonale gauche
3. collision diagonale droite
4. collision gauche
5. collision droite

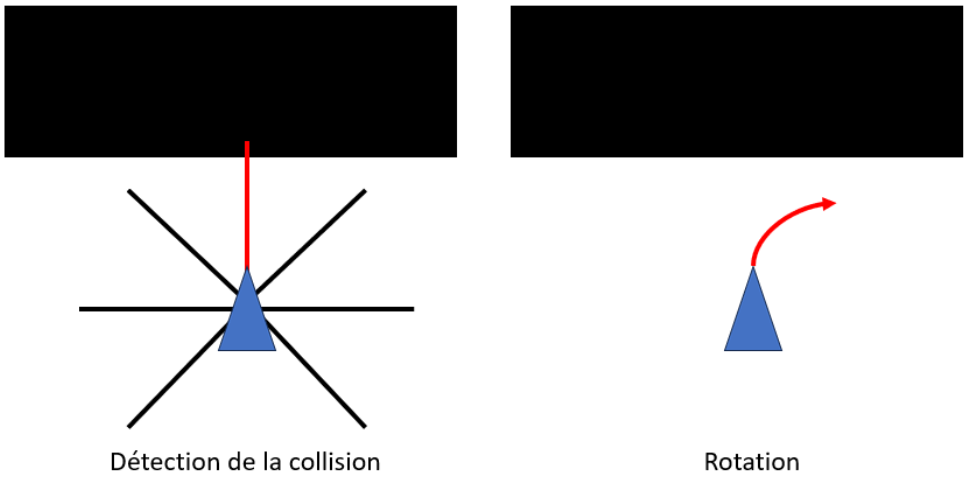

FIGURE 3.5 – Schéma du comportement d'évitement des murs

3.2.5 Séparation

Le comportement de séparation ou répulsion correspond au fait que les Boids vont chercher à éviter le Boid le plus proche d'eux. Lorsque l'on récupère les collisions, on ajoute une variable *closest* qui correspond à la position du Boid le plus proche. Ensuite on détermine la rotation à appliquer en fonction de la position du Boid par rapport à l'agent. Afin de pouvoir détecter plusieurs Boids dans la même direction on utilise la méthode *Physics.RaycastAll()*. Cependant il faut filtrer les éléments détectés car cette méthode détecte aussi le corps du Boid qui y fait appel.

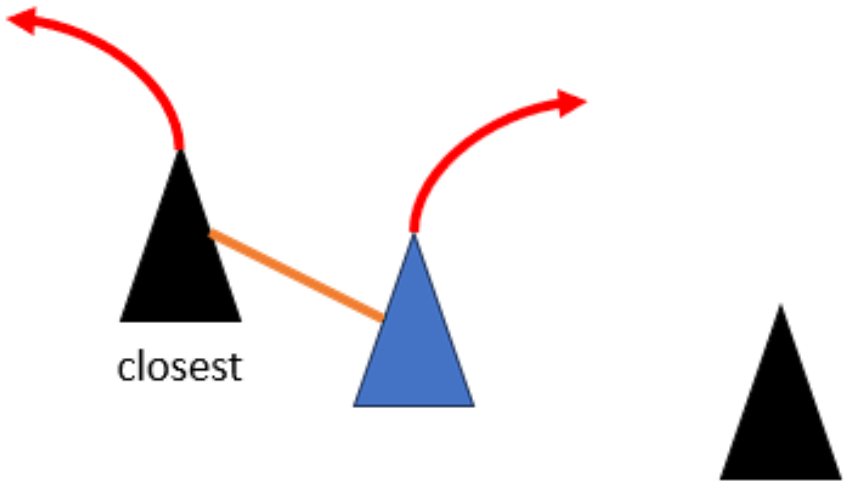

FIGURE 3.6 – Règle de Séparation

3.2.6 Cohésion

Pour la Cohésion il faut que le Boid corrige sa trajectoire vis à vis de la trajectoire des autres Boids à proximité. Pour cela lorsque l'on récupère les différentes positions des collisions, on cherche aussi à récupérer l'objet *Transform* lié afin d'obtenir le vecteur *transform.forward* qui correspond la direction du Boid avec lequel on est entré en collision. La rotation est obtenue en faisant la somme des corrections à effectuer pour le Boid. C'est ici que l'attribut *filter* sert puisque les valeurs de rotation de sortie peuvent être élevées.

