Présentation des réalisations produites à l'aide de moteurs de jeux

Florian BOISSET
Galerie Itch.io
À propos de moi

15 mai 2026

Résumé

Ce document a pour but la présentation et la valorisation de mon travail en tant que développeur. Mes réalisations sont produites à l'aide de moteurs de jeux, et je souhaite partager ces projets pour démontrer mes compétences et mon expérience dans ce domaine. Ainsi, pour chaque projet, je présenterai le contexte de développement, les objectifs du projet, les technologies utilisées, les défis rencontrés et les solutions apportées. Je mettrai également en avant les résultats obtenus et les fonctionnalités implémentées. Pour les projets ayant été publiés, je fournirai des liens vers les plateformes de distribution ou les démonstrations en ligne. De nombreuses captures d'écran et vidéos seront incluses pour illustrer les réalisations et permettre une meilleure compréhension de chaque projet.

Table des matières

1 AGAGAB - A Game About Growing A Bean

1.1 Présentation du jeu

AGAGAB est le premier jeu dont le développement est allé jusqu'à la publication. Il s'agit d'un jeu dans lequel le joueur a pour objectif de faire pousser un haricot jusqu'à atteindre sa maison, envolée sur une île flottante. Des agents gouvernementaux viendront lui mettre des bâtons dans les roues. AGAGAB a été développpé à l'aide de Unity, et a été publié sur Itch.io en mars 2026. Il continue d'être mis à jour régulièrement.

La page Itch.io du jeu !

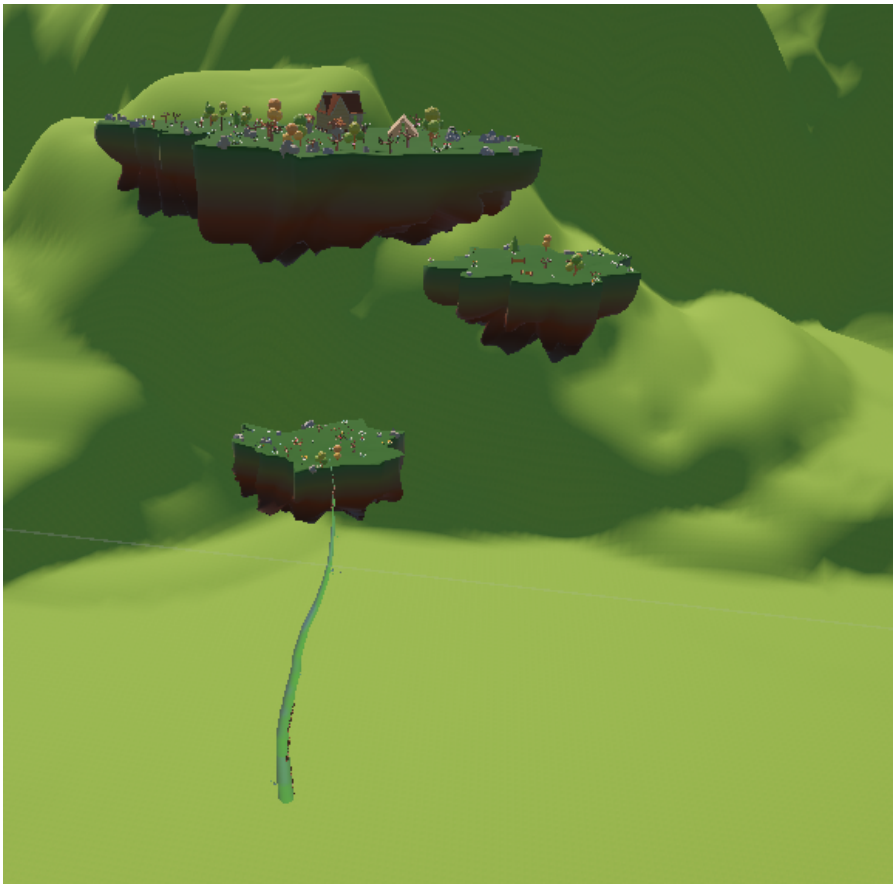


FIGURE 1.1 – Capture montrant des îles flottantes et une plante qui pousse.

1.2 Contenu du jeu

1.2.1 La map

Des îles flottantes présentes dans le ciel composent l'environnement de jeu. Le joueur doit faire pousser son haricot depuis le plancher des vaches pour atteindre les îles sur lesquelles il pourra trouver des objets à débloquer lui permettant de rejoindre des îles de plus en plus haut jusqu'à rejoindre sa maison sur l'île la plus haute.

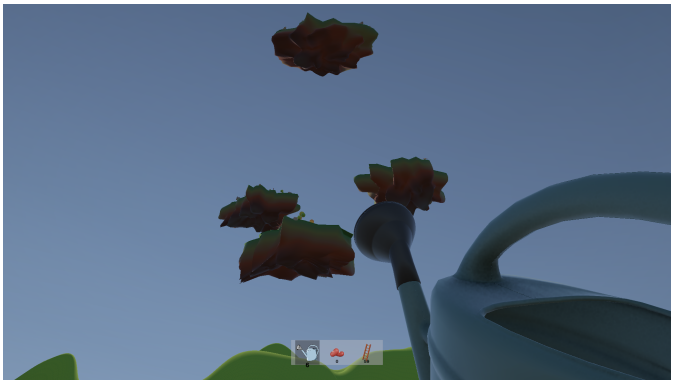


FIGURE 1.2 – Screenshot des îles vues du sol.

1.2.2 La plante

Le joueur fait pousser un haricot en l'arrosant, et en ajoutant de l'engrais. Il plante une graine dans le sol, sur une île flottante, ou sur une autre plante pour faire pousser un nouveau haricot. Le haricot produit des fruits de deux types, le premier sert de monnaie, le second, bien plus rare, sert de graine pour replanter.

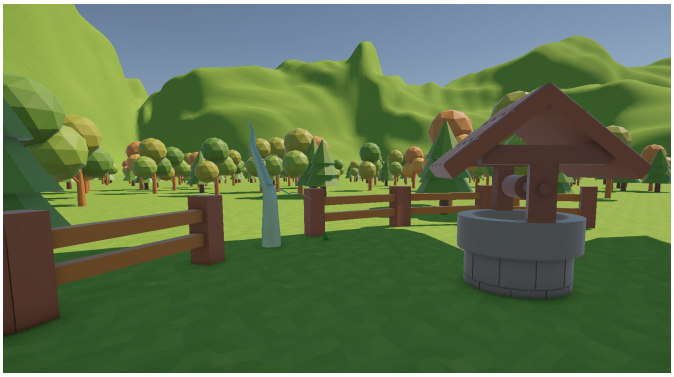


FIGURE 1.3 – La plante au sol en cours de croissance.

1.2.3 Les échelles

Pour grimper sur les plantes verticales, le joueur doit placer des échelles sur ces dernières. En poussant, les plantes englobent les échelles, forçant le joueur à en placer de nouvelles.

