

Presentation of projects produced with game engines

Florian BOISSET

[Itch.io Gallery](#)

[About me](#)

August 17, 2026

Abstract

This document aims to present and showcase my work as a developer. My projects are built using game engines, and I wish to share them to demonstrate my skills and experience in this field. For each project, I will present the development context, the project objectives, the technologies used, the challenges encountered and the solutions developed. I will also highlight the results obtained and the features implemented. For published projects, I will provide links to distribution platforms or online demos. Numerous screenshots and videos are included to illustrate the work and provide a clearer understanding of each project.

Contents

1	AGAGAB - A Game About Growing A Bean	3
1.1	Game overview	3
1.2	Game content	4
1.3	Development	7
	Internship Report Summary at Assystem – Virtual Reality Project for the Nuclear Industry	10
1.4	Serious Game in Virtual Reality (March 2025)	10
1.5	Context	10
1.6	Technical achievements	11
2	Serious Game in Virtual Reality (March 2025)	14
2.1	Presentation and context	14
2.2	Module 1: tyre sorting simulation	15
2.3	Module 2: crane simulator	18
2.4	Overall assessment and outlook	20
3	Godot Drone Simulator – Python Control (November 2024 – March 2025)	21
3.1	Project summary: Drone simulation and control	21
3.2	Objectives and specifications	22
3.3	Solution architecture	22
3.4	Drone modelling	23
3.5	Physical simulation and control	23
3.6	User interface	24
4	Sport 2 Scroll - S2S	25
4.1	Project overview	25
4.2	Technical design	25

5	Combo Survivor	31
5.1	Game overview	31
5.2	Game content	31
6	ThousandsPong	33
6.1	Game overview	33
6.2	Game rules	33
6.3	Server authority	34
6.4	Server infrastructure and security	34
6.5	Download	35

1 AGAGAB - A Game About Growing A Bean

1.1 Game overview

AGAGAB is the first game whose development went all the way to publication. It is a game in which the player's objective is to grow a bean plant until it reaches their home, which has floated away onto a floating island. Government agents will try to get in their way. AGAGAB was developed using Unity and was published on Itch.io in March 2026. It continues to be updated regularly.

The game's Itch.io page!

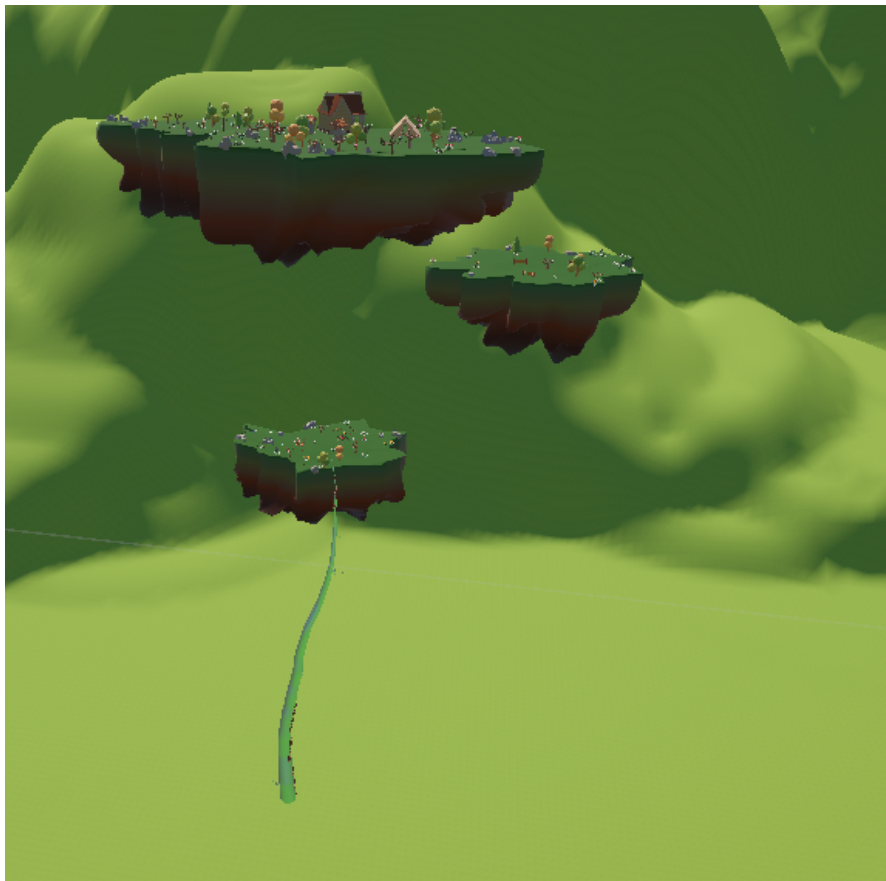


Figure 1.1: Screenshot showing floating islands and a growing plant.

1.2 Game content

1.2.1 The map

Floating islands in the sky make up the game environment. The player must grow their bean from the ground level to reach the islands, where they can find items to unlock that allow them to reach higher and higher islands, until they reach their home on the highest island.



Figure 1.2: Screenshot of the islands viewed from the ground.

1.2.2 The plant

The player grows a bean by watering it and adding fertiliser. They plant a seed in the ground, on a floating island, or on another plant to grow a new bean. The bean produces two types of fruit: the first serves as currency, the second, much rarer, serves as a seed for replanting.

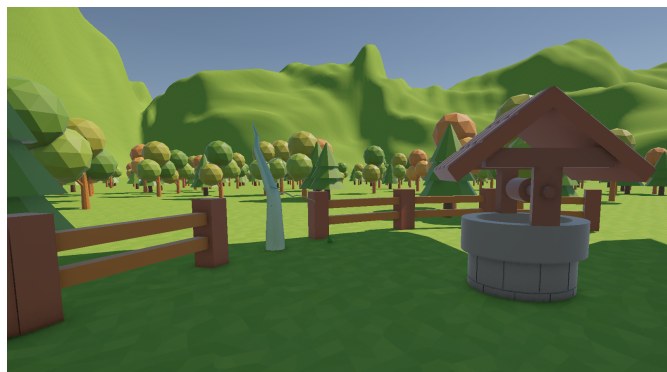


Figure 1.3: The plant growing at ground level.

