ROOFTOP RIVALS

RAPPORT DE SECONDE SOUTENANCE – SAE J3D

SALVAN ACHILLE - BENNJAKHOUKH RAYAN – CLEMENT GREGORY-LUCAS – HADJAB MEHDI – PUIJALON MILAN

MARCH 2026 / EPITA PARIS

SUMMARY

Table of Contents

SUMMARY 2
INTRODUCTION 3
ORGANIZATION & METHODOLOGY 4
INDIVIDUALS CONTRIBUTIONS 5
Rayan BENNJAKHOUKH – Group Leader 5
Objectives 5
Map Design 5
Menu Development 8
Conclusion 12
Achille SALVAN – Technical Lead 13
Objectives 13
Architecture and Physics Refactoring 13
New Movements Mechanics and Fixes 15
Conclusion 18
Mehdi HADJAB – Game Manager 19
Introduction 19
The SUPER Powers 19
Milan PUIJALON – Creative Director 23
Objectives 23
Planning 23
Progress 24
Gregory-Lucas CLEMENT – Team Coordinator 29
Objectives 29
HUD Design and Information Choices 29
Technical Implementation in Panda3D 29
Visual Identity and Readability 30
Challenges and Work in Progress 30
CONCLUSION 31
BIBLIOGRAPHY 31

INTRODUCTION

In our first year at EPITA, we have to develop a videogame with Python. At the beginning of the year, after a few group meetings, we decided to create a 3D videogame with a future theme. In our videogame, there is a chase game with two persons. One has to catch the other one, but at the same time, this other has to escape from him, when a player is caught, the turn changes and he has to be chased. The player who stays hunted the most time at the end of a round wins. In a game, three rounds are played. Each round lasts five minutes. To develop this videogame we took the decision to use Panda 3D because it was the tool that suited the most with our goals an d objectives.

During the first part of the year, from September to December we focused a lot on the movements, the menus, the music and the website but as reported it in our last report we were behind on AI and the graphical side. Our goal from December to now have thus been to continue to develop all the parts of our game as planned but also to work on these two major topics. We kept going on the same way with a same vision for our game, a tag game in a futuristic environment, cyberpunk-styled.

The first report was focused on our brainstorming about the game and it was the period where we learned about the tool that we have been using. For example during the first semester we were late on graphical aspects because Blender is such a difficult tool. On one hand, during the first semester was very focused on the planning, design thinking, teamwork to help us build a great foundation for this project. At the beginning of the second semester, it was more important for us to be more efficient in developing the videogame to have a working game as quickly as possible that we'll upgrade during the fourth bimester. Overall, on this third Bimester everyone had their own tasks to focus on.

After the first semester we took the decision to change the distribution of tasks slightly. Achille is now in charge of the website, Gregory is in charge of the HUD with Mehdi while Rayan in charge of the map. These changes have been made so Mehdi can focus more on AI and this permits to split the graphical work in two with Milan handling the characters and Rayan handling the map.

ORGANIZATION & METHODOLOGY

During this third bimester, organization has been even more important than during the first semester. In fact, the third bimester has been more challenging overall, it was more difficult to dedicate time to this project because our lectures were difficult and demanding. Therefore, we had to organize ourselves carefully to keep progressing on the project even with less time. It was thus even more critical to be well organized during this bimester, and we are already aware that in the following bimester we will have less time while having more work to do, so it is mandatory to plan and organize ourselves as efficiently as possible.

To stay well organized, we kept our weekly meetings through Discord. These meetings allow everyone to share their progression and updates, so that if someone is falling a bit behind on their objectives, they can ask the rest of the team for help and support. This habit has proven to be essential in maintaining a steady workflow throughout the bimester.

The fact that our videogame is very ambitious makes teamwork even more important. Building a 3D videogame in Python is a real technical challenge, and without strong collaboration and clear communication with the team, it would be very difficult to progress efficiently. Everyone's contribution matters and the coordination between each member is key to the success of this project.

At each meeting, we plan some deadlines that allow us to merge our parts step by step. Even if we sometimes faced some difficulties, we succeeded by improvising so the work could be done even with some problems. For example, we had a lot of problems with the deadlines so one of the group members took the responsibility to pause their work to help another member. This allowed them to learn about something new and to help the others.

INDIVIDUALS CONTRIBUTIONS

Rayan BENNJAKHOUKH – *Group Leader*

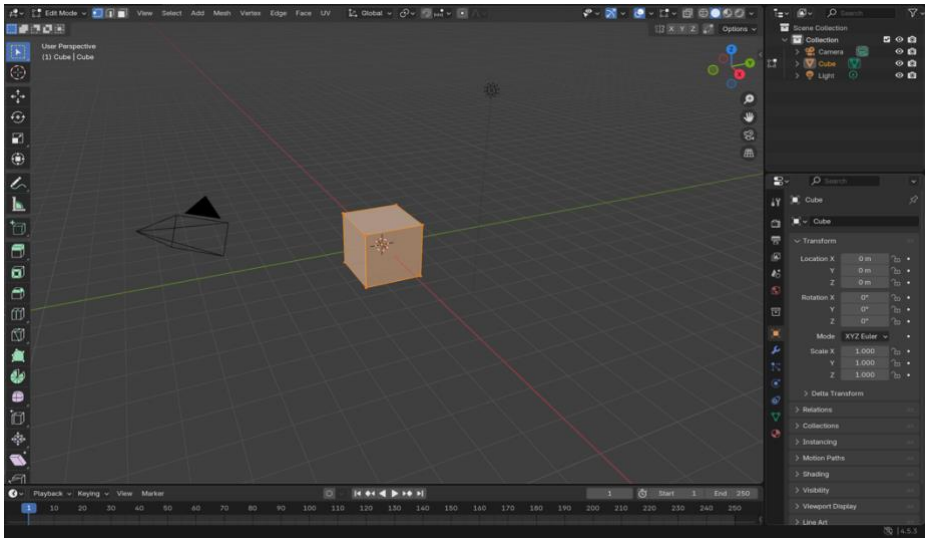
Objectives

During this bimester I am in charge of the menus and the map of the video game. I am also in charge of the organization and management of the group as the group leader. For the menus I have to continue on the same path as in the first semester. For the map I have to learn Blender and then use it to create a 3D map for the videogame with a futuristic, cyberpunk mood, a map designed for chase tag.

Map Design

Learning of Blender

During these first weeks I had to learn Blender from scratch which was honestly pretty challenging at first. Blender is a very powerful tool but it is not the easiest one to get into. I spent a few weeks just getting comfortable with the basics, understanding how the software works, how to navigate in the 3D viewport, and getting familiar with the different modes. The two main ones I had to understand were the Object Mode and the Edit Mode. The Object Mode is basically where you manage your objects as a whole, you can move them, scale them, rotate them, place them in your scene. The Edit Mode is where things get more precise, it is where you actually go inside your object and start modifying the geometry, working with vertices, edges and faces to shape your mesh the way you want. It took me some time to really understand the difference between the two and when to use which one, but once it clicked it started to make a lot more sense. The screenshot below permits to discover all the different tools in Blender.