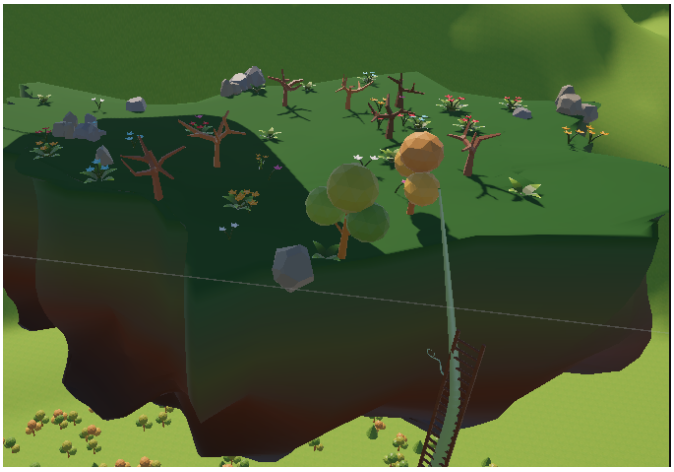


FIGURE 1.4 – La plante arrivant au niveau d'une île avec une échelle placée.

1.2.4 La boutique

Une boutique permet au joueur d'acheter des consommables, tels que les échelles, les engrais ou des munitions.

FIGURE 1.5 – La boutique où le joueur peut acheter des consommables.

1.2.5 Les ennemis

Des espions viennent d'abord en reconnaissance. S'ils restent en vie suffisamment longtemps et qu'ils effectuent suffisamment de signalements, des soldats apparaîtront et cibleront le joueur. Le joueur dispose d'armes pour se défendre, mais

il doit d'abord les débloquer en trouvant les licences d'armes cachées sur les îles flottantes.

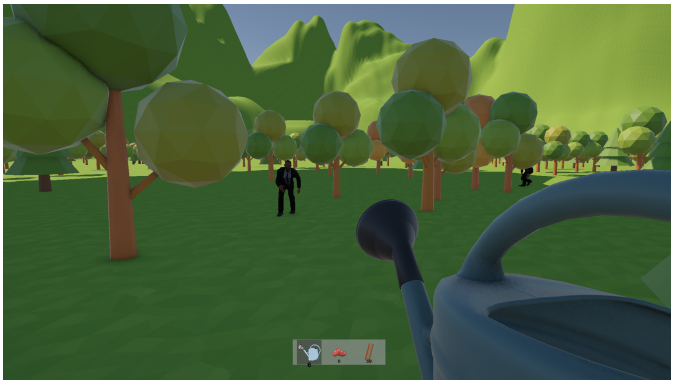


FIGURE 1.6 – Des espions en reconnaissance.

1.2.6 Sauvegarde

Le joueur a la possibilité de sauvegarder sa progression pour recharger sa partie plus tard. Pour cela, il doit se rendre à un point de sauvegarde (tente ou cabane) et interagir avec le bâtiment.

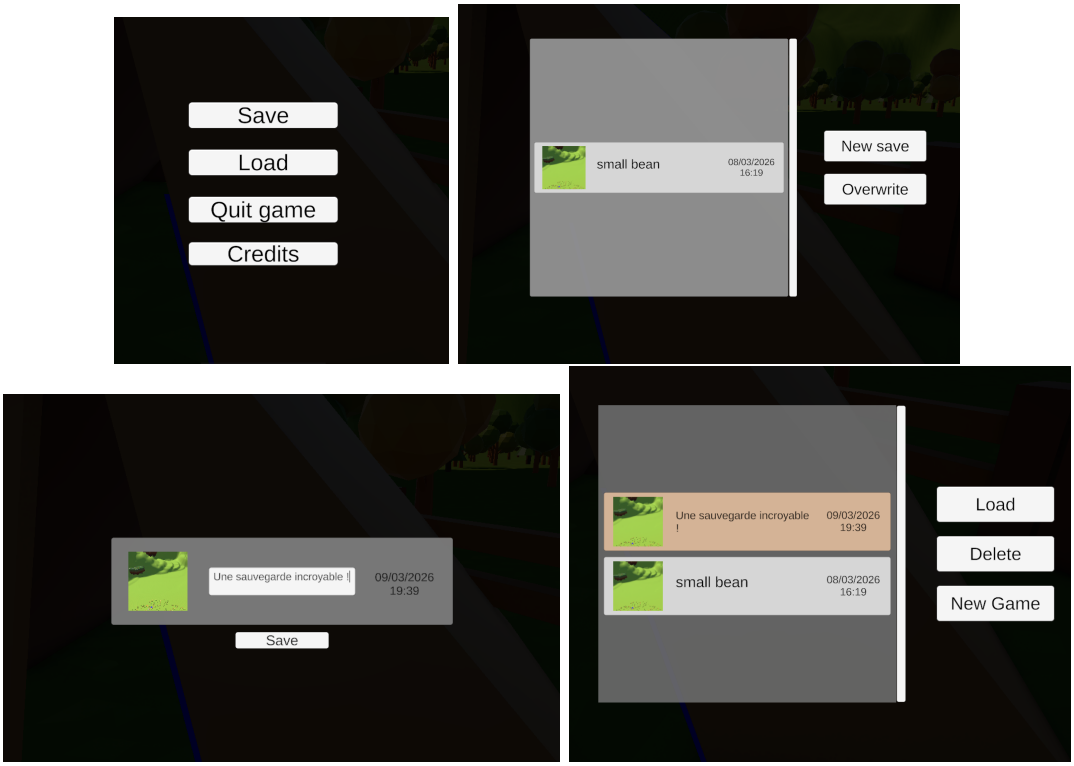


FIGURE 1.7 – Capture des menus pour sauvegarder et charger.

1.3 Développement du jeu

1.3.1 La map

Les îles flottantes sont générées **procéduralement**. On part d'une sphère unitaire UV, puis on vient la transformer. D'abord une **élongation** est appliquée pour donner une forme plus éllipsoïdale à notre île. Des bruits de Perlin désynchronisés (avec des offsets) sont utilisés pour déformer l'espace de coordonnée **(Domain Warping)** pour donner une forme plus organique. On déplace les points avec un **Fractional Brownian Motion** dont on contrôle la persistance et la fréquence pour donner plus ou moins de détails à l'île. Enfin, on applatie le dessus de l'île et on étire le bas. On applique un shader qui colore l'île en fonction de l'altitude du triangle (marron en bas, vert en haut), on évite de devoir vraiment les texturer une par une.

On place ensuite des objets sur l'île :

— Sur l'ile la plus haute, on place la maison (objectif finale) ;

— Trit des objets de la liste par tag (plante, rocher, batiment...) ;

— On sélectionne aléatoirement des triangles du maillage de la surface de l'île ;
— On filtre les triangles selon leur inclinaison, la distance aux autres triangles... ;
— On tire aléatoirement une catégorie et un item de la catégorie.

1.3.2 La plante

Les plantes sont générées procéduralement. Un **Perlin Worm** influencé est utilisé pour générer une forme de plante, qui est ensuite convertie en une série de **segments**. Chaque segment est composé d'un cylindre. En poussant, on agrandit les cylindres des segments, déplaçant les segments suivants. On active les segments une fois un certain diamètre atteint. Plusieurs méthodes de génération de plantes ont été testées, la méthode retenue consiste à **générer une pool de plantes au lancement du jeu**, qui sont placées dans la scène lorsque le joueur plante une graine.

