ACG Final Report

Yiping Liu[*]
liu-yp23@mails.tsinghua.edu.cn
2023010874

Mengjie Zhao
zhaomj23@mails.tsinghua.edu.cn

Figure 1: A poster of our project.

1 Introduction

In this project, we aim to simulate the experience of flying a real FPV (First-Person View) drone, a challenging task that requires precise control and excellent hand-eye coordination. To replicate the complexity of a real drone flight, our game introduces the need for four separate key bindings to control the four degrees of freedom, testing the player's skill and precision. The game also features bullets with tracking capabilities, adding an extra layer of strategic gameplay.

The objective is for players to fly through as many rings as possible while avoiding or destroying self-destructing enemy drones. Successfully passing through rings or eliminating enemies earns points, and the goal is to achieve the highest score. Additionally, our game supports a two-player mode, where the players must compete to be the first to pass through each ring, or eliminate each other. Since each ring can only be passed through once, this creates an exciting competitive dynamic.

In real-world FPV drone racing, pilots use specialized joysticks and display equipment for control, while keyboard operation is less intuitive. Thus, designing an effective control scheme for our simulation was a significant challenge. Our solution is to control the drone using a keyboard that simulates a mechanical joystick, and the joystick's parameters are then used to manipulate the drone's flight dynamics. This method required redesigning the drone control algorithm to account for the mechanical joystick's input. Through careful tuning of parameters, we aim to provide the best possible control feel, though mastering the game remains challenging for players.

[*]The author of this report.

2 Method

2.1 Scene layout

This subsection corresponds to the part of *Scene layout: reasonable object geometry, textures, and materials*. This part is basic.

In our project, we utilize `Three.js` to create a well-structured scene layout by carefully selecting object geometries, textures, and materials. For object geometry, we use built-in shapes like cubes, spheres, and custom 3D models that align with the scene's design. Textures are applied to give surfaces realistic detail, while materials such as `MeshStandardMaterial` and `MeshPhongMaterial` are chosen to achieve the desired visual effects, including lighting interactions and reflections. This combination ensures a visually appealing and interactive 3D environment.

2.2 Environment lighting

This part is worth 1pt.

We implement environment lighting in `Three.js` by using various light sources such as `AmbientLight` and `DirectionalLight`.

In addition, we use `Three.js`'s environment mapping to implement environment reflection, which allows objects to achieve relatively realistic reflection effects while improving performance.

To achieve this, in addition to setting up ambient light and directional light in the scene, we can load an HDR texture representing the background and set both `scene.background` and `scene.environment` to this texture. Figure 2 demonstrates this effect, where we can see that the object's surface reflects the color of the background.

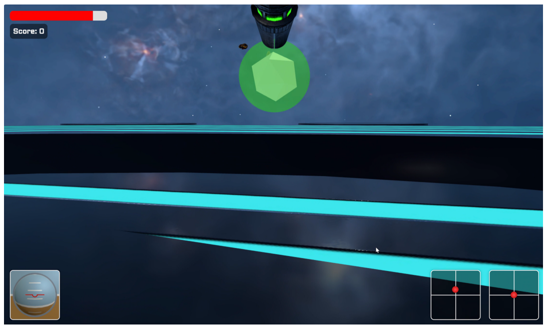


Figure 2: This environment light.

2.3 Synchronized audio

This part is worth 1pt.

We use `Three.js` to implement audio by creating an `AudioListener` and attaching it to the camera, which allows the listener to perceive sound in the 3D space.

For positional audio, we utilize `PositionalAudio` objects, which are linked to specific 3D objects, enabling sound to vary in volume and direction based on the listener's position relative to the audio source.

2.4 Animation

This part corresponds to the part *Include animations, whether simple keyframe animations or more complex ones*, which is basic.

We use the `requestAnimationFrame()` method to implement animations, which schedules the next frame of the animation to be rendered.

2.5 Eliminating Stuttering and Jitter by Smoothing Frame Rate Instability

This part is not on the grading sheet, but we believe it is worth at least 1 point, as the method used here is simple but powerful.