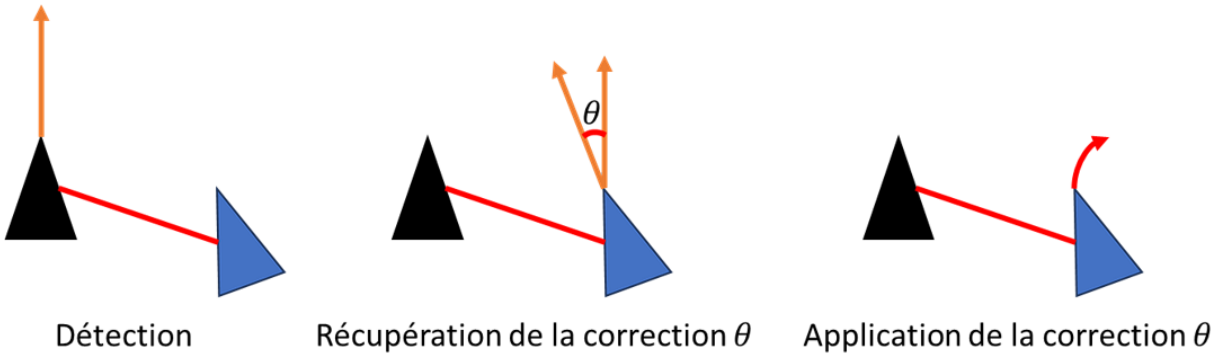

FIGURE 3.7 – Règle de Cohésion

Le procédé décrit par la figure ci-dessus est appliqué par chaque Boid pour tous les Boids qui sont détectés. Le fait que chaque Boid cherche à s'aligner avec les autres est ce qui fait qu'un groupe de Boids finira par avancer dans la même direction.

3.2.7 Attraction

Le dernier comportement du Boid implémenté est celui d'attraction. Dans celui-ci un Boid, dont la rotation n'est écrasée par aucun autre comportement, va chercher à se rapprocher des autres. Pour cela le procédé de récupération des collisions est similaires puisque l'on va chercher à se diriger vers le groupe le plus proche. On récupère donc le Boid le plus proche et on retourne la valeur d'angle entre le Boid et la destination voulue. Même si les valeurs sont filtrées, l'efficacité est atteinte par la répétition des rotations à chaque pas de temps.

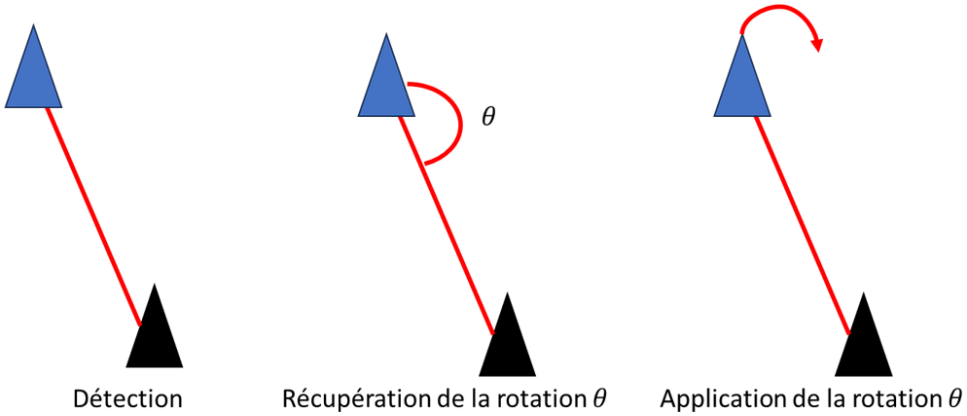

FIGURE 3.8 – Règle d'Alignement[8]

3.3 Implémentation de l'environnement

Dans cette partie, nous allons voir comment l'environnement a été généré. Plutôt que d'utiliser un objet déjà existant, nous allons créer nous-mêmes un objet afin d'avoir plus de contrôle sur sa forme, sa structure et son interaction. Pour créer un objet sur Unity, il nous faut plusieurs composants : un *MeshFilter* afin d'obtenir la forme que nous voulons et un *MeshRenderer* afin de pouvoir visualiser l'objet et lui appliquer une texture. Chaque objets que nous allons instancier comporte une liste de points nommés *vertices* qui correspondent à chaque sommet de l'objet, ainsi que des triangles qui correspondent à chaque face de l'objet.