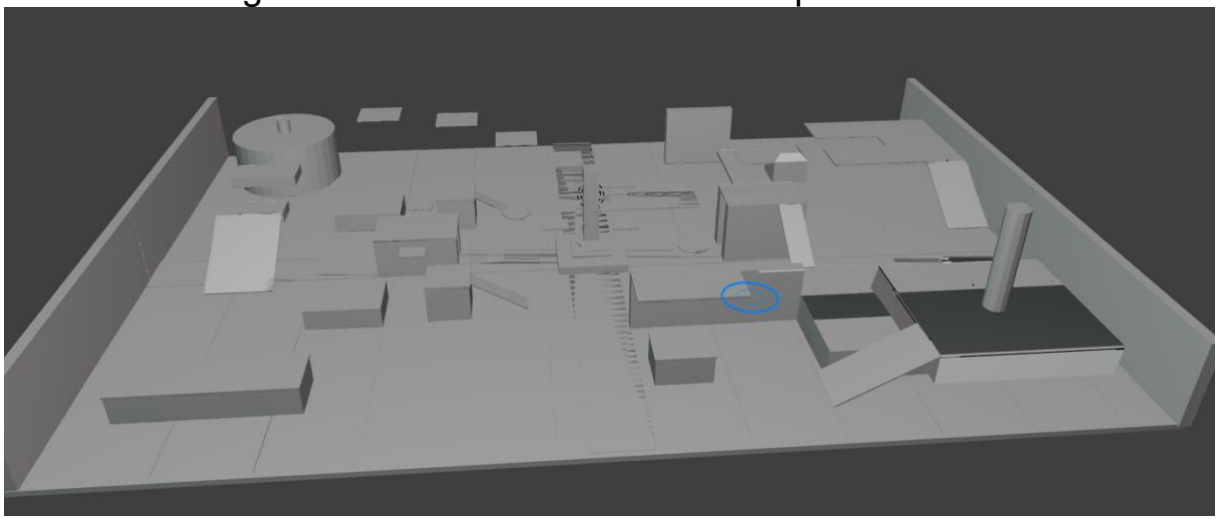


Geometrical Design of the map

Once I got comfortable enough with Blender, I started working on the actual map. The idea was to create something that really fits the cyberpunk mood we had in mind while also being fun and well balanced to play on. I built the map using different geometric shapes, cylinders, ramps, and cubes acting as platforms, nothing too complicated in terms of shapes but the way I arranged them is what really matters. I thought about the whole thing like a parkour map, I wanted the player to always have something to jump on, to slide under, to climb, to run through, so that the movement never feels boring or repetitive.

I also designed the map with the mechanics coded by Achille in mind, especially the double jumps and the slides. It was important that the map actually takes advantage of these moves, so I made sure to create situations where a double jump is useful to reach a higher platform or where a slide lets you go faster under an obstacle. A map where these mechanics are never used would have been a waste so I really tried to make them feel necessary and satisfying to use.

On top of that I also thought about the powers that Mehdi coded and tried to make the map work well with them, making sure there are enough open spaces or interesting spots where these powers can make a real difference during a chase. The goal was to have a map that feels alive and where every mechanic has its place.
Here is all the geometric forms that form our map :



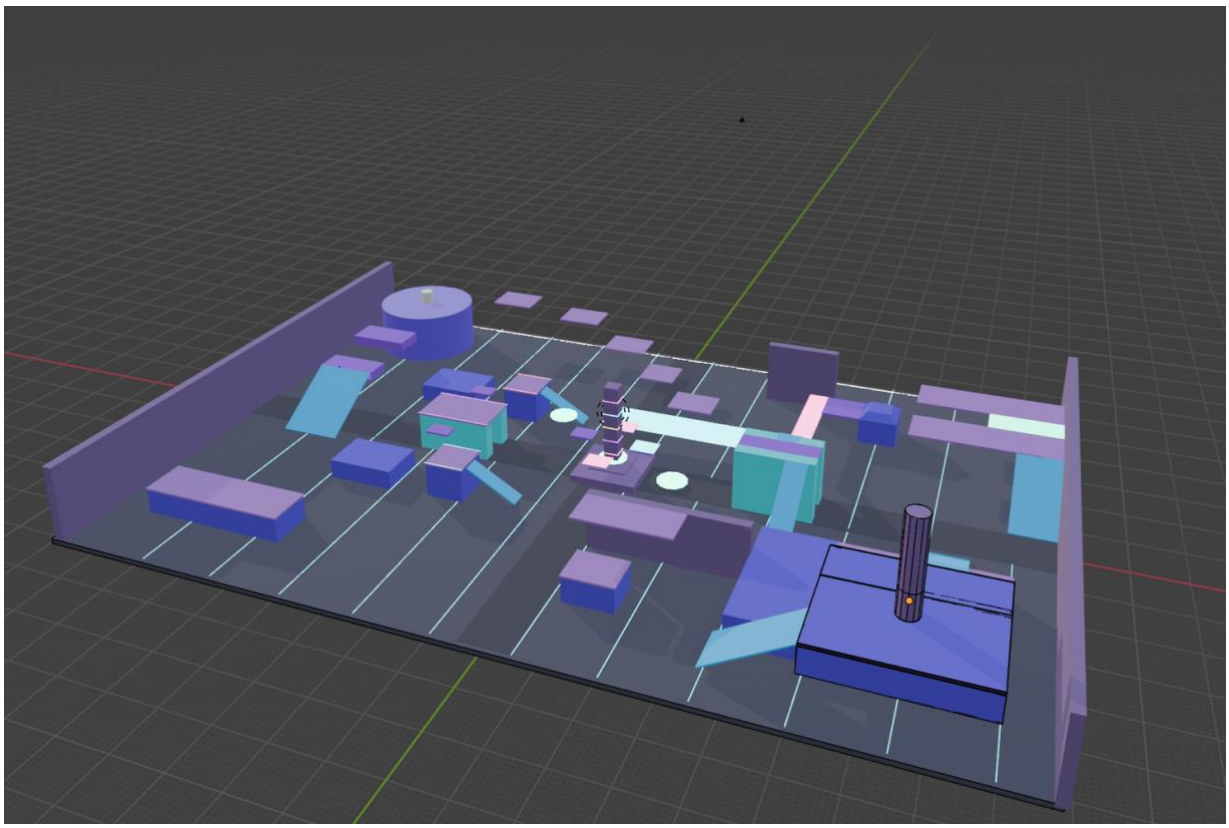
Colors choosen for the map

For the colors I went with purple, cyan, dark blue and some green, basically neon looking colors that fit the futuristic and cyberpunk theme we had from the start. I was careful about how I combined them together because it is not just about picking cool colors, they have to actually work well with each other.

The general atmosphere of the map is quite dark, I kept the structures and the ground pretty dark on purpose so that the bright colors really stand out. This contrast is what gives that cyberpunk feel, if everything was bright it would just look messy and the neon effect would be completely lost.

I also tried not to put the vivid colors everywhere, too much of it and it becomes overwhelming. So I placed them carefully to keep something that looks clean and coherent while still being visually interesting. I think the result fits well with the universe we are going for.