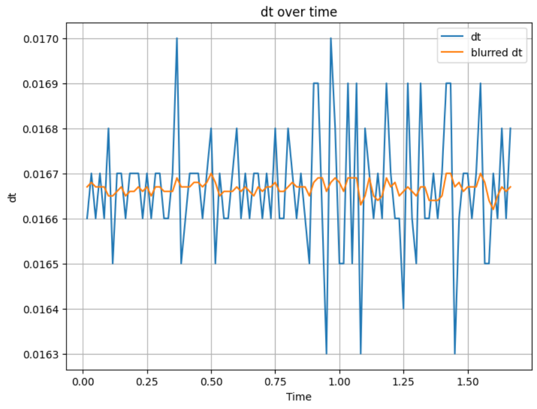


Figure 3: *dt* over time.

To implement movement, the common approach is to use the time difference between two frames as dt, approximating the movement of an object between those frames.

In our game, the movement speed of objects is very fast, and the method for calculating object movement is relatively complex. For example, in first-person perspective, the camera follows the complex movements of a drone, and in third-person perspective, the camera adds inertia, among other factors.

However, unstable frame rates often occur, such as when the game starts, when certain events are triggered, or when switching tabs. If a naive implementation is used, frame rate instability could cause the movement of the camera and certain objects to become uneven, resulting in noticeable stuttering and jitter that shouldn't exist.

To solve this problem, we need to eliminate the high-frequency components in the dt frequency. Inspired by anti-aliasing algorithms, we use the following method to calculate the actual dt for each frame:

$$dt_{blurred}[i] = \frac{1}{k}\sum_{j=0}^{k-1} dt[i-j]. \qquad (1)$$

That is, we take the average of the previous k frames. This approach can almost completely eliminate stuttering and jitter. Figure 3 shows the difference between the processed dt and the unprocessed dt over time.

2.6 Collision handling

This part is worth 2pts.

We use `cannon-es` as our physics engine to handle collisions. It provides built-in support for simulating elasticity and friction. Additionally, we detect collision events and react based on the information of each event.

For example, we use the `getImpactVelocityAlongNormal()` method to get the velocity along the normal, and apply the corresponding damage to the object based on it.

2.7 Control of the main game character

This part is basic, but given the complexity of the control algorithm we use, I believe this deserves at least 1 extra point.

The control of a drone is quite complex. A real-world drone is controlled by two joysticks, each controlling two degrees of freedom. Specifically:

- The up/down direction of the first joystick controls the throttle, i.e., the vertical force.
- The left/right direction of the first joystick controls the yaw speed.
- The up/down direction of the second joystick controls the pitch speed.
- The left/right direction of the second joystick controls the roll speed.

Additionally, the throttle size slightly affects the drone's ability to control its attitude. When the throttle is zero, the drone's attitude cannot be controlled at all. As the throttle increases, the ability to hold the attitude becomes stronger, reaching its maximum at a certain value.

We chose to use the keyboard to control the drone, so we need four sets of keys (a total of eight keys) to control the four degrees of freedom. The challenge with keyboard control is that it only has two states—“pressed” or “not pressed”—and cannot control the force like a joystick to smoothly control the speed. Our solution is to use the keyboard to control two virtual joysticks, and these two joysticks will control the drone.

Keyboard-Controlled Joystick. Each joystick's two dimensions are independent, and we consider one dimension here. The joystick's coordinate x is constrained to $[-1, 1]$, and the joystick's output is its coordinate x. The state of the keyboard at each frame is a value target $\in \{-1, 0, 1\}$. The joystick's speed is then given by:

$$v = \text{target} \cdot \text{pullForce} - x \cdot \text{springForce},$$

where pullForce and springForce are constants representing the pull force and spring force.

The joystick also needs a "dead zone," meaning it tends to stay at the origin when close to it. To implement this, after each frame of motion, if $|x| \leq \text{deadZone}$, we multiply x by 0.99.