1.3.3 Les objets

Un gestionnaire d'objets gère l'état interne des différents objets initialisés (Verrouilé, Débloqué, Equipé, Sélectionné). Un **RayCast** réguilier vérifie la présence d'un objet à équiper ou à débloquer devant le joueur. Une **classe objet** définit les comportements et variables classiques des objets, et est **héritée par les différents types d'objets** (quantité, prix, description pour la boutique...). Les objets sont **abonnés directement à certains inputs** du joueur (action principale, secondaire, interaction). On peut ainsi ajouter facilement de nouveaux objets dans le jeu. Ces **abonnements sont révoqués temporairement** par exemple si le joueur est face à la boutique pour cliquer sans activer l'objet.

— Les échelles sont issues d'un package de l'asset Store à peine modifié, puis exploité.
— Les comportements des autres objets ont été créés de toute pièce.
— Leurs modèles 3D sont pour la plupart issus de l'asset store, quelques uns ont été modélisés à l'aide de Blender.

1.3.4 La boutique

La boutique est une UI sur un canvas en world space qui répond à un curseur virtuel placé au centre de l'écran. Régulièrement, on tire aléatoirement les objets qui seront vendus et un multiplicateur pour les prix. Un template d'UI d'objet est alors rempli avec ces informations. Une seconde fenêtre d'information affiche les détails de l'objet sélectionné. On utilise les Buttons, ScrollBar, Panel de Unity pour

construire l'interface et détecter les interactions. Lorsque le joueur regarde la boutique, on révoque l'abonnement du gestionnaire d'objets à l'input de changement d'objet pour pouvoir scroller.

1.3.5 Les ennemis

Une **machine d'état** détermine le comportement des espions et des soldats. Le déplacement des ennemis est assuré par un **NavMeshAgent**, qui leur permet de se déplacer sur le terrain qui est un **NavMeshSurface**. Des comportements spécifiques ont été créés pour les soldats tels que trouver un objet pour se protéger. Les animations sont gérées par un **Animator**, qui reçoit des paramètres de la machine d'état pour faire correspondre les animations au comportement de l'ennemi.

1.3.6 Sauvegarde

Un système de sauvegarde basé sur la sérialisation JSON a été mis en place. Une classe de sauvegarde contient les variables à sauvegarder, et est sérialisée en JSON pour être écrite dans un fichier. Pour éviter de surcharger le fichier de sauvegarde, on garde uniquement les Seeds des éléments procéduraux (map, plantes). On économise ainsi énormément de stockage. On enregistre aussi l'état des objets (débloqués, équipés, leurs quantités...) ainsi que le point de départ et d'arrivée de chaque échelle placée par le joueur. Lors du chargement, le fichier est désérialisé pour récupérer les données de sauvegarde.

1.3.7 Détection d'objets

Le joueur a plusieurs systèmes de détection, pour les objets collectibles, on lance un raycast devant le joueur régulièrement et on assigne aux Prefabs des objets un layer. Selon l'objet détecté par le Raycast, on déclenche une méthode ou une autre. Dans le cas du sol ou des échelles, on utilise un trigger placé au pied ou devant le joueur.

Synthèse de rapport de stage chez Assystem - Projet de Réalité Virtuelle pour le nucléaire

1.4 Serious Game en Réalité Virtuelle (mars 2025)

Poste : Ingénieur Développement en Réalité Virtuelle
Entreprise : Assystem — agence de Venelles (Bouches-du-Rhône)
Durée : 6 mois (mars – août 2025)
Filière ISIMA : Informatique des systèmes interactifs pour l'embarqué et le virtuel
Technologies principales : Unity, C#, Python, Pixyz, HTC VIVE, CATIA
Domaine applicatif : Nucléaire — projet ITER / CEA Cadarache

1.5 Contexte

1.5.1 L'entreprise

Assystem est un groupe international d'ingénierie spécialisé dans la gestion de projets à forte composante technologique (énergie nucléaire, infrastructures critiques, transports, santé). L'agence de Venelles contribue notamment au projet **ITER** (*International Thermonuclear Experimental Reactor*), réacteur expérimental de fusion nucléaire implanté à Cadarache, et accompagne le **CEA Cadarache** sur plusieurs projets de sûreté nucléaire.

1.5.2 Enjeu du stage

Face à des environnements industriels complexes, dangereux ou inaccessibles physiquement, Assystem a fait le choix stratégique d'intégrer la **Réalité Virtuelle**

(RV) à ses processus d'ingénierie. La RV permet de simuler des interventions humaines (assemblage, maintenance, démantèlement) pour :

— évaluer la **faisabilité ergonomique** des gestes techniques dans des espaces confinés ;

— quantifier la **dose radiologique** reçue par les opérateurs lors de chaque scénario ;

— réduire les erreurs de conception avant toute fabrication physique.

L'objectif central du stage était de rendre ces simulations **accessibles à des ingénieurs non développeurs**, en réduisant le temps de préparation et en industrialisant les workflows Unity.

1.6 Réalisations techniques

1.6.1 Template Unity unifié

Les deux projets existants (ergonomie et dosimétrie) étaient maintenus séparément, entraînant une duplication des assets et des changements de contexte coûteux. J'ai **fusionné ces projets** en un seul template générique activable par modules, ce qui réduit l'espace disque et simplifie la gestion de versions.

Intégration du projet étudiant (Centrale Lyon) : un module de manipulation d'objets en scène (instanciation, translation/rotation, étiquetage) développé par des étudiants sous émulateur a été adapté aux contrôleurs HTC VIVE d'Assystem, en corrigeant les incompatibilités d'UI/UX.

Outils instanciables : développement d'un éditeur Unity permettant d'ajouter un outil 3D (format FBX) à une simulation en quelques clics — attribution automatique de collisions, gravité et saisie VR, génération d'une icône identifiante — sans compétence Unity requise.

1.6.2 Outils d'édition et de préparation de scènes

— **CamViewTool :** interface de contrôle des caméras spectateurs depuis l'inspecteur Unity (caméra orbitale, mode libre clavier, caméras fixes, sélection automatique par comptage de pixels de l'opérateur visible). Intègre l'activation de l'analyse REBA.

— **PrefabCleaner :** outil de nettoyage automatique des Prefabs importés via Pixyz — suppression des scripts manquants, conversion des matériaux vers URP/Lit, application d'un matériau par défaut — pour éliminer erreurs et warnings bloquants sans expertise Unity.

— **Volume Filter Tool :** suppression automatique des petits éléments 3D (vis, boulons…) en-dessous d'un seuil de volume défini par référence, allégeant significativement les scènes.

1.6.3 Calculateur REBA en Réalité Virtuelle

Le **REBA** (*Rapid Entire Body Assessment*) est une méthode d'analyse ergonomique évaluant le risque musculosquelettique d'une posture en tenant compte de l'ensemble du corps (tronc, cou, jambes, bras, avant-bras, poignets, charges). À ma connaissance, aucun plugin commercial ne propose d'implémentation REBA clé en main dans Unity.

J'ai développé un **calculateur REBA natif Unity** :

— Référencement des segments corporels via l'inspecteur du mannequin VR ;

— Mesure des angles articulaires à chaque frame par calcul vectoriel entre segments parent/enfant ;

— Fonction de calibration sur posture neutre pour corriger la morphologie initiale ;

— Application des règles de notation REBA par segment (scores +1 pour inclinaisons/rotations) ;

— Évaluation **bilatérale** (gauche et droit) avec conservation du score le plus conservatif ;

— Cases à cocher pour renseigner charge soulevée et qualité de préhension ;

— Visualisation temps réel du squelette colorisé et du score global dans l'interface de supervision.