Here is the final result of the Map :



Menu Development

Upgrade of the Principal Menu

Part 1 — The original menu

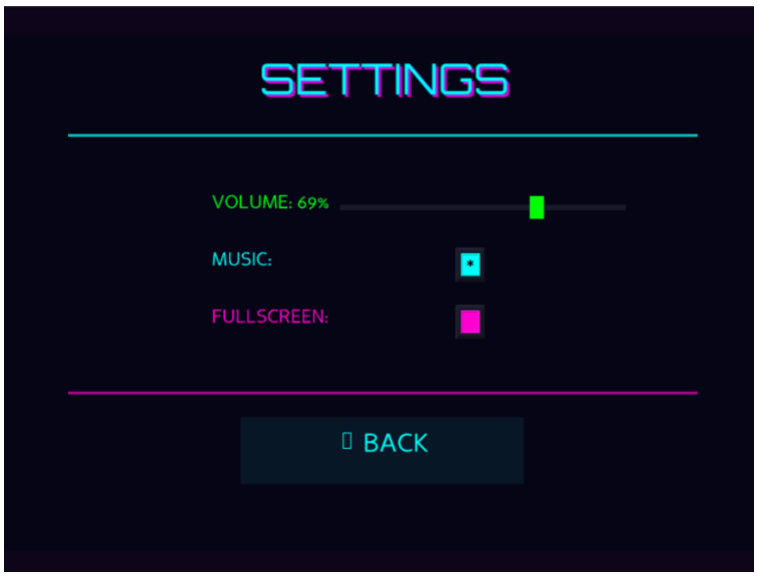
Back in December, I put together the first version of the main menu for Rooftop Rivals. At that point I was mostly focused on getting the visuals right — the neon cyan and pink colors, the title, the subtitle, the overall dark urban aesthetic. The three buttons were already there, Start, Settings and Exit, and the layout was clean. But honestly, only the Exit button actually did something. Clicking Play or Settings just did nothing, the buttons were basically decorative. It was a good starting point though, the structure was there and I knew what I wanted to build on top of it.

Part 2 — The Settings menu

This is probably where I spent the most time. I wanted the Settings menu to actually work, not just look good. So I built three real options. The first one is a volume slider — when you drag it left or right, the percentage updates live on screen and the music volume changes at the same time, instantly. No need to confirm or press anything, it just reacts. The second option is a simple checkbox to turn the music on or off entirely, which either plays or stops the track. The third is a fullscreen toggle to switch between windowed and fullscreen mode. The music itself was made by my teammate Milan, I just handled plugging it into the engine. The way the slider connects to the audio is actually pretty straightforward in Panda3D, every time the slider moves it calls this function :

```
def changerVolume(self):
    self.volume = self.sliderVolume['value']
    self.labelVolume.setText(f"VOLUME: {int(self.volume * 100)}%")
    self.musique.setVolume(self.volume)
```

So self.musique.setVolume() is what actually talks to the audio engine and adjusts the volume in real time. One thing I also made sure of is that when you leave the Settings menu and go back to the main menu, every single element gets properly destroyed rather than just hidden, to avoid any memory issues or things showing up where they shouldn't.

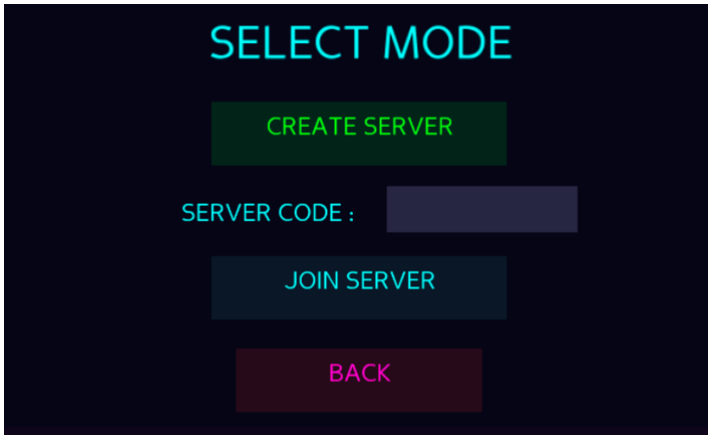


Part 3 — The Play button

The Play button was the other big thing I worked on. Before it did nothing, now clicking it opens a whole new screen called Select Mode. On this screen the player has two choices : create a server or join an existing one. The interesting part is the join flow — there's a real text field in the middle of the screen where the player can click and type a 6-character server code directly from their keyboard. This is using Panda3D's DirectEntry widget which works like a proper input field, nothing fancy to set up but it makes the experience feel a lot more complete. Once the code is typed, the player can either press Enter or click the Join Server button, both of which call the same function to grab what was typed :

```
def recupererCode(self):
    code = self.champCode.get()
    print(f"Code saisi : {code}")
    return code
```

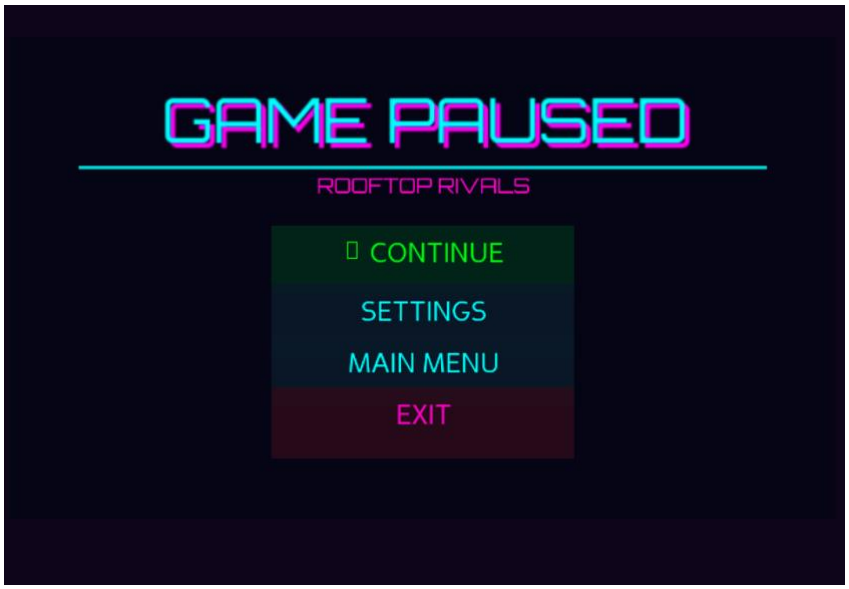
From there the code can be passed to the networking side of the project to actually connect to the right server. That part is handled by my teammates, but the menu is fully ready to send the value over.



The upgrade of the in-game Menu

Part 1 — The ingame menu

On top of the main menu, I also built a separate ingame menu, the one that pops up when you're already in a game and you want to pause. Visually it keeps exactly the same aesthetic as the main menu, same dark background, same neon colors, same font so it feels consistent across the whole game. It has four buttons : Continue, Settings, Main Menu and Exit.



Part 2 — The Settings menu

The Settings menu is exactly the same as the one from the main menu. Same volume slider, same music toggle, same fullscreen option. I didn't rebuild it from scratch, I just reused the exact same logic and structure since there was no reason for it to be different. The player gets the same

experience whether they access Settings from the main menu or from inside a game.