1.2.3 Ladders

To climb vertical plants, the player must place ladders on them. As the plants grow, they engulf the ladders, forcing the player to place new ones.

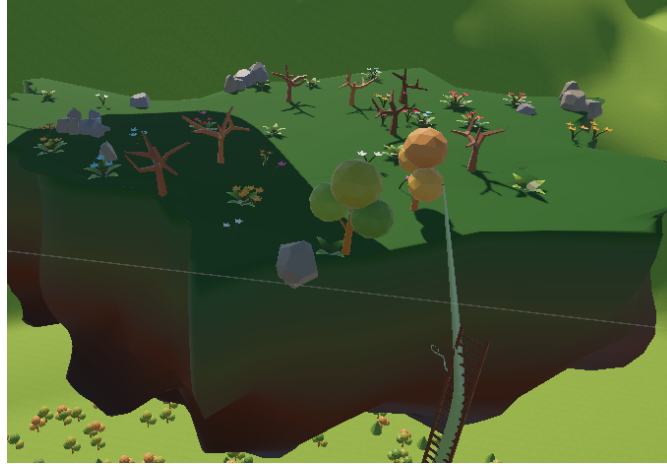


Figure 1.4: The plant reaching an island level with a ladder placed.

1.2.4 The shop

A shop allows the player to buy consumables such as ladders, fertilisers or ammunition.



Figure 1.5: The shop where the player can purchase consumables.

1.2.5 Enemies

Spies first come in for reconnaissance. If they stay alive long enough and make enough reports, soldiers will appear and target the player. The player has weapons to defend themselves, but must first unlock them by finding weapon licences hidden on the floating islands.



Figure 1.6: Spies on reconnaissance.

1.2.6 Save system

The player can save their progress to reload their game later. To do so, they must go to a save point (tent or hut) and interact with the building.



Figure 1.7: Screenshots of the save and load menus.

1.3 Development

1.3.1 The map

Floating islands are generated **procedurally**. Starting from a unit UV sphere, it is then transformed. First, an **elongation** is applied to give the island a more ellipsoidal shape. Desynchronised Perlin noise (with offsets) is used to warp the coordinate space (**Domain Warping**) to give a more organic shape. The points are displaced using **Fractional Brownian Motion**, whose persistence and frequency are controlled to add more or less detail to the island. Finally, the top of the island is flattened and the bottom is stretched. A shader colours the island according to triangle altitude (brown at the bottom, green at the top), avoiding the need to texture each island individually.

Objects are then placed on the island:

- On the highest island, the house (final objective) is placed;
- Objects from the list are sorted by tag (plant, rock, building...);

- Triangles from the island’s surface mesh are randomly selected;
- Triangles are filtered by slope, distance to other triangles, etc.;
- A category and an item within the category are randomly drawn.

1.3.2 The plant

Plants are generated procedurally. An **influenced Perlin Worm** is used to generate a plant shape, which is then converted into a series of **segments**. Each segment is made up of a cylinder. As the plant grows, the segment cylinders are enlarged, pushing the following segments forward. Segments are activated once a certain diameter is reached. Several plant generation methods were tested; the chosen approach consists of **generating a pool of plants at game launch**, which are placed in the scene when the player plants a seed.

1.3.3 Items

An item manager handles the internal state of the various initialised items (Locked, Unlocked, Equipped, Selected). A regular **RayCast** checks for the presence of an item to equip or unlock in front of the player. An **item class** defines the standard behaviours and variables for items, and is **inherited by the different item types** (quantity, price, description for the shop, etc.). Items are **directly subscribed to certain player inputs** (primary action, secondary, interaction), making it easy to add new items to the game. These **subscriptions are temporarily revoked**, for example when the player is facing the shop, to allow clicking without activating the item.

- Ladders come from an Asset Store package that was slightly modified then integrated.
- The behaviours of the other items were built from scratch.
- Their 3D models are mostly sourced from the Asset Store; a few were modelled in Blender.

1.3.4 The shop

The shop is a UI on a world-space canvas that responds to a virtual cursor placed at the centre of the screen. At regular intervals, items to be sold and a price multiplier are randomly drawn. A UI item template is then filled with this information. A second info window displays the details of the selected item. Unity’s Buttons,

ScrollBar and Panel components are used to build the interface and detect interactions. When the player is looking at the shop, the item manager's subscription to the item-switch input is revoked to allow scrolling.

Internship Report Summary at Assystem – Virtual Reality Project for the Nuclear Industry

1.4 Serious Game in Virtual Reality (March 2025)

Position: Virtual Reality Development Engineer

Company: Assystem — Venelles office (Bouches-du-Rhône)

Duration: 6 months (March – August 2025)

ISIMA track: Interactive Systems for Embedded and Virtual Environments

Main technologies: Unity, C#, Python, Pixyz, HTC VIVE, CATIA

Application domain: Nuclear industry — ITER project / CEA Cadarache

1.5 Context

1.5.1 The company

Assystem is an international engineering group specialising in the management of technology-intensive projects (nuclear energy, critical infrastructure, transport, healthcare). The Venelles office contributes in particular to the **ITER** (*International Thermonuclear Experimental Reactor*) project, an experimental nuclear fusion reactor located in Cadarache, and supports **CEA Cadarache** on several nuclear safety projects.

1.5.2 Internship objectives

Faced with complex, hazardous or physically inaccessible industrial environments, Assystem made the strategic decision to integrate **Virtual Reality (VR)** into its engineering processes. VR enables the simulation of human interventions (assembly, maintenance, dismantling) in order to:

- assess the **ergonomic feasibility** of technical tasks in confined spaces;
- quantify the **radiological dose** received by operators during each scenario;
- reduce design errors before any physical manufacturing.

The central objective of the internship was to make these simulations **accessible to non-developer engineers**, by reducing preparation time and industrialising Unity workflows.

1.6 Technical achievements

1.6.1 Unified Unity template

The two existing projects (ergonomics and dosimetry) were maintained separately, causing asset duplication and costly context switching. I **merged these projects** into a single generic template activatable by modules, reducing disk usage and simplifying version management.