1.6.4 Module Radiations — dosimétrie en VR

— **Reformatage de fichiers :** développement d'un script Python convertissant automatiquement deux formats de cartes radiologiques (données fournies par d'autres équipes) vers le format compatible avec le système Unity. Ajout d'une décimation du nuage de points pour limiter la latence de rendu.

— **Outil de placement de zone :** nouveau composant Unity permettant de matérialiser, déplacer et redimensionner interactivement le nuage de radiation dans la scène, comblant l'absence totale d'outil de repositionnement dans l'existant.

1.6.5 Pipeline CAO → Unity via Pixyz

Les modèles CATIA (format 3DXML) n'étant pas directement compatibles avec Unity, Pixyz est utilisé comme couche de conversion et d'optimisation. J'ai

défini un **projet Unity préconfiguré** avec des *Rule Engines* Pixyz adaptés à deux cas d'usage :

— **Objets manipulables :** haute précision géométrique + colliders précis ;

— **Éléments passifs** (structures, conduites) : simplification poussée pour les performances.

Le système LOD (*Levels of Detail*) a été évalué puis écarté, sa configuration par-modèle étant incompatible avec l'objectif de préparation rapide et standardisée.

1.7 Compétences mobilisées

— **Développement 3D/RV :** Unity (C#), outils éditeur custom, XR Interaction Toolkit, HTC VIVE.

— **Traitement de données :** Python (conversion/décimation de nuages de points), formats CAO.

— **Pipeline CAO :** Pixyz (Rule Engine, optimisation maillage, colliders), 3DXML, FBX.

— **Ergonomie :** méthode REBA, analyse posturale, calcul d'angles articulaires en temps réel.

— **Radioprotection :** dosimétrie VR, nuages de radiation, algorithme Octree.

— **Transversal :** documentation technique, UX/UI pour non-développeurs.

1.8 Bilan

Ce stage m'a permis de mener des développements complets sur un périmètre large — du traitement de données CAO jusqu'à l'ergonomie et la radioprotection — dans un contexte industriel exigeant (nucléaire, contraintes réglementaires). Le fil directeur, rendre la RV accessible à des ingénieurs non informaticiens, m'a conduit à soigner autant l'expérience développeur (outils éditeur Unity, documentation) que les performances runtime.

Sur le plan personnel, cette expérience a renforcé ma capacité à identifier rapidement les manques d'une solution existante, à proposer des architectures pragmatiques et à arbitrer entre généricité et performance — compétence particulièrement sollicitée sur les sujets Pixyz/LOD et REBA/automatisation squelettique.

Mots-clés : Unity, C#, Python, Réalité Virtuelle, REBA, Pixyz, Dosimétrie, ITER, Ergonomie, Nucléaire.

2 Serious Game en Realite Virtuelle (mars 2025)

Projet réalisé en binôme sur Unity en réalité virtuelle pour un Meta Quest 2. Noms des participants : Florian BOISSET et Elisa MARCHAND.

2.1 Présentation et contexte du projet

Cette partie présentera une synthèse d'un rapport de projet réalisé dans un cadre pédagogique de réalisation d'un *Serious Game VR*. Le projet vise à concevoir deux expériences immersives de formation en réalité virtuelle : une simulation de tri de pneus et un simulateur de grue. Les deux modules ont été développés autour d'une architecture orientée événements, d'une interaction utilisateur basée sur les outils XR, et d'un système de score persistant. Cette synthèse met en avant les choix techniques, le déroulement des interactions, les critères de réussite, ainsi que les limites observées et les pistes d'amélioration.

L'objectif de ce serious game est de proposer des situations proches du terrain professionnel. Les objectifs principaux étaient les suivants :

— concevoir des interactions naturelles en réalité virtuelle ;

— structurer les comportements des objets et des interfaces via des événements ;

— fournir un retour immédiat à l'utilisateur (visuel, sonore et score) ;

— enregistrer les performances pour suivre la progression d'une session à l'autre.

Le système est composé de deux épreuves complémentaires. La première évalue la capacité de l'utilisateur à identifier et trier des pneus selon leur conformité. La seconde évalue la précision de pilotage d'une grue dans un parcours variable. Ensemble, ces deux modules couvrent des compétences de diagnostic, de coordination gestuelle et de maîtrise opérationnelle.

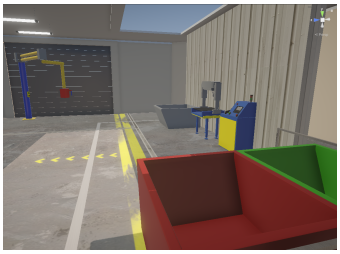


FIGURE 2.1 – Vue d'ensemble de la scène VR.

2.2 Module 1 : simulation de tri de pneus

2.2.1 Génération et gestion des pneus

Le coeur du module est centralise dans un gestionnaire dedie (*TireManager.cs*). Le lancement de l'epreuve est realise par un bouton vert interactif VR. Lorsqu'une main entre dans la zone d'interaction, un evenement de type *spawn* est declenche et demarre la generation progressive des pneus.

Les pneus apparaissent a une position predeterminee dans la scene. Une coroutine permet d'etaler leur instanciation dans le temps afin d'obtenir un flux regulier et lisible pour le joueur. Chaque pneu instancie recoit automatiquement deux proprietes booleennes aleatoires :

— `isCorrect` : indique si le pneu est conforme ;

— `isCodeBarreDetected` : indique si son code est detectable automatiquement.

FIGURE 2.2 – Console avec les boutons de génération et suppression des pneus.

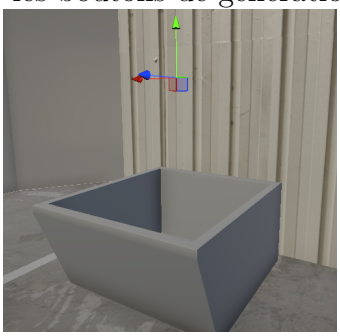


FIGURE 2.3 – Zone d'apparition des pneus.

Le démarrage de l'épreuve déclenche également un chronomètre. Un bouton rouge complète le dispositif en offrant une remise à zéro rapide de la scène (suppression des pneus présents).

2.2.2 Détection, scan et retour utilisateur

Deux détecteurs coexistent dans l'exercice : un détecteur manuel et un détecteur automatique. Chacun dispose d'une zone *trigger* et d'un script de lecture des entrées. Selon le type de détection, des événements distincts sont émis.

Lors d'un scan manuel, le système lit directement la conformité du pneu et met à jour l'écran de console avec une information visuelle adaptée. Pour le scan automatique, la logique ajoute un contrôle préalable de la lisibilité du code-barres. Si le code n'est pas détectable, un état d'erreur spécifique est affiché.

Cette architecture offre un avantage important : la séparation entre acquisition des données (détecteurs), logique métier (gestionnaire), et restitution (console). Le comportement est plus facile à maintenir, à tester, et à étendre.

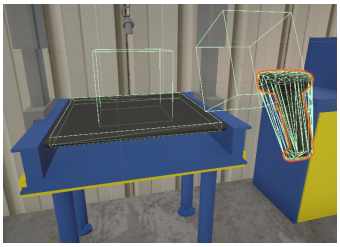


FIGURE 2.4 – Détecteur manuel et automatique.

FIGURE 2.5 – Affichage de la console avec les résultats du scan.