Part 3 — The Continue and Exit buttons

Continue and Exit are straightforward. Exit closes the game entirely using Panda3D's built-in self.userExit(). Continue is intentionally left doing nothing for now, it's there visually but it's waiting for the actual game logic to be connected to it later, once the gameplay side is more advanced. The button exists, it just has command=None for the moment.

Part 4 — The Main Menu button

The most interesting button is Main Menu. When the player clicks it, it doesn't just go back to a previous screen, it actually closes the ingame menu entirely and relaunches the main menu as a completely separate process. The way I handled it is by using Python's subprocess module to start menu_sout1.py as a new independent process, and then immediately calling self.userExit() to shut down the current window :

```
def allerMenuPrincipal(self):
    import subprocess
    import sys
    subprocess.Popen([sys.executable, 'menu_sout1.py'])
    self.userExit()
```

This way the two menus are fully independent from each other, and going back to the main menu feels clean with no leftover state from the ingame session.

Conclusion

Overall, this bimester I feel like I genuinely hit the goals I set for myself. On the menu side, I did exactly what I planned, I took something that was mostly visual and turned it into a fully functional system with real navigation, working settings and a proper server join flow. But beyond the menus, I also made a lot of progress on the map itself, which taught me a ton of things I didn't know going in. I got way more comfortable with Blender, learning how to model and export assets properly, and I deepened my understanding of Panda3D and Python, figuring out how the engine works, how to load models, handle audio, manage windows and build interfaces from scratch.

But honestly, the thing I'm most proud of this semester isn't just the technical stuff. Managing a team on a project like this, where everyone has different tasks, different speeds and different levels of experience, is genuinely hard. There were moments with a lot of pressure and a lot of things happening at the same time, and I had to learn how to keep things moving, communicate clearly with my teammates and make sure everyone knew what they had to do. That's something you can't really learn from a tutorial, you just have to go through it. And I think I came out of this semester a lot better at it than I was at the start.

Going forward, I want to keep that momentum going. One thing I really want to focus on next semester is being more available for my teammates, making sure no one gets stuck on something for too long, helping out when someone is struggling and keeping the whole team aligned so we can deliver something we're all actually proud of. The technical skills matter, but at the end of the day a good final product comes from a team that works well together, and that's something I want to keep investing in.

Achille SALVAN – *Technical Lead*

Objectives

My main objective for this presentation was to upgrade the movements and add the main mechanics of the game. Movements will define how the game will be played, we need them to make our final choices on the game design, but especially to build the map. The final map must completely exploit movement mechanics to ensure a fun experience and to have a smooth gameplay. This is the reason why movements must be thought and created before the final map.

For this presentation, a part of my job was also to combine all of the progress that we made. This presentation, the objective was to be able to show our game working, it still looks like a prototype because we didn't focus on the appearance of the game, but the main logic has been implemented and will allow us to submit a fully working game in the deadline.

Architecture and Physics Refactoring

Physics System Migration

Initially, I used a "BulletCharacterControllerNode" which provides built-in physics systems like jumping, slope management, and crouching. However, these existing systems became restrictive. When trying to implement custom movement mechanics, I found myself constantly fighting against the engine to make them work. Furthermore, many basic functionalities were missing; for instance, retrieving the player's true velocity required manual computation.

I decided to rebuild the movement system from scratch using the "BulletRigidBodyNode". This is a much lower-level approach where the only pre-existing physics applied are basic collisions, giving us total mathematical control over the player's behavior.

Class Architecture Redesign

To keep our codebase clean and scalable, I implemented an object-oriented architecture around the player entity. The Player class handles the core logic shared across all player types such as hitbox, basic physics parameters, visual nodes, and the power-up system.

From this base, I created two classes. The Controller class, a child of Player, and implements local input polling, camera management, and complex movement logic. And secondly, the NetworkPlayer class which is also a child of Player but is kinematic, meaning that no physics logic is applied to it, it only relies on network updates and interpolation to synchronize positions without computing local physics.

```python
# Base class handling core physics and visuals
class Player:
    def __init__(self, game, start_pos=None) -> None:
        self.controller = BulletRigidBodyNode("player")
        self.controller.setMass(80.0)
        # ... Hitbox and orientation setup

# Local player handling inputs and mechanics
class Controller(Player):
    def __init__(self, game, start_pos=None) -> None:
        super().__init__(game, start_pos)
        self.slide = PlayerSlide(self)
        self.dash = PlayerDash(self)
        # ... Input bindings

# Network counterpart inheriting the same base structure
class NetworkPlayer(Player):
    def __init__(self, game, player_id, start_pos=None) -> None:
        super().__init__(game, start_pos)
        self.controller.setKinematic(True)
        self.controller.setMass(0.0)
        # ... Interpolation logic
```

New Movements Mechanics and Fixes

Smooth Air control

To make platforming enjoyable, players need to adjust their trajectory mid-air without losing momentum. By calculating acceleration differently whether the player is grounded or airborne, we introduced a smooth air control system. We use mathematical decay based on delta time to compute the acceleration factor, applying different movements logic and constraints to air movements compared to ground movements.

```
# Code from Controller._move(self, dt)
if self.is_on_ground:
    accel_factor = 1.0 - math.exp(-GROUND_ACCELERATION * dt)
    diff_x = (target_vel_x - current_vel.x) * accel_factor
    # Apply ground impulse...
else:
    accel_factor = 1.0 - math.exp(-AIR_ACCELERATION * dt)
    # Applying AIR_CONTROL dampening to allow slight trajectory adjustment
    diff_x = (target_vel_x - current_vel.x) * AIR_CONTROL * accel_factor
    self.controller.applyCentralImpulse(Vec3(diff_x * mass, diff_y * mass, 0))
```

Sliding

The sliding mechanic allows players to gain a temporary speed boost that decays over a set distance, lowering their camera to simulate crouching. We calculate the progress of the slide using the distance covered compared to the maximum slide distance (SLIDE_DIST). As the slide progresses, the speed is linearly interpolated down to a final speed threshold.

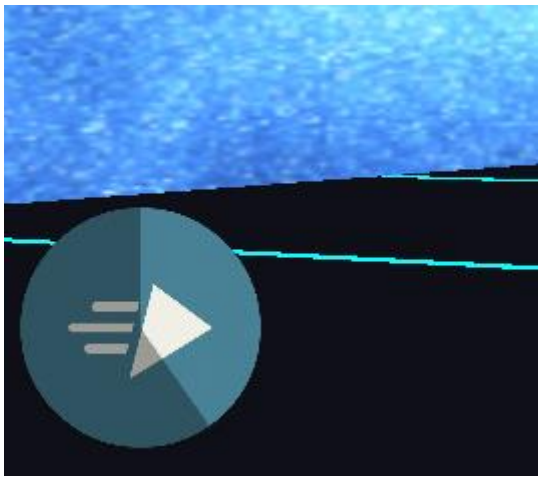
```
# Code from PlayerSlide.update(self, dt)
current_pos = self.controller.node_path.getPos(self.controller.game.app.render)
dist_covered = (current_pos - self.start_pos).length()
progress = min(dist_covered / SLIDE_DIST, 1.0)

# Linear interpolation for speed decay
final_speed_abs = self.controller.speed * SLIDE_FINAL_SPEED
self.current_speed_val = self.start_speed - (self.start_speed - final_speed_abs) * progress

if progress >= 1.0 or self.current_speed_val <= final_speed_abs:
    self.stop() # Stop sliding
```

Dashing and UI

Dashing provides an instant boost of velocity in the direction of the camera, and it also temporarily disabling gravity to ensure a straight trajectory. The dash was overpowered, so I added a cooldown which forces players to think about when they should use the dash. To give players visual feedback on the mechanic's cooldown, I created a custom HUD element.