By independently setting the values of each dimension for each joystick, we can balance the feel of different operations.

Joystick Control of the Drone. Let the current frame's joystick inputs for throttle, yaw, pitch, and roll be $c_{\text{throttle}}, c_{\text{Yaw}}, c_{\text{Pitch}}, c_{\text{Roll}} \in [-1, 1]$. We also define constants $L_{\text{throttle}}, L_{\text{Yaw}}, L_{\text{Pitch}}, L_{\text{Roll}}$ that represent the limits of the corresponding degrees of freedom. By multiplying the corresponding input and limit, we get the control parameters for the drone at this frame: throttle, ΔYaw, ΔPitch, ΔRoll. We aim to apply an upward force of size throttle and want the drone's angular velocity to be $\omega_c = \omega(\Delta\text{Yaw}, \Delta\text{Pitch}, \Delta\text{Roll})$.

First, calculate the parameter:

$$p_r = \frac{\min(0.1, |c_{\text{throttle}}|)}{0.1} \times 0.9$$

which represents the drone's ability to control its rotation. The next frame's angular velocity is:

$$\omega_{\text{next}} = (1 - p_r)\omega_{\text{last}} + p_r\omega_c.$$

Next, calculate the parameter:

$$p_v = \min(0.5, |c_{\text{throttle}}|) \times 0.01$$

which represents the drone's ability to control its translation. We multiply this by the velocity to simulate the drone's control over its attitude.

Finally, apply a force with size throttle in the upward direction to complete the control for this frame.

2.8 Camera motion control

This subsection corresponds to the part of *Camera motion control either in third-person or first-person view*. This part is basic.

To achieve this, we only need to set the relative position and angle between the camera and the object in both the first-person and third-person views, and then assign a switch key to allow toggling between the two views at any time.

Additionally, sometimes the objects viewed in the two perspectives are different. For example, in the first-person view, we do not want to see the body, and the position of the weapon may need to be adjusted in the view. We can use the ‘layer‘ functionality in `Three.js`, setting two layers to represent the objects visible in the first-person and third-person views. Then, we assign the corresponding layer to each object. We can also set a layer to display the collision boxes, which will be shown when needed.

Figure 4 illustrates both the objects visible in the first-person and third-person views, as well as the collision boxes. The white box in the image represents the collision box, and the small gun in the top right corner is what would be visible in the first-person view, but it is not displayed in the third-person view.

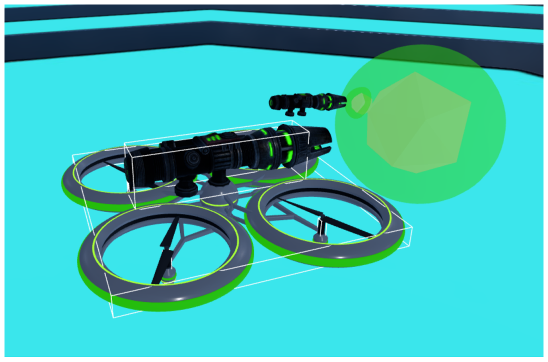


Figure 4: The objects visible in first-person and third-person views.

2.9 Camera shakes

This part corresponds to the part of *Implement suitable camera shakes during collisions or movement to enhance immersion*, which is worth 2pts.

To implement this, we only need to add a small random displacement to the camera in each frame.

2.10 A complex third view camera

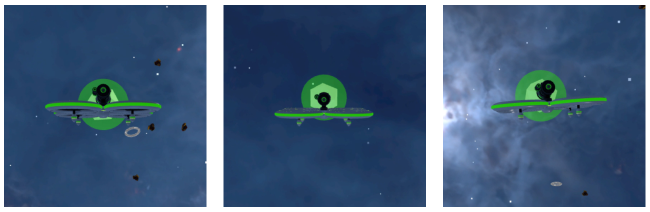


Figure 5: Camera rotations.

We implemented a more complex third view camera that has inertia and cannot go through the walls. We believe this deserves at least 2 point.