2.2.3 Condition de victoire et persistance des scores

Le tri final est validé par des zones de dépôt (bennes), chacune associée à une détection d'entrée/sortie. Lorsqu'un pneu est déposé dans la bonne benne, le compteur de réussite est incrémenté ; s'il en ressort, le compteur est décrémenté. La victoire est atteinte lorsque 10 pneus sont correctement placés.

Une fois la victoire atteinte, un clavier virtuel est affiché pour saisir un nom ou pseudo. Le score (temps réalisé) est ensuite enregistré dans un dictionnaire, puis sauvegardé en local dans un fichier JSON. Ce choix permet de conserver un historique entre les sessions. Une liste triée des scores est finalement affichée, du meilleur au moins bon résultat.

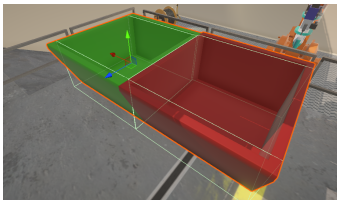


FIGURE 2.6 – Zone de dépôt des pneus.

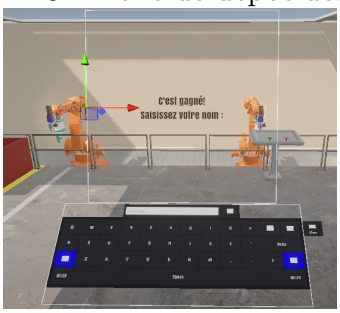


FIGURE 2.7 – Affichage des scores.

2.2.4 Habillage sonore

Le module intègre plusieurs sons spatialisés reliés aux événements de jeu : apparition des pneus, bip de scan, collisions, appui sur les boutons et son de victoire. L'audio renforce la compréhension de l'action en cours et l'immersion globale. La spatialisation est particulièrement utile pour associer chaque son à son origine dans la scène.

2.3 Module 2 : simulateur de grue

2.3.1 Conception mécanique et interactions XR

Le simulateur de grue repose sur un assemblage de joints physiques (*fixed joints* et *configurable joints*) permettant des rotations contrôlées sur des axes précis. Cette décomposition mécanique reproduit le comportement attendu d'un bras articulé : mouvements verticaux, rotation autour de la base et orientation de l'effecteur.

L'utilisateur agit au moyen de deux poignées VR saisissables, associées aux mains gauche et droite. Cette contrainte bimanuel est pertinente pédagogiquement, car elle simule une coordination motrice proche d'une tâche réelle de conduite d'engin.

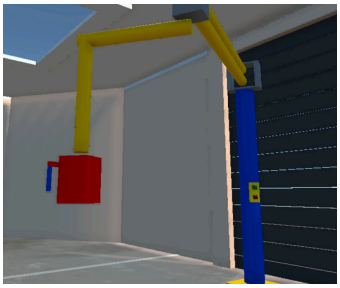


FIGURE 2.8 – Déplacement de la grue via les poignées VR.

2.3.2 Génération du parcours et calcul du score

Le parcours est activé via un bouton vert et désactivé via un bouton rouge. La ligne de guidage est générée aléatoirement entre 5 et 10 segments, ce qui limite l'effet de mémorisation et encourage l'entraînement sur des configurations variées.

Le parcours comporte un bloc de début et un bloc de fin. Le bloc de début initialise le score et signale l'entrée correcte dans la ligne. Durant le trajet :

— lorsque l'effecteur reste sur la ligne, celle-ci est affichée en vert ;

— lorsque l'effecteur sort de la ligne, elle devient rouge et le score diminue progressivement.

Le bloc de fin met fin à l'épreuve et déclenche l'affichage du clavier virtuel pour l'enregistrement du score, avec une logique similaire à celle du module pneus.

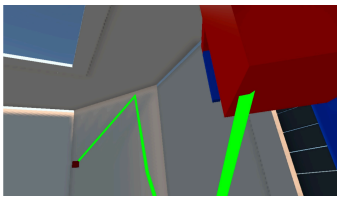


FIGURE 2.9 – Affichage de la ligne de guidage en vert.

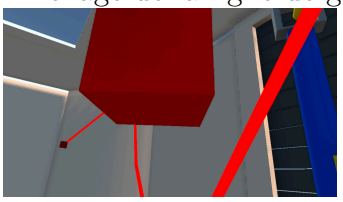


FIGURE 2.10 – Affichage de la ligne de guidage en rouge.

2.3.3 Retour sonore

Le module grue inclut un habillage sonore plus minimaliste, principalement axé sur le son de fin de parcours. Ce choix reste cohérent avec l'objectif de concentration motrice, mais pourrait être enrichi par des retours intermédiaires (alerte de sortie de ligne, confirmation de reprise de trajectoire, etc.).

2.4 Bilan global et perspectives

Le projet atteint ses objectifs principaux : proposer deux situations de formation VR interactives, reposant sur des mécanismes robustes (événements, triggers, persistance de données) et un retour utilisateur multi-modal. L'utilisation de scripts spécialisés et de points d'entrée clairs facilite la maintenance du code et la compréhension de la logique applicative.

Plusieurs axes d'amélioration ont toutefois été identifiés :

— **Stabilité physique des pneus** : certaines déformations apparaissent lors de la prise en main ;

— **Équilibrage de la préhension sur la grue** : la priorité de la poignée droite peut provoquer des comportements incohérents ;

— **Robustesse du système de score** : absence de majuscules et validation incomplète de la saisie dans certaines versions de build.

A moyen terme, des améliorations concrètes peuvent être envisagées : validation stricte des entrées clavier, meilleur traitement des doublons de pseudo, enrichissement des feedbacks sonores contextuels, et ajout d'indicateurs visuels d'aide à la correction en temps réel. Ces évolutions renforceraient à la fois la fiabilité technique et la valeur pédagogique du serious game.

Conclusion

Cette synthèse montre que le *Serious Game VR* constitue une base solide pour des scénarios de formation immersive. Les deux exercices sont complémentaires et exploitent efficacement les possibilités de la réalité virtuelle : manipulation naturelle, évaluation continue, et mémorisation des performances. Avec quelques ajustements techniques et ergonomiques, la solution peut évoluer vers un outil de formation encore plus précis, stable et adaptatif.