I wanted a circular animation and not just a timer, Panda3D doesn't handle videos on HUD, but it supports images. A video is just an image sequence, I just had to have all frames of the cooldown, and to put them on screen one after the other. Instead of manually drawing 150 frames for the cooldown animation, I wrote an independent Python script using the PIL (Pillow) library to generate a pie-slice animation automatically. This approach saved a massive amount of time and ensures perfectly accurate frame timing.

```python
# Direction de la caméra via les quaternions
quat = self.controller.camera_pivot.getQuat(self.controller.game.app.render)
self.dash_dir = quat.xform(Vec3(0, 1, 0))

# Sin max de Z à partir de l'angle max du dash
max_sin = math.sin(math.radians(MAX_DASH_ANGLE))

if self.dash_dir.z > max_sin:
    self.dash_dir.z = max_sin

    # Recalcul de X et Y pour conserver un vecteur normalisé
    xy_length = math.cos(math.radians(MAX_DASH_ANGLE))

    horizontal_dir = Vec3(self.dash_dir.x, self.dash_dir.y, 0)
    if horizontal_dir.length_squared() > 0:
        horizontal_dir.normalize()

    self.dash_dir.x = horizontal_dir.x * xy_length
    self.dash_dir.y = horizontal_dir.y * xy_length
else:
    if self.dash_dir.length_squared() > 0:
        self.dash_dir.normalize()

# Désactivation de la gravité pendant le dash
self.controller.controller.setGravity(Vec3(0, 0, 0))
```

Sticky Ground

One major issue with using a raw RigidBody for a player controller is that every small relief is making the player bounce. To fix this, I implemented a vector projection technique. By calculating the cross product of the movement vector and the surface normal, we ensure the player's velocity is strictly parallel to the ground they are walking on.

```python
# Vector projection to stick to ground in Controller._move
if self.is_on_ground and move_vec.length_squared() > 0:
    # Cross product to find the right vector relative to the slope
    right_vec = move_vec.cross(self.ground_normal)

    # Cross product to adjust the forward vector parallel to the slope
    move_vec = self.ground_normal.cross(right_vec)
    move_vec.normalize()

    current_vel = self.controller.getLinearVelocity()
    # Cancel upward bouncing if we didn't jump
    if current_vel.z > 0 and self.jump_timer <= 0:
        current_vel.z = 0.0
        self.controller.setLinearVelocity(current_vel)
```

Conclusion

At this stage, I've successfully implemented the majority of the core movement mechanics. The modular object-oriented architecture I built around the Player class makes me able to add a new mechanic without breaking the existing ones and makes this process straightforward. The next feature to integrate, and probably the last one for movement, is the wallrun. Once that is done, I will consider the movement system complete. What is going to remain will mostly be just tweaking variables, speeds, distances, impulses to get a smooth gameplay and make sure it matches our vision.

Looking ahead, the next big technical challenge is going to be the implementation and network synchronization of Artificial Intelligence. To not overload the bandwidth and ensure smooth gameplay across all clients even with a bad connection, I'm planning to explore a deterministic approach based on a shared random seed. The idea is that by synchronizing the seed, each client can compute the AI's "random" pathfinding locally on their end. That way, the server only needs to send the initial spawn locations and seeds, which cuts down a massive amount of data going through the network.

When AI synchronization is finished, we'll hit the technical feature freeze. From there, the focus shifts entirely to the final phase, polishing. That means improving the visual assets, refining the overall aesthetics, and pushing the game feel as far as we can, basically making sure the result feels fun and finished.

Mehdi HADJAB – *Game Manager*

Introduction

As part of our video game project, we aimed to design a gaming experience that is dynamic, accessible, and strategic. The main objective was to offer gameplay that is easy to understand while remaining rich enough to keep players engaged over time. To achieve this, we worked on game mechanics, code structure, and the addition of original features.

Rooftop Rivals is inspired by a tag-like game concept: one player takes on the role of the mouse, while the other plays the hunter. The core of the game is based on time management, movement, and interactions between entities. To enrich this foundation, we decided to implement a system of temporary powers called **SUPER Powers**, which add an extra layer of strategy.

This report presents the technical and conceptual choices made, as well as the different stages of the project's implementation.

The SUPER Powers

The SUPER Powers were designed to make matches more dynamic and to avoid repetitive gameplay. The idea was to randomly spawn special entities on the game map. When a player comes into contact with one of the called "drones", they receive a temporary power that alters their abilities.

These powers serve several purposes:

- Introducing variety into matches
- Rewarding risk-taking and map exploration
- Creating unpredictable game-changing situations

Among the powers envisioned, the first one implemented was **SUPER Speed**, which allows a player to increase their movement speed for a short period of time.

The power-up system was designed around three main elements: the collectable object placed in the world, the manager that controls all drones, and the power-up logic itself. This structure makes the system modular, easier to maintain, and easier to extend with new effects.

The first part is the collectable object, implemented in game/PowerUps/collectable.py. A collectable represents one drone in the

3D world. It contains a visible model and a BulletGhostNode, which is used as a trigger zone. The ghost node does not physically block the player, but it detects overlaps. This is what makes the object collectible. When the player touches the collectable, the method on_collect(player) is called. This removes the collision object from the physics world and gives the effect to the player

```
shape = BulletSphereShape(radius)
self.ghost = BulletGhostNode("collectable")
self.ghost.addShape(shape)
self.ghost.setIntoCollideMask(BitMask32.bit(1))

self.node = game.app.render.attachNewNode(self.ghost)
self.node.setPos(pos)

game.world.attachGhost(self.ghost)
```

The second part is the drone manager, implemented in game/PowerUps/collectables.py. This class stores all active drones inside a list and updates them every frame. Its role is to spawn drones, move them, and check if the player has collected them. Initially, drones appeared near the camera, but the system was later improved so that they could move randomly across the whole map. To do that, the manager reads the map boundaries and uses raycasts to place drones on valid ground positions.

```
def update(self, dt):
    self.item_system.update(dt)
    elapsed_time = self.app.taskMgr.globalClock.getFrameTime()
    for collectable in self.collectables[:]:
        collectable.update_motion(dt, elapsed_time)
        if not collectable.can_collect():
            continue
        if self._is_touching_player(collectable):
            collectable.on_collect(self.player)
            self.collectables.remove(collectable)
```

The random movement itself is handled in collectable.py. Each drone has a target position and moves toward it. When it reaches that position, it selects a new random target. A floating effect was also added with a sine function so that the drone looks alive instead of static.