Prenons exemple sur la construction d'un carré en deux dimensions.

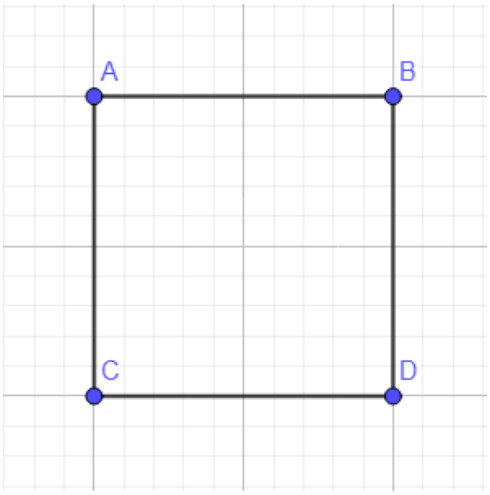

FIGURE 3.9 – Construction d'un carré

Sur le carré ci-dessus, si l'on prend pour coordonnées $A = (1, 0)$, $B = (1, 1)$, $C = (0, 0)$ et $D = (0, 1)$, on a alors la liste $[A, B, C, D]$ qui correspond à la liste de *vertices*. Il nous manque plus qu'à ajouter les triangles afin de pouvoir visualiser le carré. Un carré est composé de 2 triangles, le premier triangle sera donc le triangle A, D, C, le second sera A, B, D. À noter que l'ordre de sélection des points est important afin de visualiser l'objet correctement. L'ordre de sélection dans le sens horaire permet une visualisation extérieure de l'objet tandis que le sens anti-horaire permet une visualisation intérieure de l'objet.

En procédant de la même manière, nous pouvons alors créer un cube en 3 dimensions ayant pour layer "Sol", comportant 4 fils qui correspondent à 4 faces supplémentaires sur les côtés ayant pour layer "Mur" afin que les Boids détectent un obstacle. En utilisant cette structure si le cube est déplacé, les 4 faces supplémentaires seront aussi déplacées. Grâce au cube précédemment créé, nous pouvons alors générer une arène comportant un damier pour que nos Boids puissent s'y déplacer dessus.

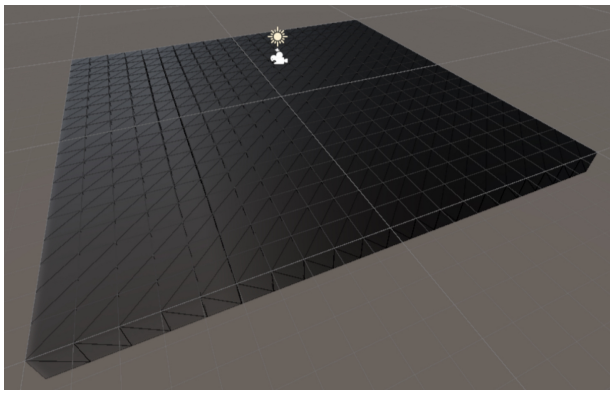

FIGURE 3.10 – Arène comportant 20*20 cubes

Concernant la limitation de l'arène, dans le fichier *CircleWallScript*, nous créons un mur en fonction du nombre de faces (*sides*), du diamètre (*radius*) et de la hauteur (*height*) souhaités. La fonction *DrawWall()* va d'abord calculer un angle initial de 2π qu'elle divise par le nombre de faces désiré, puis elle place *sides* fois un point à la hauteur 0 et un autre à la hauteur *height*, en laissant un écart de $2\pi/sides$ entre chaque paire de points. On obtient par exemple 4 faces pour une arène carrée, 6 faces pour une arène hexagonale, etc. Après avoir récupéré tous les points, nous allons construire tous les triangles pour enfin pouvoir visualiser le mur.