Integration of the student project (Centrale Lyon): an object manipulation module (instantiation, translation/rotation, labelling) developed by students under an emulator was adapted to Assystem’s HTC VIVE controllers, correcting UI/UX incompatibilities.

Instantiable tools: development of a Unity editor tool allowing a 3D tool (FBX format) to be added to a simulation in a few clicks — automatic assignment of collisions, gravity and VR grabbing, generation of an identifying icon — with no Unity expertise required.

1.6.2 Scene editing and preparation tools

- **CamViewTool:** spectator camera control interface from the Unity inspector (orbital camera, free keyboard mode, fixed cameras, automatic selection by counting visible operator pixels). Integrates REBA analysis activation.
- **PrefabCleaner:** automatic cleaning tool for Prefabs imported via Pixyz — removal of missing scripts, conversion of materials to URP/Lit, application of a default material — to eliminate blocking errors and warnings without Unity expertise.
- **Volume Filter Tool:** automatic removal of small 3D elements (screws, bolts, etc.) below a volume threshold defined by reference, significantly lightening scenes.

1.6.3 REBA Calculator in Virtual Reality

REBA (*Rapid Entire Body Assessment*) is an ergonomic analysis method that evaluates the musculoskeletal risk of a posture by considering the entire body (trunk, neck, legs, arms, forearms, wrists, loads). To my knowledge, no commercial plugin offers a turnkey REBA implementation in Unity.

I developed a **native Unity REBA calculator**:

- Referencing of body segments via the VR mannequin inspector;
- Measurement of joint angles at each frame by vector computation between parent/child segments;
- Calibration function on neutral posture to correct initial morphology;
- Application of REBA scoring rules per segment (scores +1 for inclinations/rotations);
- **Bilateral** evaluation (left and right) retaining the most conservative score;
- Checkboxes to input lifted load and grip quality;
- Real-time visualisation of the coloured skeleton and global score in the supervision interface.

1.6.4 Radiation module – dosimetry in VR

- **File reformatting**: development of a Python script automatically converting two formats of radiological maps (data provided by other teams) to the Unity-compatible format. Addition of point cloud decimation to limit rendering latency.
- **Zone placement tool**: new Unity component allowing the radiation point cloud to be materialised, moved and resized interactively in the scene, filling the complete absence of any repositioning tool in the existing system.

1.6.5 CAD → Unity pipeline via Pixyz

CATIA models (3DXML format) are not directly compatible with Unity; Pixyz is used as a conversion and optimisation layer. I defined a **pre-configured Unity project** with Pixyz Rule Engines adapted to two use cases:

- **Manipulable objects**: high geometric precision + precise colliders;
- **Passive elements** (structures, pipes): heavy simplification for performance.

The LOD (*Levels of Detail*) system was evaluated and then discarded, as its per-model configuration was incompatible with the goal of rapid, standardised preparation.

2 Serious Game in Virtual Reality (March 2025)

Project carried out in a pair using Unity in virtual reality for a Meta Quest 2.
Participants: Florian BOISSET and Elisa MARCHAND.

2.1 Presentation and context

This section presents a synthesis of a project report produced in an educational setting for the development of a *Serious Game in VR*. The project aims to design two immersive VR training experiences: a tyre-sorting simulation and a crane simulator. Both modules were built around an event-driven architecture, XR-tool-based user interaction, and a persistent score system. This synthesis highlights the technical choices, interaction flow, success criteria, as well as observed limitations and areas for improvement.

The objective of this serious game is to offer situations close to real professional settings. The main goals were:

- to design natural interactions in virtual reality;
- to structure object and interface behaviours via events;
- to provide immediate feedback to the user (visual, audio and score);
- to record performance data to track progression from one session to the next.

The system consists of two complementary exercises. The first assesses the user's ability to identify and sort tyres by conformity. The second evaluates the precision of crane operation along a variable course. Together, these two modules cover skills of diagnosis, gestural coordination and operational control.

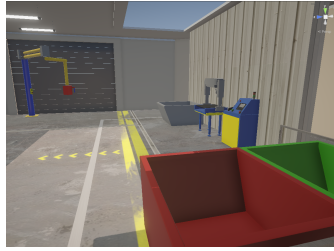


Figure 2.1: Overview of the VR scene.

2.2 Module 1: tyre sorting simulation

2.2.1 Tyre generation and management

The core of the module is centralised in a dedicated manager (*TireManager.cs*). The exercise is started by an interactive green VR button. When a hand enters the interaction zone, a *spawn* event is triggered and progressive tyre generation begins.

Tyres appear at a predetermined position in the scene. A coroutine spreads their instantiation over time to produce a regular, readable flow for the player. Each instantiated tyre automatically receives two random boolean properties:

- **isCorrect**: indicates whether the tyre is conformant;
- **isCodeBarreDetected**: indicates whether its barcode is automatically detectable.



Figure 2.2: Console with buttons to spawn and remove tyres.

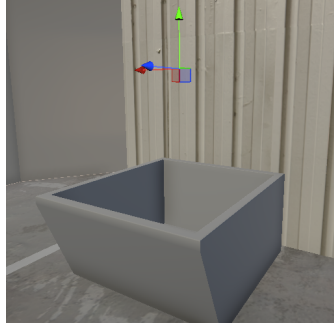


Figure 2.3: Tyre spawn zone.

Starting the exercise also triggers a stopwatch. A red button completes the setup by providing a quick scene reset (removal of all present tyres).

2.2.2 Detection, scanning and user feedback

Two detectors coexist in the exercise: a manual detector and an automatic detector. Each has a *trigger* zone and an input reading script. Depending on the detection type, distinct events are emitted.

During a manual scan, the system reads the tyre's conformity directly and updates the console screen with appropriate visual information. For automatic scanning, the logic adds a prior check on barcode readability. If the barcode is not detectable, a specific error state is displayed.

This architecture offers a key advantage: the separation between data acquisition (detectors), business logic (manager), and feedback (console). The behaviour is easier to maintain, test and extend.

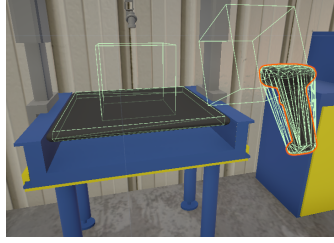


Figure 2.4: Manual and automatic detectors.