To avoid dizziness during flight, we need to add inertia to the third-person camera, meaning the camera's position will slightly shift based on the current controls. After experimentation, we found that the best effect in preventing dizziness is achieved when both rotational inertia and distance inertia are applied simultaneously.

Additionally, in third-person view, if there is an obstacle behind the drone, we need to prevent the camera from passing through the obstacle.

To achieve rotational inertia, we calculate the current angle of the camera relative to the drone, denoted as ω_{now}, and the target angle (i.e., the angle without inertia), denoted as ω_{goal}. Let the time between two frames be Δt seconds. The camera's rotation for the next frame is set as:

$$\text{slerp}(\omega_{\text{now}}, \omega_{\text{goal}}, 1 - 0.9^{100\Delta t}).$$

See Figure 5 for different camera rotations.

For distance, we need to consider both inertia and obstacles. We can use `Three.js`'s `Raycaster` to detect if there are any obstacles along the camera's path to the drone and determine the position of the closest obstacle to the drone.

Case 1: There are obstacles. The camera's distance for the next frame is set as:

$$\max(d_{\text{min}}, d_{\text{inter}}, d_{\text{now}} - v_{\text{max}}\Delta t),$$

where d_{min} represents the minimum distance limit, d_{inter} represents the position of the closest obstacle to the drone, and d_{now} represents the previous frame's camera distance. v_{max} represents the speed limit.

Case 2: No obstacles. Let $\vec{r}$ be the vector from the camera's target position to the drone's current position, and let $d_{\text{goal}} = |\vec{r}|$, $\hat{r} = \vec{r}/d_{\text{goal}}$, and $\Delta\vec{x}$ be the drone's displacement in this frame.

There are two stages in calculating the distance for the next frame. To implement inertia, we first calculate:

$$d_1 = d_{\text{now}} + \text{clamp}(\Delta x \cdot \hat{r}, -1, 1),$$

where $\text{clamp}(x, x_l, x_r) = \max(x_l, \min(x_r, x))$.

Then, to ensure repositioning after moving away from an obstacle or when the drone stops moving, we calculate:

$$d_2 = d_{\text{goal}} + \text{clamp}\left((d_1 - d_{\text{goal}}) \cdot k^{\Delta t}, -v_{\text{max}}\Delta t, v_{\text{max}}\Delta t\right).$$

If d_2 is unobstructed, we set the camera distance to d_2.

2.11 UI Design

This subsection corresponds to:

- Proper game start and end interfaces (basic)
- Additional auxiliary interfaces (up to 2pts)
- User-friendly layout with visually appealing design (1pt)

We build a scene using `Three.js` to implement the start and end interfaces, with property animations. We place the player in a sci-fi themed room, where they can interact with objects in the room to select a drone and start the game. It looks **extremely cool**. See Figure 6.

Moreover, we designed a blood bar, an attitude indicator, a score displayer, a pair of joystick displayers, and a warning sign when the drone leaves the map border. See Figure 7.

The attitude indicator is a ball that shows the drone's attitude. We use another 3D scene to show it, and render it separately.

The blood bar features a built-in mechanism that displays a trace when the player is injurede. The trace moves at a speed of kx, where x is the difference between the current trace position and the actual health.

Also, we set a red border around the screen indicates that the player is injured, The width of the border depends on the severity of the injury.