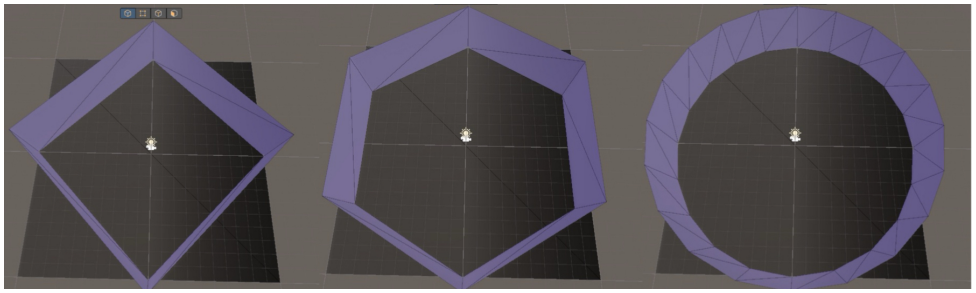

FIGURE 3.11 – Exemple d'arènes comportant des murs avec 4, 6 et 20 faces

3.4 Interaction avec l'environnement

Dans cette partie, nous allons voir comment l'interaction entre l'utilisateur et l'environnement est possible.

Pour établir une interaction entre l'utilisateur et l'environnement, nous avons besoin de deux composants. Le premier est un composant de type *XR Interactor* nous allons utiliser le *XR Direct Interactor*, qui sera placé sur un ou les deux contrôleurs afin qu'ils puissent interagir avec d'autres objets. Le deuxième composant est de type *XR Interactable* ; nous allons utiliser le *XR Simple Interactable*, qui sera placé sur les objets avec lesquels nous voulons obtenir une interaction.

Concernant le *XR Direct Interactor*, il doit simplement être placé sur un des contrôleurs ; nous l'avons placé sur la main gauche, et il n'y a pas d'ajout spécifique à faire. Pour le *XR Simple Interactable*, nous allons ajouter les interactions nous-mêmes. Nous avons implémenté trois interactions. Avec la fonction *OnHoverEntered()*, chaque fois que les contrôleurs entrent en contact avec l'objet en question, la couleur de l'objet va changer grâce à la fonction *setNewMesh* des objets. Pour *OnHoverExited()*, chaque fois que la

main gauche n'est plus en contact avec l'objet, nous remettons la couleur initiale de l'objet. Enfin, avec la fonction *ProcessInteractable()*, chaque fois que la main gauche est en contact avec un objet et que la gâchette est utilisée, le *vertice* supérieur de l'objet va être modifié en fonction de la hauteur de la main gauche.

3.5 Résultats

Grâce à tous ces objets nous pouvons alors créer une arène comportant des Boids.

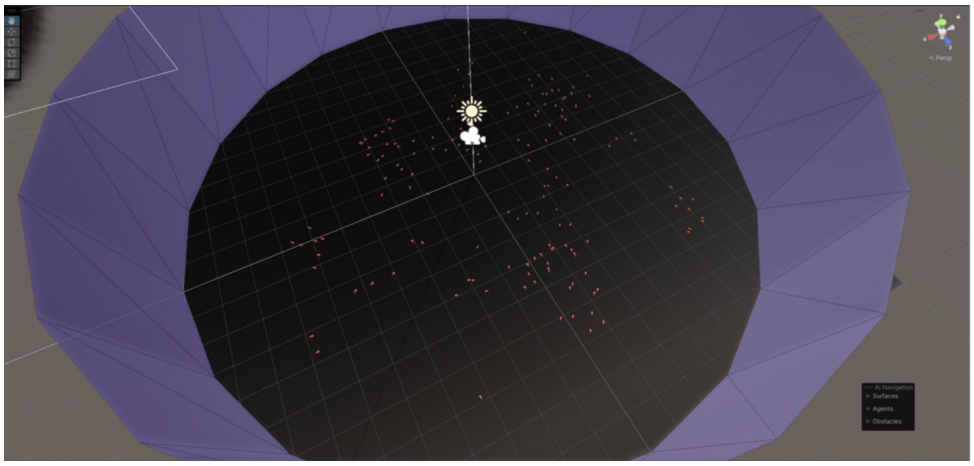

FIGURE 3.12 – Arène comportant 150 Boids

Nous avons pu étudier les différents comportements des Boids, et nous trouvons des résultats imparfaits mais assez cohérents avec ce que nous désirons. De plus il est possible, de faire varier la hauteur des cubes si l'arène est générée en damier et d'observer que les Boids réagissent en conséquence. On remarque tout de même des petits problèmes lors de l'évitement des murs, probablement réglables avec un nombre de raycast plus grand.