The third part is the power-up logic, implemented in game/PowerUps/powerup.py. Each collectable contains an Item, and each item contains one power-up. When the player collects a drone, the power-up is applied and remains active for a limited duration. The player keeps active effects in a list, and each effect is updated every frame until it expires.

The simplest power-ups are stat modifiers such as speed boost and jump boost. These effects temporarily change one player attribute, then restore the original value when the timer ends.

```
class PowerUp:
```

```
def apply(self, player):
    self.original_value = getattr(player, self.attr_name)
```

Several additional power-ups were added after the original jump boost. The first one is Invisibility. It reduces the alpha of the player model, making the player harder to see for a few seconds.

```
class InvisibilityPowerUp:
```

```
def apply(self, player):
    self.end_time = time.time() + self.duration
    self._target_visual = getattr(player, "visual", None)
    if self._target_visual is not None and not self._target_visu
        self._target_visual.setAlphaScale(self.alpha)
```

The second one is Radar. This power-up allows the player to see the opponent through walls by changing the rendering settings of the opponent model and highlighting it with a strong color.

The third one is Shield. This adds a temporary protection effect. A sphere is visually attached around the player, and the shield prevents some negative effects such as stun.

Another offensive effect is Stun. It targets the opponent and temporarily blocks their movement. However, if the opponent has an active shield, the stun does not apply.

To make the gameplay less predictable, a RandomPowerUp system was added. Instead of spawning one drone per effect, the game now uses one drone type that gives a random power-up when collected. This keeps the world visually simple while creating variety during gameplay.

```
options = [
    lambda: PowerUp(name="SpeedBoost", attr_name="speed", multiplier=1.22, duration=6.0),
    lambda: PowerUp(name="JumpBoost", attr_name="jump_speed", multiplier=1.35, duration=5.0),
    lambda: StunOpponentPowerUp(duration=2.4),
    lambda: InvisibilityPowerUp(duration=5.0),
    lambda: RadarOpponentPowerUp(duration=6.0),
    lambda: ShieldPowerUp(duration=5.0),
]
```

Finally, the entire system is integrated into the main loop in game/game.py. The collectable manager is instantiated once, then updated every frame. This means that drones can spawn, move around

the map, detect player contact, and activate random temporary powers correctly during.

```python
def update(self, task):
    raw_dt = self.app.clock.getDt()
    self.last_tick += raw_dt

    if self.last_tick > 1 / SERVER_TICK:
        self.serverUpdate = True
        self.last_tick = 0
    else:
        self.serverUpdate = False

    dt = min(raw_dt, PHYSICS_MAX_DT)
    self.player.update(dt)
    self.hud.update()
    self.collectables.update(dt)
```

In conclusion, the collectable drone and power-up system was implemented as a modular gameplay feature combining Panda3D rendering, Bullet ghost collisions, timed power-up effects, and random movement across the map. This design allows the game to remain expandable, since new effects or new drone behaviors can be added without changing the whole structure.

I also implemented part of the HUD system to display the most important information directly on screen during the game. It includes the current round, the player's role, the match status, and a small help section for useful controls. I also gave it a neon futuristic style in HUD.py, then connected it to the main game and tag game logic in game.py and TagGame.py so that it updates in real time during gameplay.

```python
class HUD(ShowBase):
```

Milan PUIJALON – *Creative Director*

Objectives

There are 3 main objectives:

- Create a high quality soudtrack that reflects the spririt and atmosphere of the game.
- Create characters that fit well into the game's universe and can be implemented in a parkour and chase game.
- Create this universe with a convincing map that connects to the gameplay.

More specifically:

- The soundtrack must convey a cyberpunk universe—a chaotic, futuristic city where anarchy reigns and everything is becoming robotic, but without forgetting a certain sense of sadness about the decay of the human condition and its amorality.
  To highlight these two dominant aspects, the soundtrack should include both calm, detached music and more aggressive, dynamic music.
- The characters must be futuristic and semi-robotic to fit into the cyberpunk universe. They must also connect to the gameplay by representing the game of tag.
- The map must convey a convincing cyberpunk universe while being functional for parkour and chase gameplay. Players must be able to interact with environmental elements and use them to their advantage.

Planning

How to achieve these different objectives in terms of time and technique?

- **Music**

Since I was young, I have played various instruments, especially drums. So, I already have some musical experience. Additionally, for a few years, I have been composing music using my computer and have already released an album on streaming platforms under the name stylo bleu.

I plan to continue creating music continuously and integrating it into the video game, but this time with time, style, and spatial constraints in the creation process.

- **Characters**

I have no past experience in creating 3D models. So, it is necessary to start by learning how to use quality software for 3D model creation, while experimenting on my own and trying to create models from scratch. Also, I need to think about the artistic direction for the characters.

- **Map**

Knowing how to use 3D modeling software is also necessary for the map. Additionally, I need to start thinking about the layout of the city so that it is functional for parkour while remaining immersive. However, since creating a map takes time, I will start by thinking about the atmosphere I want to give it, but prioritize the characters and learning the software.

Progress

- **Music**

When I started working on the project, I focused on music creation because it was the only area where I had experience.
But this time, I had constraints to ensure my creations aligned with the game's artistic direction, as well as spatial and time constraints.
The artistic direction decided with the group is a cyberpunk aesthetic. To fit this aesthetic, I decided to create music primarily in these two genres:

1. Electronic music, for its futuristic and technological feel, as well as its dynamic side that can emphasize the tension during chases between the hunter and the prey.
2. Chill-hop, a style of instrumental hip-hop that is relatively calm, reflecting the melancholy of the modern world in which the characters evolve.

For music creation, I primarily use Studio One, a DAW with very interesting features when you know how to use it properly. However, it requires some investment to master, which is why, alongside creation, I also watch tutorials or research how the DAW works.
I compose mainly using a MIDI master keyboard, which allows me to send information to my computer and use a wide variety of sounds without necessarily needing to record acoustic instruments. This is very useful when you don't have space for physical instruments at home.