Figure 2.5: Console display with scan results.

2.2.3 Win condition and score persistence

The final sorting is validated by deposit zones (bins), each associated with an entry/exit detection. When a tyre is placed in the correct bin, the success counter is incremented; if it is removed, the counter is decremented. Victory is reached when 10 tyres are correctly placed.

Once victory is achieved, a virtual keyboard is displayed to enter a name or alias. The score (time achieved) is then saved in a dictionary and stored locally in a JSON file. This approach allows a history to be maintained between sessions. A sorted list of scores is displayed, from best to worst.

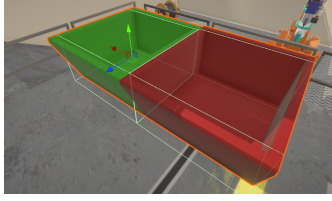


Figure 2.6: Tyre deposit zone.

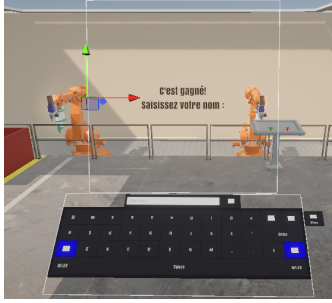


Figure 2.7: Score display.

2.2.4 Sound design

The module integrates several spatialised sounds tied to game events: tyre spawn, scan beep, collisions, button presses and victory sound. Audio reinforces the understanding of the current action and overall immersion. Spatialisation is particularly useful for associating each sound with its origin in the scene.

2.3 Module 2: crane simulator

2.3.1 Mechanical design and XR interactions

The crane simulator is based on an assembly of physical joints (*fixed joints* and *configurable joints*) enabling controlled rotations on precise axes. This mechanical decomposition reproduces the expected behaviour of an articulated arm: vertical movements, rotation around the base and end-effector orientation.

The user operates via two graspable VR handles, one for each hand. This bimanual constraint is pedagogically relevant as it simulates motor coordination close to a real crane operation task.

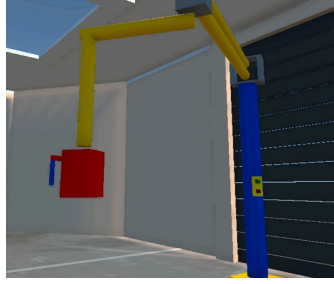


Figure 2.8: Crane movement using VR handles.

2.3.2 Course generation and score calculation

The course is activated with a green button and deactivated with a red button. The guiding path is randomly generated with between 5 and 10 segments, limiting memorisation effects and encouraging training on varied configurations.

The course has a start block and an end block. The start block initialises the score and signals correct entry into the path. During transit:

- when the end-effector stays on the path, it is displayed in green;
- when the end-effector leaves the path, it turns red and the score decreases progressively.

The end block concludes the exercise and triggers the display of the virtual keyboard for score recording, with logic similar to the tyre module.

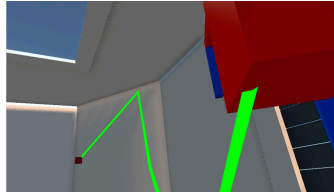


Figure 2.9: Guiding path displayed in green.

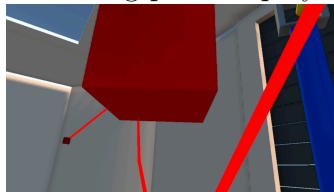


Figure 2.10: Guiding path displayed in red.

2.3.3 Sound feedback

The crane module includes a more minimal sound design, mainly focused on the end-of-course sound. This choice remains consistent with the objective of motor concentration, but could be enriched with intermediate feedback (path-exit alert, trajectory resumption confirmation, etc.).

2.4 Overall assessment and outlook

The project achieves its main objectives: offering two interactive VR training situations, built on robust mechanisms (events, triggers, data persistence) and multi-modal user feedback. The use of specialised scripts and clear entry points facilitates code maintenance and understanding of the application logic.

Several areas for improvement were nonetheless identified:

- **Tyre physics stability:** some deformations appear when grasping;
- **Crane grip balancing:** right-handle priority can cause inconsistent behaviour;
- **Score system robustness:** absence of uppercase conversion and incomplete input validation in some build versions.

In the medium term, concrete improvements can be considered: strict keyboard input validation, better handling of duplicate aliases, enrichment of contextual sound feedback, and addition of real-time visual correction indicators. These developments would strengthen both the technical reliability and the pedagogical value of the serious game.

Conclusion

This synthesis shows that the *VR Serious Game* constitutes a solid foundation for immersive training scenarios. The two exercises are complementary and effectively exploit the possibilities of virtual reality: natural manipulation, continuous assessment and performance tracking. With a few technical and ergonomic adjustments, the solution can evolve into an even more precise, stable and adaptive training tool.