Bien que la performance de la simulation dépend des ordinateurs utilisés, nous pouvons instancier 300 Boids sans le damier tout en gardant un nombre d'images par seconde de 30 FPS. En utilisant le damier nous pouvons instancier environ 100 Boids tout gardant 25 FPS.

Chapitre 4

Conclusion

Nous avons réussi à implémenter une simulation satisfaisante de Boids en Unity, avec les comportements de séparation, cohésion et d'attraction permettant d'observer les déplacements de foules de la même façon que Reynolds l'a fait dans les années 80. De plus l'implémentation en Réalité Virtuelle, bien que rudimentaire, permet à l'utilisateur d'interagir avec les Boids de façon directe, donnant ainsi un côté plus immersif à la simulation.

Cependant, l'aspect visuel des Boids et l'arène, bien que pratique pour le développement, est simple à l'excès et pourrait être plus attirant. De plus, l'interactivité est suffisante mais pourrait être enrichie avec l'introduction d'interfaces permettant de rajouter un plus grand nombre et plus de diversité d'obstacles. Le fait de pouvoir manipuler directement le nombre de Boids ou d'activer/désactiver certains d'entre eux pourrait être envisagé. Finalement une piste de développement intéressante serait d'implémenter cette simulation non pas en Réalité Virtuelle mais en Réalité Mixte permettant à l'utilisateur de rester conscient de l'environnement réel. On éviterai ainsi le problème de l'utilisateur isolé. Cette problématique fera l'objet d'un stage de un mois pour deux membres du projet.

Pour conclure, nous avons beaucoup appris avec ce projet, malgré la quantité de nouvelles connaissances à acquérir, ce premier projet utilisant Unity a été très formateur. L'emploi de Unity comme plate-forme de développement s'est avéré challengeant, mais il s'agissait d'un des facteurs qui nous avait motivé à choisir ce projet. Nous remercions notre encadrant, M. Nicolas Bredeche de nous avoir proposé ce sujet et de nous avoir guidé tout au long du semestre.

Bibliographie

[1] « 14th Annual Conference on Computer Graphics and Interactive Techniques ». In : 1987. URL : https://history.siggraph.org/conference/siggraph-1987-14th-annual-conference-on-computer-graphics-and-interactive-techniques/ (page 2).

[2] *Stanley & Stella in : Breaking The Ice*. Le lien Youtube est une archive. 1987. URL : https://www.youtube.com/watch?v=pbFEQv259yw (page 2).

[3] Craig W. Reynolds. « Flocks, Herds, and Schools : A Distributed Behavioral Model ». In : *Computer Graphics* (1987). URL : https://team.inria.fr/imagine/files/2014/10/flocks-hers-and-schools.pdf (page 2).

[4] *Simulation de Boids en 2D*. Simulation 2D en ligne de Boids crée par Ben Eater. URL : https://eater.net/boids (page 2).

[5] *Simulation de Boids en 3D*. Simulation 3D en ligne de Boids crée par Ercan Gerçek. URL : https://ercang.github.io/boids-js/2-boids-grids/ (page 2).

[6] *Site de VPL research*. Les premiers produits "VR". URL : https://www.vrs.org.uk/virtual-reality-profiles/vpl-research.html# (page 3).

[7] Daniel Shiffman. *The Nature of Code*. Le livre est accessible gratuitement et de facon interactive. D. Shiffman, 2012. URL : https://natureofcode.com/ (pages 4, 5).

[8] Craig W. Reynolds. *Boids Background and Update*. URL : https://www.red3d.com/cwr/boids/ (pages 4, 10).