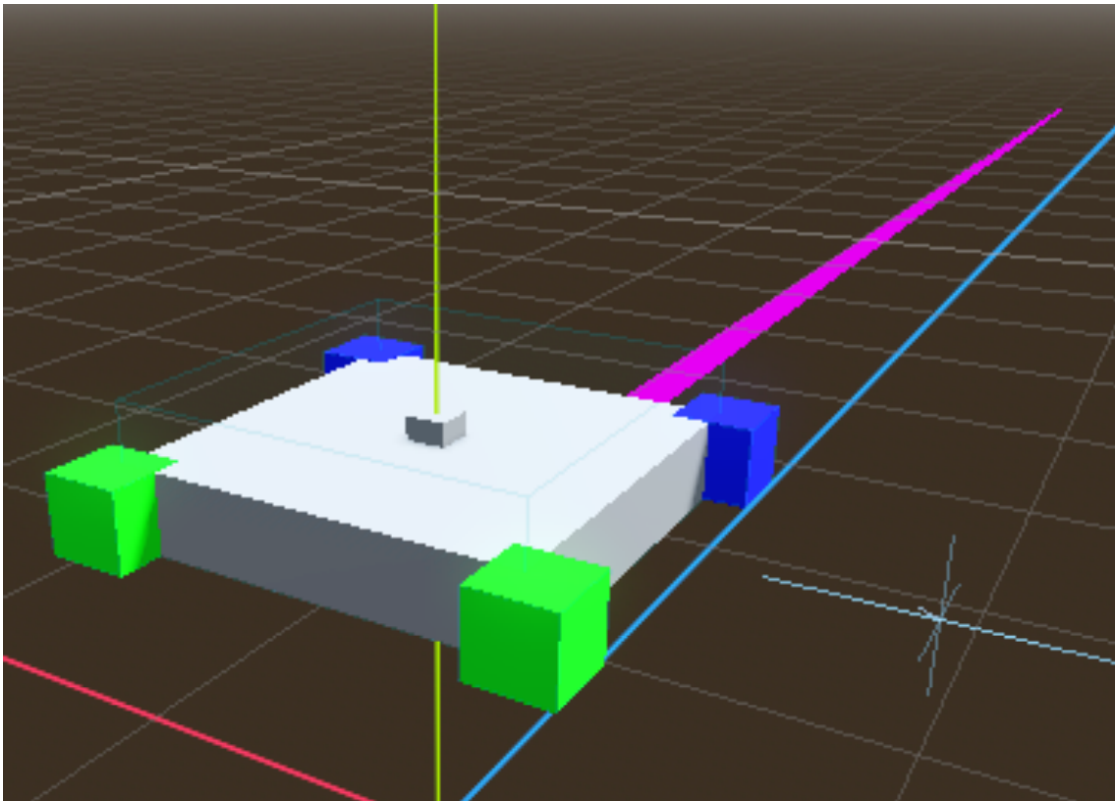
3 Simulateur de drône Godot - Pilotage Python (novembre 2024 - mars 2025)

Projet réalisé sur Godot dans un cadre pédagogique en binôme avec Elisa MARCHAND.

3.1 Synthèse du projet : Simulation et contrôle d'un drône

Ce projet a pour objectif de concevoir un simulateur de drône permettant aux étudiants d'apprendre la programmation temps réel dans un environnement interactif. Le simulateur remplace un ancien simulateur de voiture utilisé dans l'enseignement et propose un système plus riche, intégrant une simulation physique réaliste ainsi qu'une communication avec un programme externe.

Le simulateur est développé avec le moteur de jeu Godot, tandis que le contrôle du drône est effectué via un programme Python externe. L'application repose sur une architecture client–serveur permettant l'échange de données en temps réel entre la simulation et le programme de contrôle. L'utilisateur peut ainsi piloter un drône virtuel, récupérer ses informations d'état et implémenter ses propres algorithmes de contrôle.



3.2 Objectifs et cahier des charges

Le projet devait répondre à plusieurs objectifs principaux. Tout d'abord, il fallait créer un simulateur capable de représenter un drone évoluant dans un environnement 3D. Ce drone devait présenter un comportement physique réaliste, notamment grâce à la simulation des forces appliquées sur ses moteurs.

Ensuite, l'application devait permettre une communication avec un programme externe afin que les étudiants puissent écrire leurs propres algorithmes de contrôle. Cette communication devait être non bloquante afin de garantir une réactivité suffisante pour une application temps réel.

Enfin, l'ensemble devait permettre l'exécution de scénarios pédagogiques dans lesquels le drone doit se déplacer vers différentes positions tout en évitant des obstacles.

3.3 Architecture de la solution

L'application est organisée en trois composants principaux :

— le simulateur graphique développé avec Godot ;

— un serveur intégré dans l'application Godot ;

— un client Python chargé de piloter le drone.

Le simulateur représente l'environnement 3D dans lequel évolue le drone. Le serveur intégré reçoit les commandes envoyées par le programme Python et met à jour l'état du drone dans la simulation. Le client Python, quant à lui, implémente les algorithmes de contrôle et envoie des commandes au simulateur.

La communication entre le client et le serveur utilise le protocole UDP. Ce protocole a été choisi pour sa légèreté et sa rapidité, ce qui est particulièrement adapté aux systèmes temps réel. Les messages échangés utilisent le format JSON afin de faciliter leur structuration et leur interprétation.

Chaque message contient notamment un identifiant de client, un identifiant de message et un caractère indiquant le type d'action demandée (initialisation du drone, modification de la vitesse ou demande d'information).

3.4 Modélisation du drone

Le drone simulé est un quadrirotor composé de quatre moteurs et d'un corps central. Dans Godot, ce drone est modélisé sous la forme d'une scène contenant plusieurs objets physiques.

Chaque moteur est représenté par un `RigidBody3D` afin de bénéficier du moteur physique de Godot. Ces moteurs possèdent également un modèle visuel (`MeshInstance3D`) et une forme de collision (`CollisionShape3D`). Des jointures mécaniques permettent de relier les moteurs au corps principal du drone afin de former une structure cohérente.

Le drone est également équipé d'un capteur simulé à l'aide d'un `Raycast`. Ce capteur permet de détecter la présence d'obstacles sur la trajectoire du drone et de transmettre cette information au programme de contrôle.

Cette architecture modulaire facilite la manipulation des différents éléments du drone et permet d'ajouter facilement de nouveaux comportements ou capteurs.

3.5 Simulation physique et contrôle

Un des principaux défis du projet a été la mise en place d'un modèle physique permettant de simuler correctement le comportement du drone. Contrairement à une approche simplifiée où les forces seraient appliquées directement au corps du drone, chaque moteur génère ici une poussée indépendante.

Le contrôle du drone repose sur un système de régulation basé sur des contrôleurs PID (Proportionnel–Intégral–Dérivé). Ces contrôleurs permettent d'ajuster

dynamiquement la force appliquée par chaque moteur afin d'atteindre une vitesse cible.

Le système de régulation est organisé sous la forme d'un double PID :

— un premier PID calcule l'inclinaison nécessaire du drone pour atteindre la vitesse souhaitée ;

— un second PID ajuste les forces des moteurs afin d'atteindre cette inclinaison.

À chaque étape de simulation, le programme lit l'état actuel du drone, notamment sa vitesse et son orientation. L'erreur entre la vitesse actuelle et la vitesse cible est ensuite calculée, ce qui permet d'obtenir les composantes proportionnelle, intégrale et dérivée de la correction.

Ces valeurs sont utilisées pour déterminer les forces à appliquer à chaque moteur. La combinaison des forces permet d'incliner le drone dans la direction souhaitée ou de modifier son altitude.

Cette méthode permet d'obtenir un comportement stable et réaliste, tout en restant suffisamment flexible pour être utilisé dans différents scénarios.

3.6 Interface utilisateur

L'interface utilisateur du simulateur reste volontairement simple. Elle comporte principalement plusieurs barres de défilement permettant d'ajuster les coefficients des contrôleurs PID.

Ces paramètres peuvent être modifiés en temps réel afin d'observer leur influence sur le comportement du drone. Les valeurs des coefficients sont transmises aux scripts de contrôle via le système de signaux de Godot.

Cette approche facilite les expérimentations et permet aux étudiants de comprendre l'impact des paramètres de régulation sur la stabilité du système.

3.7 Communication entre Python et Godot

La communication entre le programme Python et le simulateur Godot constitue un élément central du projet.