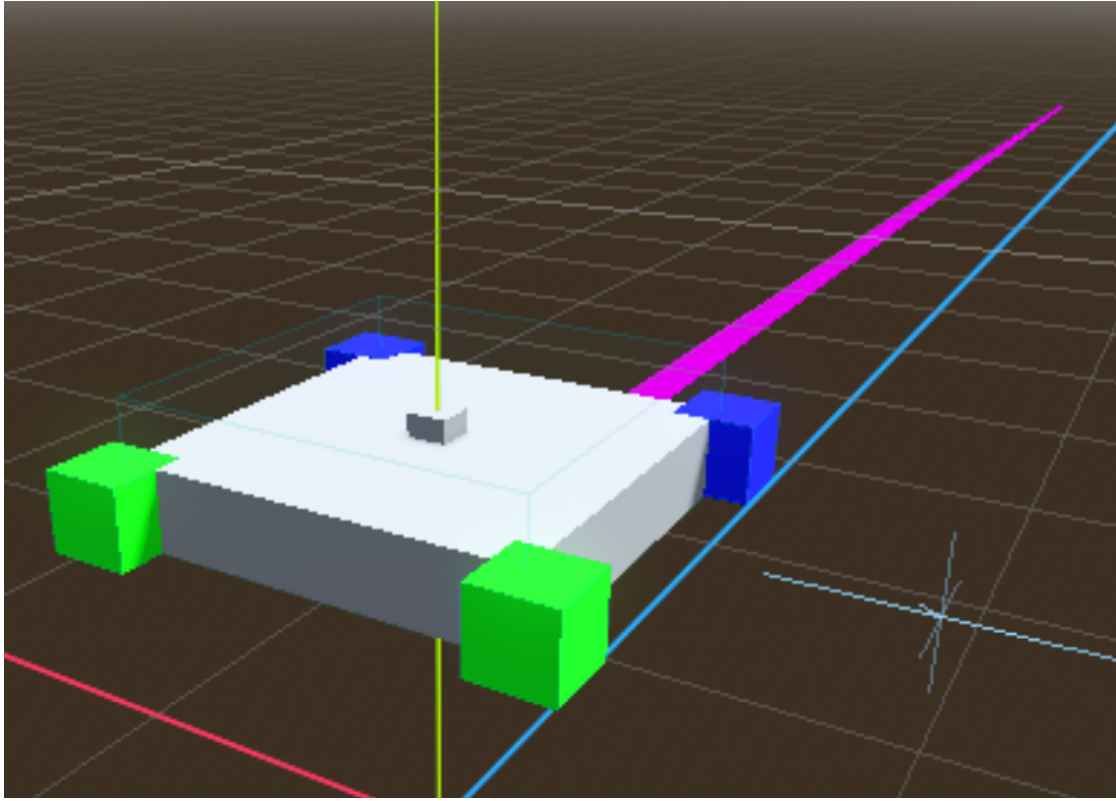
3 Godot Drone Simulator – Python Control (November 2024 – March 2025)

Project carried out in Godot in an educational setting, in a pair with Elisa MARC-HAND.

3.1 Project summary: Drone simulation and control

This project aims to design a drone simulator that allows students to learn real-time programming in an interactive environment. The simulator replaces an older car simulator used in teaching and offers a richer system, integrating realistic physical simulation as well as communication with an external programme.

The simulator is developed using the Godot game engine, while drone control is handled via an external Python programme. The application relies on a client–server architecture enabling real-time data exchange between the simulation and the control programme. The user can thus pilot a virtual drone, retrieve its state information and implement their own control algorithms.



3.2 Objectives and specifications

The project had to meet several main objectives. First, a simulator capable of representing a drone evolving in a 3D environment had to be created. This drone needed to exhibit realistic physical behaviour, in particular through the simulation of forces applied to its motors.

Next, the application had to allow communication with an external programme so that students could write their own control algorithms. This communication had to be non-blocking in order to guarantee sufficient responsiveness for a real-time application.

Finally, the whole system had to allow the execution of educational scenarios in which the drone must move to various positions while avoiding obstacles.

3.3 Solution architecture

The application is organised into three main components:

- the graphical simulator developed with Godot;

- a server embedded in the Godot application;
- a Python client responsible for piloting the drone.

The simulator represents the 3D environment in which the drone operates. The embedded server receives commands sent by the Python programme and updates the drone’s state in the simulation. The Python client implements the control algorithms and sends commands to the simulator.

Communication between the client and server uses the UDP protocol. This protocol was chosen for its lightness and speed, which is particularly suited to real-time systems. The exchanged messages use the JSON format to facilitate their structuring and interpretation.

Each message contains in particular a client identifier, a message identifier and a character indicating the type of action requested (drone initialisation, speed modification or status request).

3.4 Drone modelling

The simulated drone is a quadrotor composed of four motors and a central body. In Godot, this drone is modelled as a scene containing several physical objects.

Each motor is represented by a `RigidBody3D` in order to benefit from Godot’s physics engine. These motors also have a visual model (`MeshInstance3D`) and a collision shape (`CollisionShape3D`). Mechanical joints connect the motors to the drone’s main body to form a coherent structure.

The drone is also equipped with a simulated sensor using a `Raycast`. This sensor detects the presence of obstacles on the drone’s trajectory and transmits this information to the control programme.

This modular architecture facilitates manipulation of the different drone elements and makes it easy to add new behaviours or sensors.

3.5 Physical simulation and control

One of the main challenges of the project was setting up a physical model capable of correctly simulating the drone’s behaviour. Unlike a simplified approach where forces would be applied directly to the drone body, each motor here generates an independent thrust force.

Drone control relies on a regulation system based on PID controllers (Proportional–Integral–Derivative). These controllers dynamically adjust the force applied by each motor to reach a target speed.

The regulation system is organised as a double PID:

- a first PID computes the tilt required by the drone to reach the desired speed;
- a second PID adjusts the motor forces to achieve this tilt.

At each simulation step, the programme reads the current state of the drone, including its speed and orientation. The error between the current speed and the target speed is then calculated, providing the proportional, integral and derivative components of the correction.

These values are used to determine the forces to apply to each motor. The combination of forces allows the drone to tilt in the desired direction or change its altitude.

This method produces stable and realistic behaviour while remaining flexible enough for use in different scenarios.

3.6 User interface

The simulator's user interface is intentionally simple. It mainly consists of several sliders for adjusting the PID controller coefficients.

These parameters can be modified in real time to observe their influence on the drone's behaviour. The coefficient values are transmitted to the control scripts via Godot's signal system.

This approach facilitates experimentation and helps students understand the impact of regulation parameters on system stability.

4 Sport 2 Scroll - S2S

4.1 Project overview



<https://florian-boisset.itch.io/sport2scroll>

Sport2Scroll is a gamified fitness mobile application project whose goal is to use body tracking to help users perform their exercises correctly. A second objective of the project is to help users control their screen time by unlocking “scroll time” through physical exercise. The user defines their initial fitness level, their goals, the exercises they wish to prioritise, and the maximum scroll time they allow themselves. Once their screen time limit is exceeded, the application takes over social media apps and prompts the user to exercise in order to unlock more scroll time.