Annexe A

Cahier des charges

L'objectif de ce projet est d'implémenter une simulation de Boids interactive fonctionnant sur un casque virtuel Meta Quest (version 2 ou 3). Les Boids sont des particules simulant les mouvements de groupes d'agents, en s'inspirant des comportements collectifs observés chez les animaux. L'utilisateur humain devra pouvoir interagir avec les Boids soit directement (en saisissant les Boids pour les déplacer), soit indirectement (en modifiant la structure de l'environnement, par exemple en plaçant des obstacles).

Visuellement, l'utilisateur humain est placé au centre d'un anneau, dans un espace réservé. L'utilisateur reste immobile et peut tourner sur lui-même.

Nous souhaitons simuler le déplacement en 2D des Boids dans un environnement en forme d'anneau. L'utilisateur est placé au centre de l'anneau, dans le cercle intérieur. Les Boids se déplacent exclusivement entre le cercle intérieur et le cercle extérieur de l'anneau. L'utilisateur peut voir les Boids, interagir avec eux et ajouter/supprimer des objets dans l'environnement (création, déplacement, suppression).

L'objectif général est de produire une simulation permettant à un utilisateur néophyte de découvrir et comprendre la dynamique collective d'une foule d'individus.

Le développement sera réalisé en C# avec Unity. Un casque Meta Quest 3 sera mis à disposition au sein du laboratoire pour effectuer des tests. Un objectif souhaité est de minimiser l'interface apparente (les commandes de manipulation de Boids ou d'objets n'apparaissent qu'à la demande).

Annexe B

Manuel utilisateur

B.1 Organisation des fichiers

Après avoir importé le projet, vous retrouverez dans le dossier "FinalScene" tous les éléments utilisés lors du projet. Il comporte cinq dossiers : Boids, Ground, Hands, Test, et Wall, une scène nommée FinalScene, ainsi qu'un script "InitSceneScript".

Dans le dossier Boids, vous trouverez le prefab du Boid, le code de son déplacement, ainsi que le code de la gravité. Dans le dossier "Ground", vous retrouverez un prefab cube ainsi qu'un prefab plane. Le prefab cube comporte un script parent et tous les scripts enfants associés. Le prefab plane comporte son script associé. Le dossier "Hands" contient les scripts pour l'animation des mains. Le dossier "Wall" contient le prefab du mur ainsi que son script. Dans la scène FinalScene, vous retrouverez la scène finale du projet utilisant le script "InitScene", qui instancie tous les prefabs du projet.

B.2 Utilisation du projet

Avant de lancer le projet, si vous avez un casque VR, et si ce n'est pas déjà fait, vous devez désactiver le "XR Device Simulator". Sinon, utilisez-le en l'activant (voir figure ci-dessous).

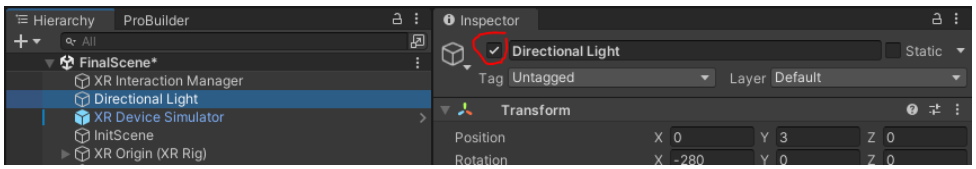

FIGURE B.1 – Activation/Désactivation du XR Device Simulator

Ensuite, en cliquant sur l'élément InitScene, vous retrouverez dans la partie Inspector tous les paramètres de l'environnement qui sont modifiables. Après avoir choisi les paramètres que vous souhaitez, vous pouvez lancer le projet en cliquant sur le bouton Play.

Si vous utilisez le "XR Device Simulator", vous utiliser la touche tab pour contrôler la main gauche puis la main droite et enfin revenir sur le contrôle de la tête. En étant sur une des deux mains vous pouvez appuyer sur la touche R afin de contrôler la rotation ou

la profondeur de la main, ainsi que de la molette pour pivoter ou de contrôler la hauteur de la main. Enfin vous pouvez appuyer sur la touche G lorsque la main est en contact avec le sol afin de pouvoir attraper le sol et ainsi modifier la hauteur de celui-ci.