Du côté du client Python, une classe dédiée gère l'envoi et la réception des messages. Les fonctions proposées permettent notamment :

— d'initialiser un drone dans la simulation ;

— d'envoyer un vecteur de vitesse cible ;

— de récupérer différentes informations sur l'état du drone.

Lorsqu'un message est envoyé, il est converti au format JSON puis transmis au serveur via UDP. Le serveur Godot reçoit ensuite ce message, le décode et l'ajoute dans une file d'attente.

Le traitement des messages est effectué dans un second thread afin d'éviter de bloquer la simulation. Chaque message est analysé afin de déterminer l'action à effectuer.

Trois types de messages principaux sont pris en charge :

— l'apparition d'un drone dans la simulation ;

— la modification de sa vitesse cible ;

— les demandes d'informations sur son état.

Une fois l'action exécutée, le serveur peut renvoyer une réponse au client Python, par exemple pour transmettre la position actuelle du drone ou l'état de son capteur.

3.8 Algorithme de pilotage

Le programme Python implémente également un algorithme de contrôle permettant au drone de suivre un parcours composé de plusieurs cibles.

Pour cela, le programme récupère régulièrement la position du drone et celle de la prochaine cible. Un contrôleur PID calcule ensuite un vecteur vitesse permettant de rapprocher le drone de cette cible.

Lorsque le drone se trouve suffisamment proche de la position cible, le programme demande la position de la cible suivante. Ce processus se répète jusqu'à ce que toutes les cibles aient été atteintes.

Le drone est également capable de détecter les obstacles présents sur sa trajectoire. Lorsque le capteur indique la présence d'un obstacle, une stratégie simple d'évitement est appliquée : le drone augmente temporairement son altitude afin de contourner l'objet.

3.9 Perspectives d'amélioration

Plusieurs pistes d'amélioration ont été identifiées pour la suite du projet.

Une première possibilité consiste à réimplémenter certaines parties du programme en C++ afin de comparer les performances avec la version actuelle en GDScript.

Il serait également envisageable d'intégrer un système d'exploitation temps réel tel que FreeRTOS afin de rapprocher davantage le projet d'un système embarqué réel.

D'autres améliorations plus modestes pourraient être apportées, comme l'ajout de nouveaux scénarios, la création de cartes supplémentaires ou la mise en place de zones interdites dans l'environnement de simulation.

3.10 Conclusion

Ce projet a permis de développer un simulateur de drone réaliste capable d'être contrôlé par un programme externe. L'utilisation du moteur Godot a facilité la création d'un environnement interactif et la simulation physique du drone.

La mise en place d'une communication client–serveur avec Python permet aux utilisateurs d'implémenter leurs propres algorithmes de pilotage, ce qui constitue un outil pédagogique intéressant pour l'apprentissage de la programmation temps réel.

Grâce à la modélisation physique des moteurs et à l'utilisation de contrôleurs PID, le drone présente un comportement cohérent et stable. Le projet constitue ainsi une base solide pour de futurs développements et pour son utilisation dans un contexte pédagogique.

4 Sport 2 Scroll - S2S

4.1 Présentation du projet

`https://florian-boisset.itch.io/sport2scroll`

Sport2Scroll est un projet d'application mobile de fitness gamifiée, ayant pour objectif d'utiliser le body tracking pour aider les utilisateurs à effectuer leurs exercices correctement. Un second objectif du projet, est d'aider les utilisateurs à maitriser leur temps d'écran en débloquant du temps de "scroll" par le sport. L'utilisateur définit son niveau initial, ses objectifs, les exercices qu'il souhaite faire en priorité, et le temps de scroll limite qu'il s'autorise. Une fois son temps d'écran dépassé, l'application prend le devant sur les applications de réseaux sociaux, et propose à l'utilisateur de faire du sport pour débloquer du temps de scroll.

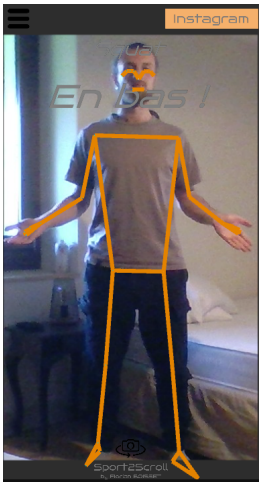
4.2 Fonctionnement technique

L'application est développée sous **Unity 6** pour Android.

Détection corporelle avec MediaPipe

La détection du corps repose sur **MediaPipe Pose Landmarker**, une bibliothèque open-source de Google spécialisée dans l'analyse de la posture humaine en temps réel. À chaque image captée par la caméra frontale, MediaPipe analyse le flux vidéo et retourne la position de **33 points clés** répartis sur le corps : épaules,

coudes, poignets, hanches, genoux, chevilles, etc. Chaque point est fourni avec un score de confiance indiquant à quel point MediaPipe est certain de sa position.



Définition des exercices

Pour définir un exercice, un **outil éditeur sur mesure** développé dans Unity permet, en mode éditeur, de se placer devant la caméra, d'effectuer la pose souhaitée et de la **capturer en un clic**. L'outil enregistre les angles de toutes les articulations à cet instant. On ajoute également une description ou indication pour l'utilisateur de la pose a adopter.

Un exercice est ensuite défini par une **séquence ordonnée de poses de référence**. Par exemple, une pompe est décrite par deux poses : bras tendus au sol, puis bras fléchis (position basse). Lors d'une session, l'application demande à l'utilisateur de reproduire chaque pose de la séquence dans l'ordre ; lorsque toutes ont été validées, une répétition est comptabilisée.

Les exercices actuellement disponibles sont :