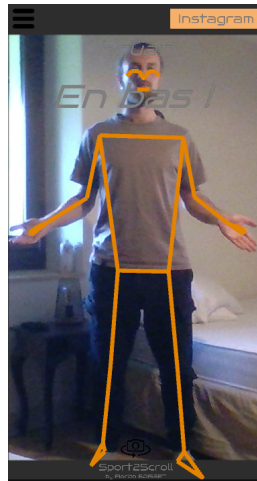
4.2 Technical design

The application is developed in **Unity 6** for Android.

Body detection with MediaPipe

Body detection relies on **MediaPipe Pose Landmarker**, an open-source library by Google specialised in real-time human posture analysis. For each frame captured by the front camera, MediaPipe analyses the video stream and returns the positions of **33 key points** distributed across the body: shoulders, elbows, wrists,

hips, knees, ankles, etc. Each point is provided with a confidence score indicating how certain MediaPipe is of its position.



Exercise definition

To define an exercise, a **custom editor tool** developed in Unity allows the user, in editor mode, to stand in front of the camera, perform the desired pose and **capture it in one click**. The tool records the angles of all joints at that moment. A description or instruction for the user is also added to guide them in adopting the pose.

An exercise is then defined by an **ordered sequence of reference poses**. For example, a push-up is described by two poses: arms extended on the floor, then arms bent (low position). During a session, the application asks the user to reproduce each pose in sequence; once all have been validated, one repetition is counted.

Currently available exercises:

- Push-ups
- Squats
- Knee-to-elbow
- Mountain Climbers
- Jumping Jacks
- Lunges

Pose Editor Tool

Pose Editor Tool

Capture de Pose

Pose Landmarker Runner
Solution (Pose Landmarker Runner)

Nom de la pose
New Pose

Description de la pose

Contrainte de posture

Corps horizontal requis
☐

Poids et tolérances par joint

!
Weight 1.0 = critique | 0.5 = secondaire | 0.0 = ignoré
Tolérance 0 = utiliser la tolérance globale du PoseComparer

Joint	Weight	Tolérance°
LeftShoulder	<input type="range"/>	1 0
LeftElbow	<input type="range"/>	1 0
RightShoulder	<input type="range"/>	1 0
RightElbow	<input type="range"/>	1 0
LeftTorso	<input type="range"/>	1 0
RightTorso	<input type="range"/>	1 0
LeftHip	<input type="range"/>	1 0
LeftKnee	<input type="range"/>	1 0
RightHip	<input type="range"/>	1 0
RightKnee	<input type="range"/>	1 0

Paramètres de capture

Délai avant capture (s)

Durée d'échantillonnage (s)

Capturer la pose actuelle

Pose validation

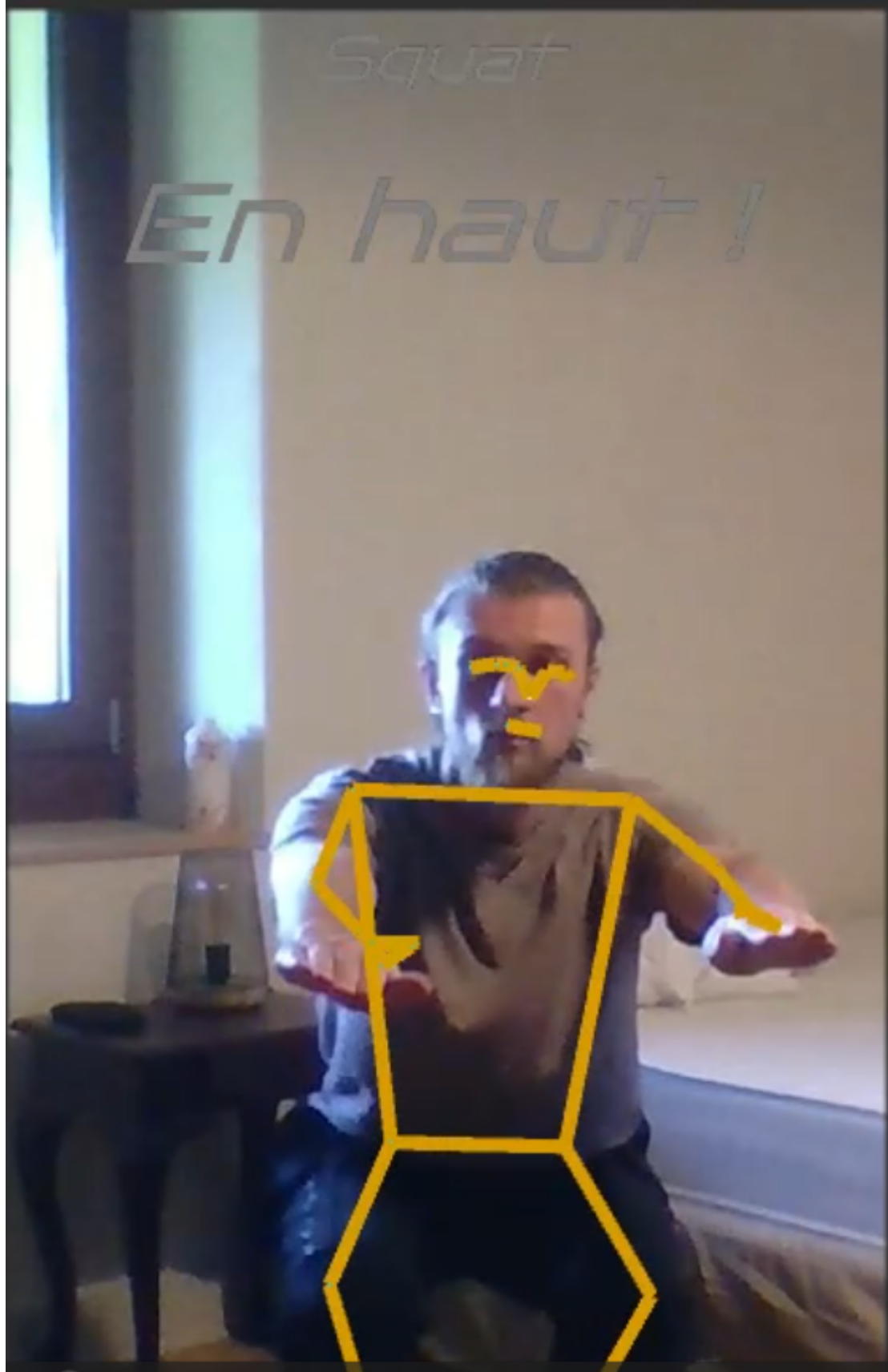
For each camera frame, the application computes the **joint angles** (e.g. elbow angle = angle between the upper arm and forearm) and compares them to those recorded in the reference pose. Each joint contributes to a **global score from 0 to 100**, with a configurable weight according to its importance in the exercise: for example, elbows are heavily weighted for a push-up but barely relevant for a squat. As soon as the score exceeds a defined threshold, the pose is validated and the application moves on to the next one in the sequence. Real-time visual feedback indicates the quality of the pose: the colour transitions from **red** (incorrect pose) to **green** (pose achieved).