Screenshot of Studio One:

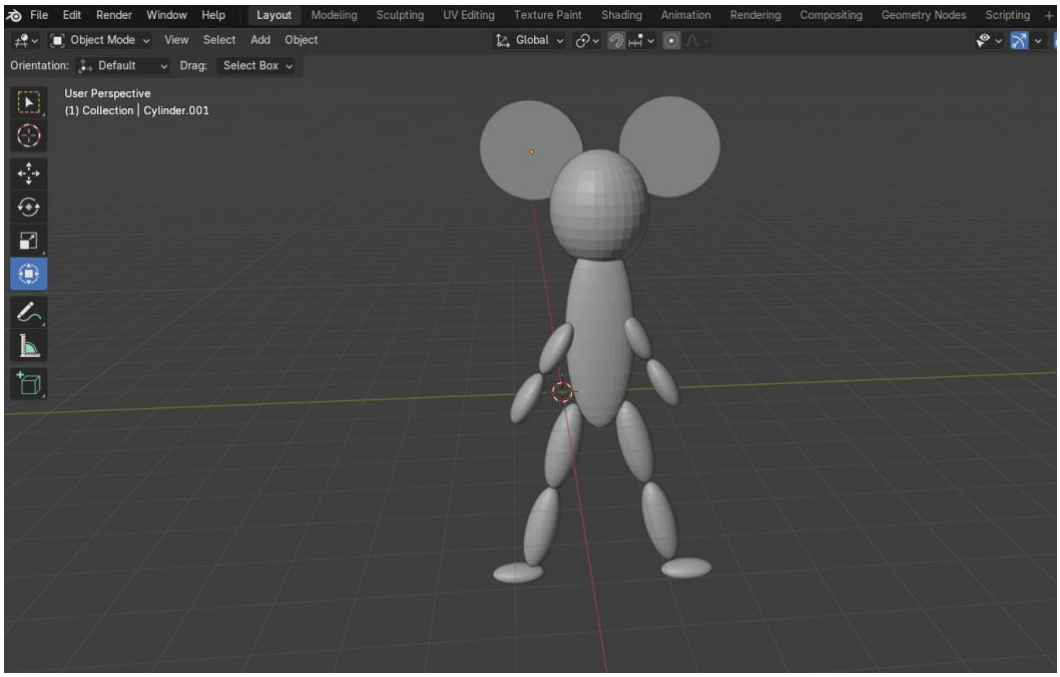
We started to think about the repartition and implementation of the musics created, for instance the menue music has already been implemented in the game.

- **Characters**

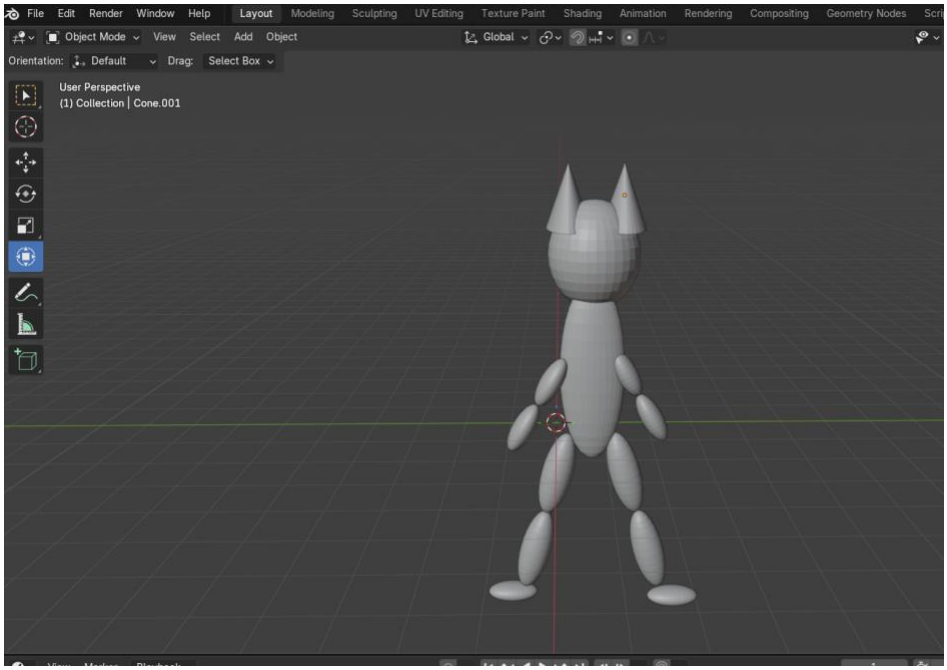
Before starting to create characters, I needed quality software for 3D modeling. I chose Blender and started watching tutorials online and learning how it works. But I quickly realized that Blender is very difficult to get started with: the controls are not intuitive, there are many possible operations on different shapes, and many shortcuts to know to create efficiently. I soon felt lost about how Blender works and forgot the commands and shortcuts I had learned because I had never practiced them. So, I decided to start creating simple shapes and trying to perform operations on them while learning how Blender works in parallel, aiming to better retain the information I was learning. After some time, I started creating characters by combining basic shapes.
I started by taking spheres and stretching them to give them the shape I wanted. Then, I tilted and moved these spheres to combine them. The result gave me a reference body that provides an idea of the future appearance of the characters once completed.
Given the game's concept—a tag game or "jeu du chat et de la souris" (cat and mouse game)—two distinct and easily recognizable characters were needed so that players could tell at a glance which is the hunter and which is the prey.

One of the characters had to be a mouse (the prey), and the other a cat (the hunter).
Here are images of these two 3D models:

The mouse:



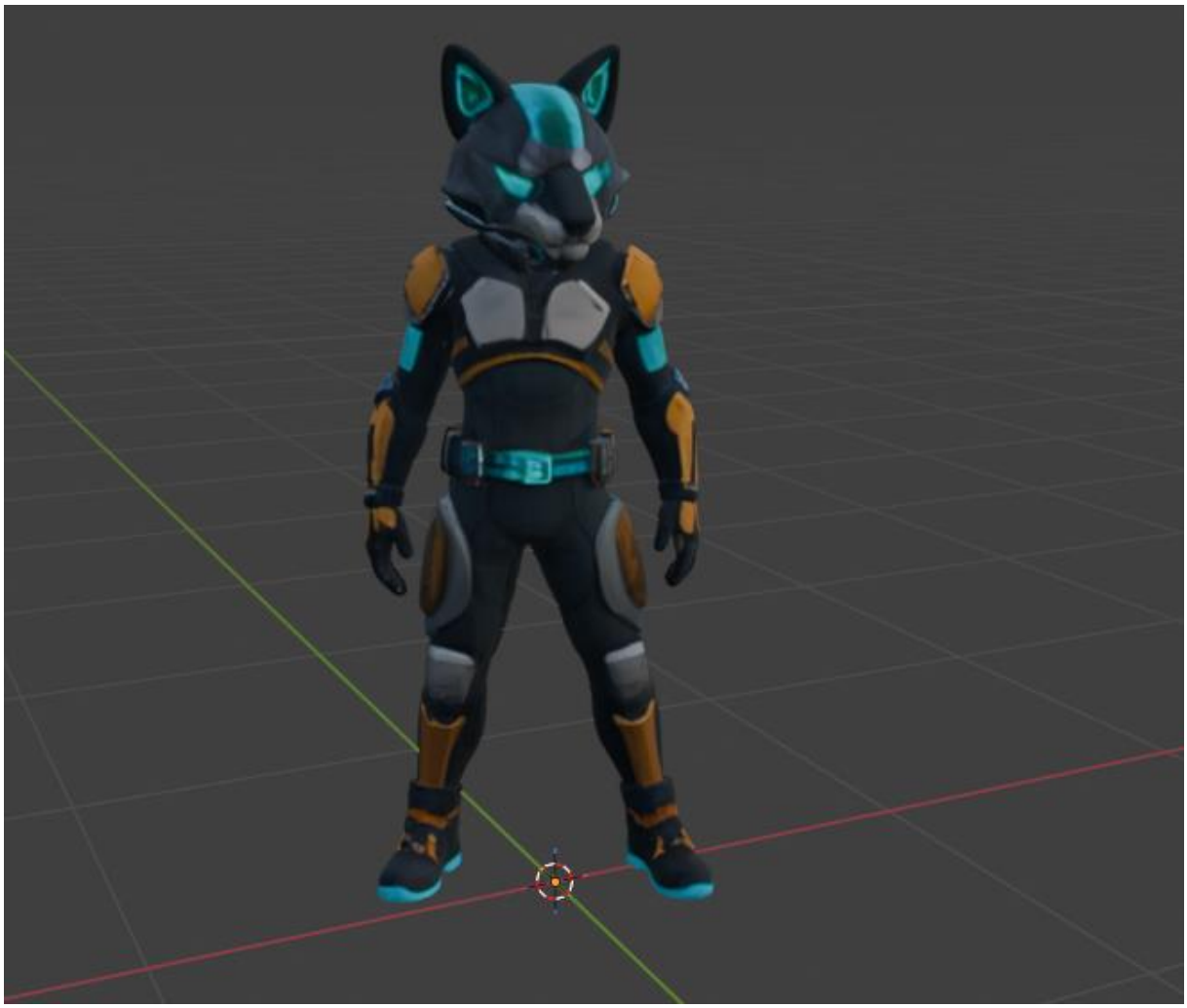
The cat:



After improving my skills, and watching a few tutorials on the internet, I was finally able to use blender efficiently to create some coherent characters that fit right in the cyberpunk world that we intended to create from the start. As a reminder, it was intended

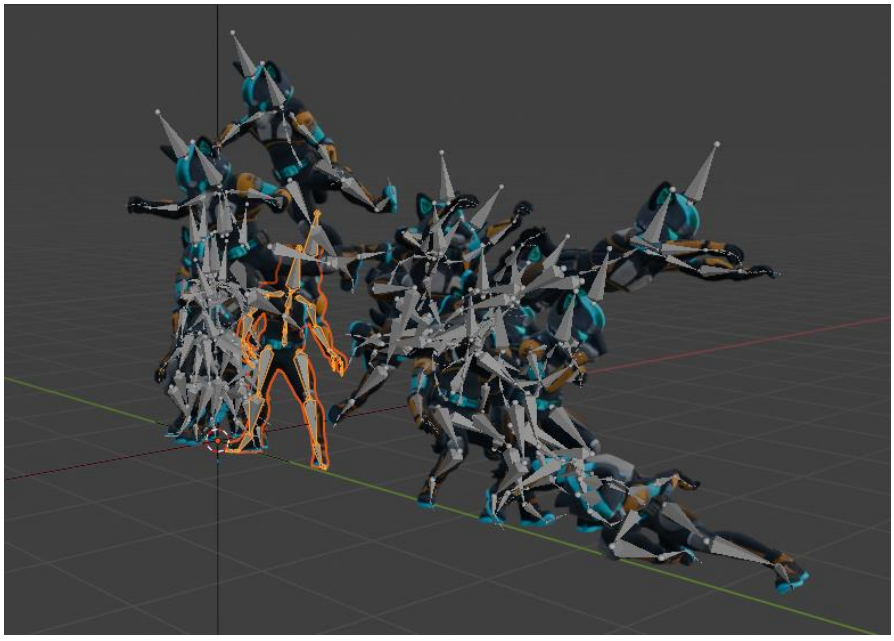
that we would create 2 characters: the hunter wich is a cat and the hunted wich is a mouse. Both of them should have either sober or neon lights colours, and overall a futuristic designs with robotical aspects.
Here are the new characters 3d models:

The cat:



The mouse:

With the characters, I also implemented some animations that, when implemented in the bam file that we use in panda 3d, should be used to suit the movements that were coded by the rest of the team.
Here is what the file looks like with all the animations at the same time:



Gregory-Lucas CLEMENT – *Team Coordinator*

Objectives

My main objective in this project is to design and implement the HUD (Heads-Up Display) of our game, Rooftop Rivals.
The HUD must clearly communicate essential information to the players (timer, roles, scores, active SUPER Powers) without breaking the immersion of our cyberpunk atmosphere.
I also aim to ensure that the interface remains readable and responsive, whether the players are in intense chase phases or in calmer moments.

HUD Design and Information Choices

Before writing any code, I started by defining what information should appear on the screen and where.
We decided that the HUD should at least display: the remaining time for the round, the current roles of each player (mouse or hunter), the score for each player, and the currently active SUPER Power, if any.
I sketched several layout ideas to avoid overloading the screen and to keep the central area as clear as possible for gameplay.
For example, the timer is placed at the top center, the scores at the top corners, and power-related information closer to the bottom of the screen.

Technical Implementation in Panda3D

To implement the HUD, I used Panda3D's 2D interface tools, in particular OnscreenText and simple 2D elements anchored to the screen.
The first step was to create a structure that allows the HUD to be updated every frame or whenever an important game event occurs (role swap, end of round, power pickup, etc.).
I worked closely with Mehdi, who manages the game logic, to connect the HUD to the variables that control the round time, the roles, and the scores.
When a value changes in the game logic (for example, a new round starts or a SUPER Power is activated), a dedicated function updates the corresponding HUD element.

Visual Identity and Readability

Since the game has a cyberpunk identity, I chose colors and fonts consistent with this universe: dark backgrounds with neon accents (cyan, magenta, green) similar to the menus and the website.
However, I had to balance style and readability; for instance, I avoided using too saturated colors for the text to keep it legible on top of the 3D scene.
I also experimented with subtle effects such as small glow or shadows on text to make the HUD readable even in bright areas of the map.

Challenges and Work in Progress

One of the main challenges was synchronizing the HUD with both the round system and the multiplayer logic.
The HUD must always display the correct information for the local player, even when roles are swapped or when a new player joins a game.
Another difficulty was updating the HUD without degrading performance or cluttering the code; I had to refactor some parts to centralize the update logic in clean functions.
For the next semester, I plan to improve the HUD by adding animations (for example, flashing the timer when the round is almost over) and better visual feedback when a SUPER Power is picked up or ends.

CONCLUSION

In this second part of the project, everyone tried to learn and work as much as we could to build solid foundations for the videgame. Even if there were some incidents, we kept going and tried to do our best. We made a table to represent our progress and our objectives for the next steps.

Tasks	Soutenance 1	Soutenance 2	Soutenance 3
Movements	15 %	70%	100 %
Menus	30 %	80 %	100 %
Multiplayer	80 %	70 %	100 %
Sounds	50 %	75 %	100 %
Map & 3D	10 %	70 %	100 %
Website	30 %	85 %	100 %
AI	10 %	50 %	100 %
GameMode & HUD	20 %	60 %	100 %

The next step will be to upgrade our game, fix all the bugs and try to have the cleaner game as possible.

BIBLIOGRAPHY

https://docs.panda3d.org/1.10/python/index
https://www.w3schools.com/python/
https://www.youtube.com/
https://github.com/
https://stackoverflow.com/questions
https://www.youtube.com/watch?v=_5Js5pbvFSw
https://www.youtube.com/watch?v=1FGWgaCyE8E&list=PLuine2he2FmOY1ILTDc1OR9vHgIMBw4_W