— Pompes

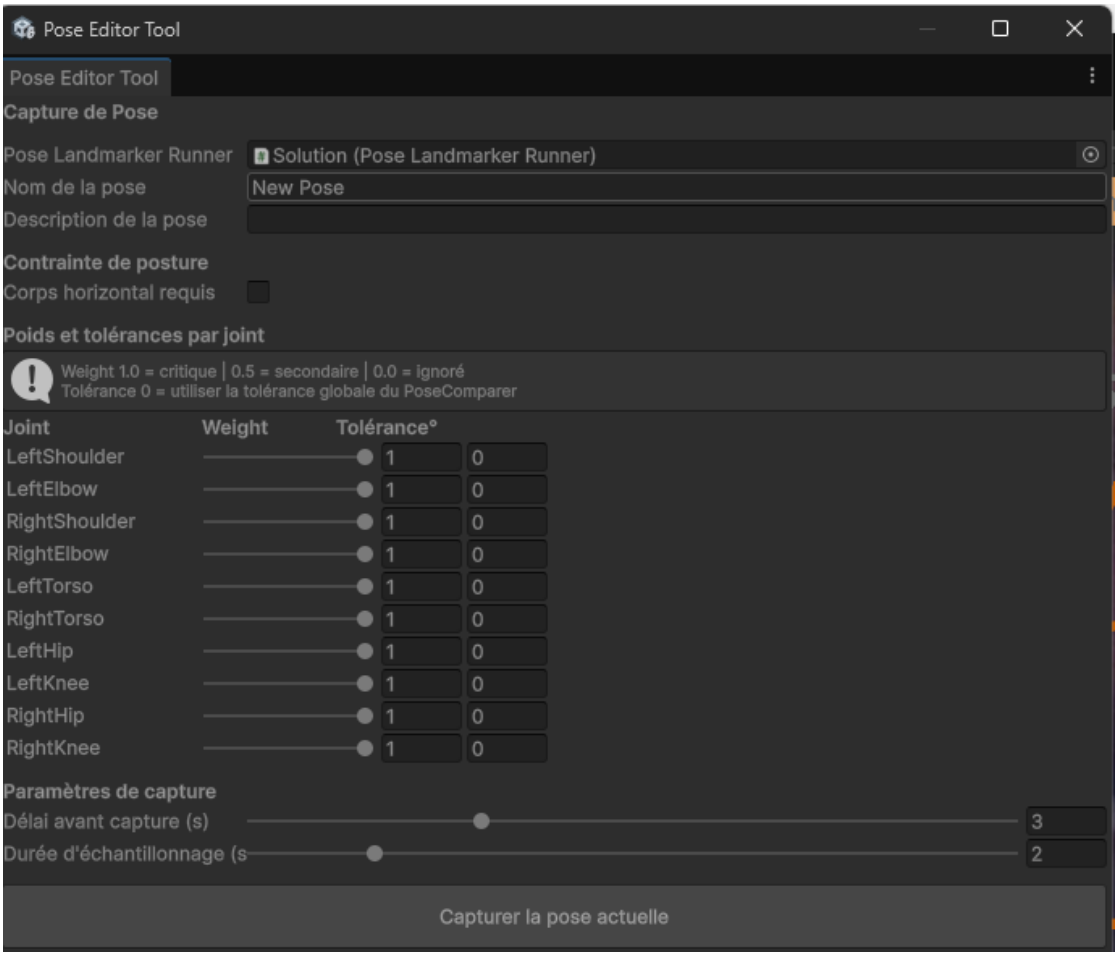
— Squats

— Genoux-Coudes

— Mountain Climbers

— Jumping Jacks

— Fentes



Validation d'une pose

Pour chaque image de la caméra, l'application calcule les **angles formés aux articulations** (par exemple, l'angle du coude = angle entre le bras et l'avant-bras) et les compare à ceux enregistrés dans la pose de référence. Chaque articulation contribue à un **score global de 0 à 100**, avec un poids configurable selon son importance dans l'exercice : par exemple, les coudes sont fortement pondérés pour une pompe mais peu importants pour un squat. Dès que le score dépasse un seuil défini, la pose est validée et l'application passe à la suivante dans la séquence. Un retour visuel en temps réel indique la qualité de la pose : la couleur passe du **rouge** (pose incorrecte) au **vert** (pose atteinte).


Instagram
Squat
En haut !

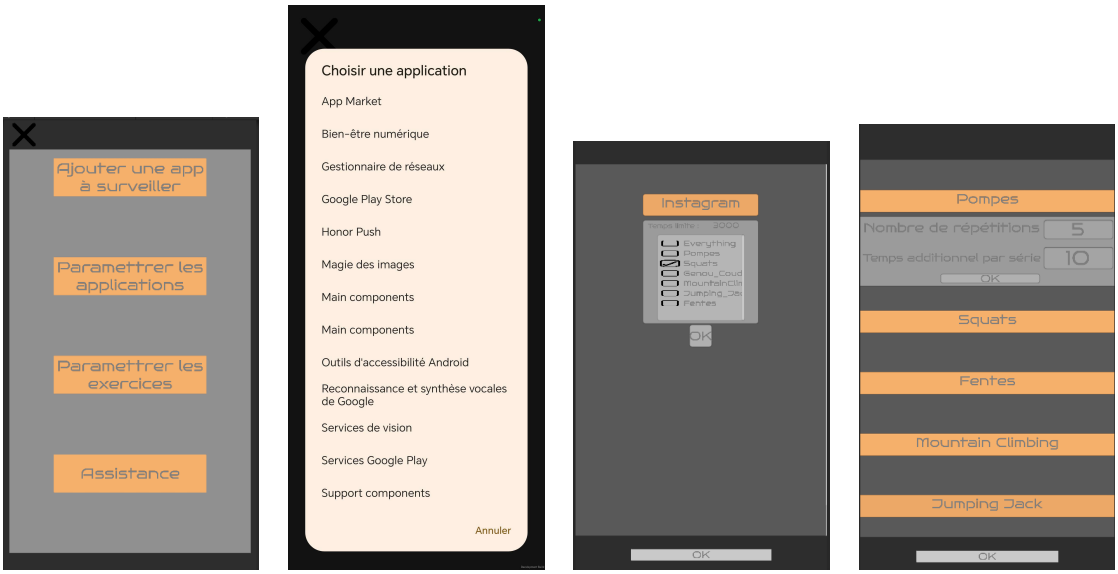
Paramétrage des exercices et des applications surveillées

Les exercices et les applications surveillées sont présentés sous forme de **listes dans l'interface**. Chaque élément expose ses paramètres directement via des **champs de texte modifiables** dans la liste. Pour les exercices, l'utilisateur règle :

— le **nombre de répétitions** à effectuer par série ;

— le **temps de scroll débloqué** à l'issue de la série.

Pour les applications surveillées, il configure le **quota quotidien** alloué à chacune. Les compteurs se remettent automatiquement à zéro chaque jour à minuit.



Gestion du temps d'écran

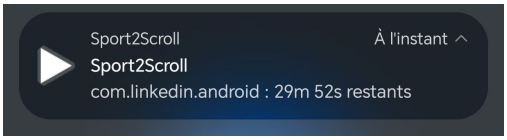
Le suivi du temps passé sur les applications repose sur deux mécanismes complémentaires :

— L'**API UsageStats** d'Android fournit le temps d'utilisation cumulé de chaque application depuis le début de la journée ;

— Un **service d'accessibilité** écoute en temps réel les événements système : ouverture d'une application surveillée, fermeture, et **verrouillage de l'écran**. Ce dernier point est crucial : si l'utilisateur éteint son écran sans fermer l'application, le compteur est mis en pause, évitant ainsi de comptabiliser du temps où l'application n'est pas réellement utilisée.

Dès qu'une application surveillée est détectée ouverte, une **notification persistante** s'affiche et se met à jour en temps réel avec le temps d'utilisation restant

pour cette application. Dès que le quota est épuisé, une **superposition** (overlay) s'affiche par-dessus l'application, invitant l'utilisateur à faire du sport pour débloquer du temps supplémentaire.



Cinq permissions Android sont nécessaires au fonctionnement de l'application :

— **Caméra** (*CAMERA*) : accès à la caméra frontale pour la détection de pose ;

— **Statistiques d'utilisation** (*PACKAGE_USAGE_STATS*) : lecture du temps cumulé par application. Permission sensible, accordée manuellement dans les paramètres système ;

— **Service d'accessibilité** (*BIND_ACCESSIBILITY_SERVICE*) : détection en temps réel de l'ouverture, de la fermeture des applications et du verrouillage de l'écran. Accordée manuellement dans les paramètres d'accessibilité ;

— **Affichage par-dessus d'autres applications** (*SYSTEM_ALERT_WINDOW*) : affichage de l'overlay de blocage et des notifications au premier plan. Accordée manuellement via les paramètres ;

— **Ignorer l'optimisation de la batterie** (*REQUEST_IGNORE_BATTERY_OPTIMIZATI*
empêche Android de suspendre le service de surveillance en arrière-plan pour économiser la batterie, garantissant un suivi continu même lorsque l'application est en veille.