Instagram

Squat

En haut !

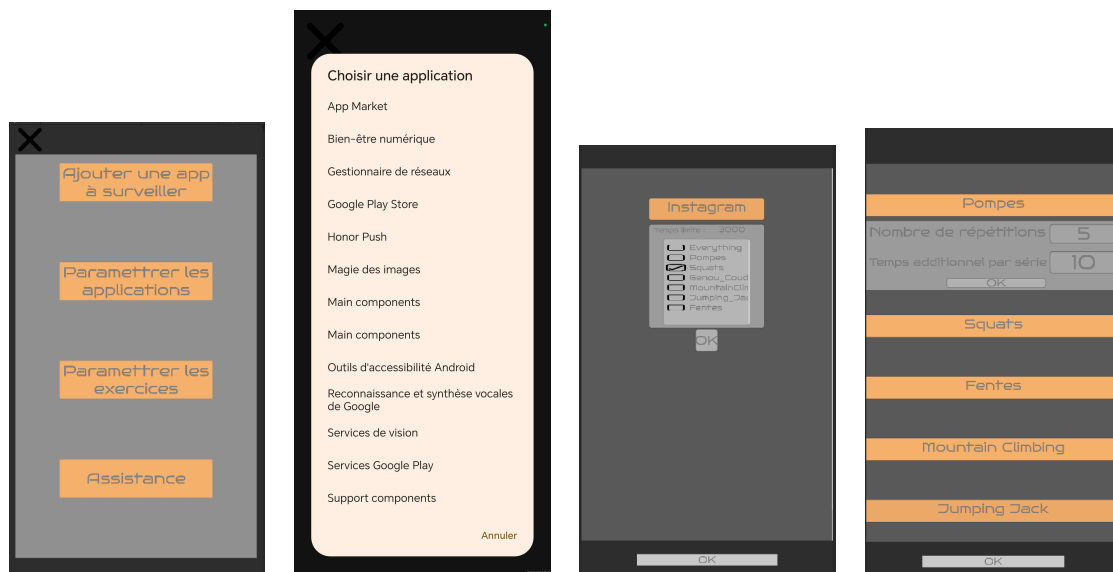


Exercise and monitored app settings

Exercises and monitored applications are presented as **lists in the interface**. Each item exposes its parameters directly via **editable text fields** in the list. For exercises, the user sets:

- the **number of repetitions** to perform per set;
- the **scroll time unlocked** at the end of the set.

For monitored applications, the user configures the **daily quota** allocated to each one. Counters automatically reset to zero every day at midnight.

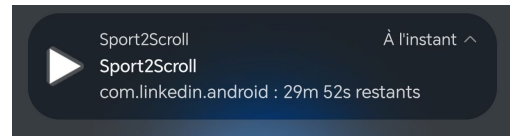
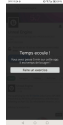


Screen time management

Tracking the time spent on applications relies on two complementary mechanisms:

- The Android **UsageStats API** provides the cumulative usage time of each application since the start of the day;
- An **accessibility service** listens in real time to system events: opening a monitored app, closing it, and **screen lock**. This last point is crucial: if the user turns off their screen without closing the app, the counter is paused, preventing screen-off time from being counted as active usage.

As soon as a monitored application is detected as open, a **persistent notification** appears and updates in real time with the remaining usage time for that application. As soon as the quota is exhausted, an **overlay** is displayed over the application, prompting the user to exercise in order to unlock more time.



Five Android permissions are required for the application to function:

- **Camera** (*CAMERA*): access to the front camera for pose detection;
- **Usage statistics** (*PACKAGE_USAGE_STATS*): reading of cumulative time per application. Sensitive permission, granted manually in system settings;
- **Accessibility service** (*BIND_ACCESSIBILITY_SERVICE*): real-time detection of app openings, closings and screen lock events. Granted manually in accessibility settings;
- **Display over other apps** (*SYSTEM_ALERT_WINDOW*): display of the blocking overlay and foreground notifications. Granted manually via settings;
- **Ignore battery optimisation** (*REQUEST_IGNORE_BATTERY_OPTIMIZATIONS*): prevents Android from suspending the background monitoring service to save battery, ensuring continuous tracking even when the application is idle.

5 Combo Survivor

5.1 Game overview

Combo Survivor is a game made during a game jam in collaboration with Esteban MELERO. The theme of this jam was 'COMBOS'.

WEBGL_EMBED_PLACEHOLDER

The game's Itch.io page!



Figure 5.1: Screenshot of the game.

5.2 Game content

5.2.1 The concept

To answer the jam's theme, several types of gameplay were considered, such as a fighting game, a merging game (in the style of Suika Game, Candy Crush, etc.) and many others. We settled on a Suika Game-style gameplay with element merging. Our thinking then led us towards a Survivor-like (for example, Vampire Survivors). In this game, the player must therefore survive waves of enemies that appear, and must eliminate them before they merge with one another.

5.2.2 The monsters

The 100 monsters and their animations come from a Unity Asset Store package.

- **Merges:** When 2 strictly identical monsters come into contact, they merge. Their stats (health and damage) are improved. They become harder to defeat and deal more damage to the player.

- **Combos:** Monsters are split into 9 types based on their appearance: fleas ('Flea'), flying balls ('Flying Ball'), round heads ('Round Head'), mounted creatures ('mounted'), toads ('Toad'), ostriches ('Ostrich'), four-legged creatures ('Four Legged'), armed creatures ('Weaponned') and a miscellaneous type. When monsters attack you, they can perform combos. For example, if 3 enemies of the same type hit you, the damage of the 3rd attack is multiplied. Likewise, if one monster of each type hits you in a row, a multiplier is also applied to the last attack. A 'Combos' button in the menu shows the different combinations and their effect.
- **Monster spawning:** Monsters initially spawn every 2 seconds; the spawn interval shrinks over time, reaching a maximum spawn rate of 20 monsters per second. As the run progresses, the list of enemies that can spawn shrinks, favouring both the appearance of stronger monsters and monster merging, since they become increasingly identical to one another.

5.2.3 The player

This asset comes from the Unity Asset Store; the scripts were extensively modified and adapted to match the behaviour wanted for our game.

- **The sword:** the player can attack enemies with sword strikes using the right mouse button. This attack is slow and has limited range. However, every enemy killed with this attack grants the player one experience point and one power point.
- **Spells:** the player can cast spells using the left mouse button. Spells travel in the direction of the mouse cursor on screen. They are far more powerful and have a longer range than sword attacks. However, they cost 1 power point, and grant neither power nor experience points when they kill a monster.
- **Levelling up:** every 10 experience points, the player gains a level. Levelling up increases the damage dealt by sword strikes and restores some health to the player. It also increases spell power and grants a number of bonus power points.

5.2.4 The menu

Press 'Escape' to bring up the menu. It pauses the game and allows the player to view the list of monster combos, the game credits, and restart the run using the play button.

6 ThousandsPong

6.1 Game overview

ThousandsPong is a multiplayer Pong game built with **Unity** and hosted on a personal server. Unlike a classic two-player Pong, the arena dynamically reshapes itself into a **regular polygon (n-gon)** whose number of sides matches the number of currently connected players : each player defends one side of the arena.

florian-boisset.duckdns.org

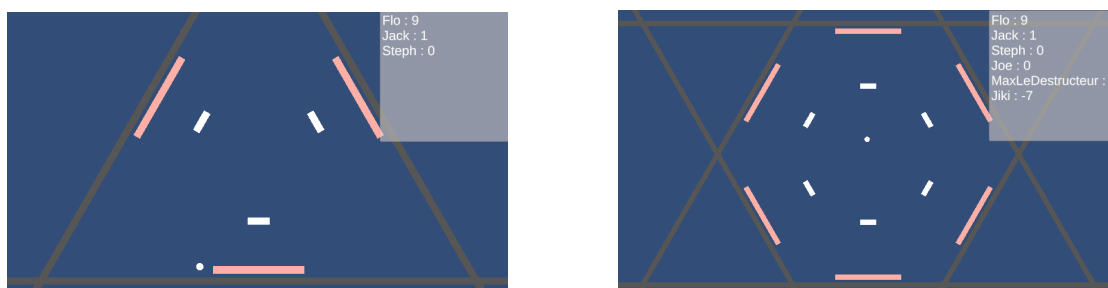


Figure 6.1: The arena reshaping into a triangle (3 players), then a hexagon (6 players).

6.2 Game rules

Each player defends a goal with a paddle positioned on their side of the polygon. When a goal is hit by the ball, the player defending it loses a point, while the last player to have touched the ball with their paddle before the impact scores a point.

When the game launches, the player is asked for a nickname. This nickname is sent to the server, which looks it up in its local **PostgreSQL** database : if an entry already exists for that name, the associated score is retrieved and the player resumes their progress ; otherwise, a new entry is created. Each player's score therefore persists from one session to the next, including when it is negative.

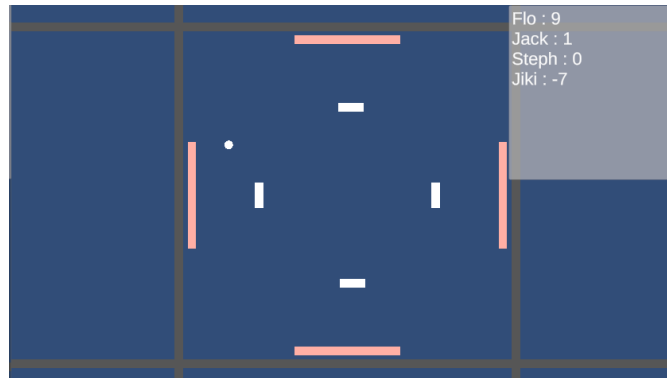


Figure 6.2: In-game persistent scoreboard, showing both positive and negative scores.

6.3 Server authority

The entire game logic (ball movement, goal detection, score calculation, arena reshaping) is handled with **server authority**. This networking architecture is built on Unity’s **Netcode for GameObjects** package (with **Unity Transport**), which handles state replication and synchronisation between the server and the clients. The server runs a **dedicated Linux build** of the game, executed on the personal server ; the clients (Windows and Linux builds) only forward player input and display the state received from the server. This design prevents any client-side cheating and centralises the handling of connections and disconnections, in particular to recompute the arena’s shape whenever the number of players present changes.

6.4 Server infrastructure and security

6.4.1 Server hardening

The personal server hosting the game is hardened in several ways : a **firewall** restricts exposed ports, processes run with **minimal privileges**, and web exposure goes through **Caddy** as a reverse proxy (with automatic HTTPS management), with the domain name provided by **DuckDNS**.

6.4.2 Game server control website

A small website (an `index.html` file, a `.js` file, and a lightweight **Python** backend) allows the Unity dedicated server process to be started and stopped remotely.

The backend triggers these actions through a dedicated **sudoers** rule, precisely restricting the commands the website is allowed to run with elevated privileges (only starting and stopping the relevant service), with the game process itself managed as a **systemd** service (**.service**).

To limit abuse from bots repeatedly starting/stopping the server process, the website includes a **Cloudflare Turnstile** challenge before triggering the action.

6.4.3 Automatic shutdown

A **systemd timer** (**.timer**) periodically triggers a script that inspects the service's **journalctl** logs to count player connections and disconnections and determine whether any players are currently present. If the arena has been empty for a certain amount of time, the script automatically stops the game server process so as not to waste resources.

6.5 Download

The website allows the **Windows build** and the **Linux build** of the game to be downloaded directly, available at florian-boisset.duckdns.org.