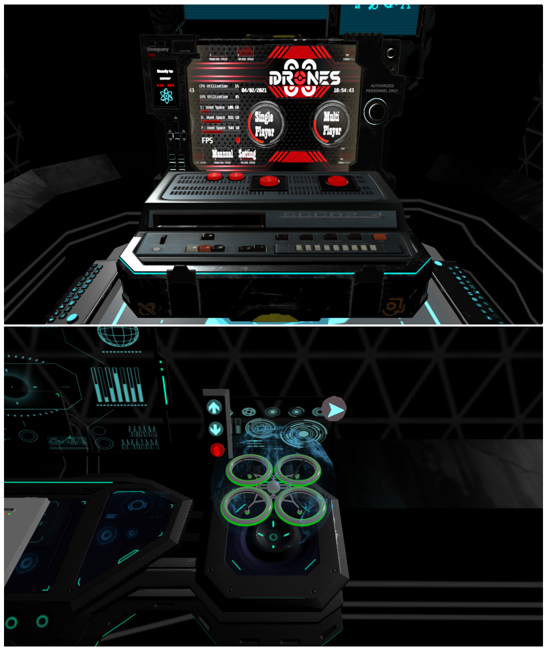
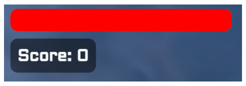
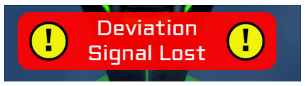


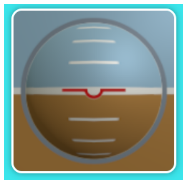
Figure 6: The start screen.



(a) The blood bar and the score displayer



(b) The warning sign


(c) The attitude indicator

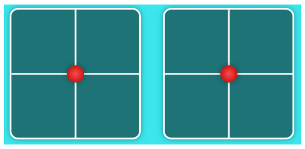

(d) Joystick displayers

Figure 7: Auxiliary interfaces.

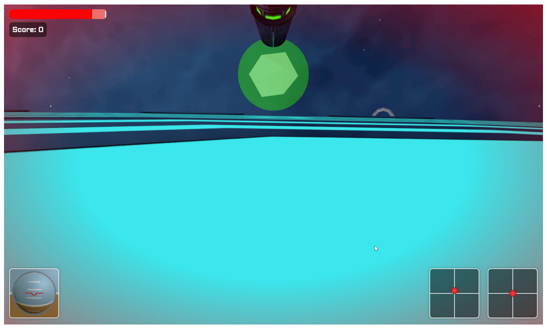


Figure 8: When the player is injured.

Figure 8 shows the blood bar trace and the border.

2.12 Entity AI

This part is not on the grading sheet, but we believe it is worth at least 2 point as it's hard to implement.

The game features two types of strategic entities: bullets with tracking functionality and enemies with AI.

The strategies of these entities can be seen as *Finite State Machine*, having different states that transition to other states after certain events. For example, the states of a bullet include "Charging," "Flying," and "Explosion," while the states of an enemy include "Wandering," "Chasing," and "Returning." The enemy's movement is further divided into three stages: "Searching for Target," "Turning," and "Moving." Each state has a corresponding strategy, and we omit the formulas for these strategies as they are trivial.

2.13 Explosion simulation

This part is not on the grading sheet, but we believe it is worth at least 1 point.

To achieve a realistic explosion, we implemented a complex particle system with five layers of particles. The properties of each type of particles are shown in Table 1.

The parameters for the five types of particles are shown in Table 2, where Rand(l, r) denotes a uniformly random real number in the interval $[l, r]$, and Shell(l, r) denotes a uniformly random vector within a spherical shell with radius $[l, r]$. Figure 9 shows a picture for each type of particles.

Property	Description
$\vec{x_0}$	Initial position, where the origin is the center of the explosion.
$\vec{v_0}$	Initial velocity.
r_0	Initial size.
o_0	Initial opacity.
s	Generation time, the particle is generated at time s after the explosion begins.
L	Particle lifetime, the particle disappears after time L.
$f_r(x)$	A function from $[0, 1]$ to $[0, 1]$. At time t after birth, the size is $r_0 \cdot f_r(t/L)$.
$f_v(x)$	A function from $[0, 1]$ to $[0, 1]$. At time t after birth, the velocity is $\vec{v_0} \cdot f_v(t/L)$.
$f_o(x)$	A function from $[0, 1]$ to $[0, 1]$. At time t after birth, the opacity is $o_0 \cdot f_o(t/L)$.

Table 1: Particle properties

2.14 Heating effect

This part is not on the grading sheet, but we believe it is worth at least 1 point.

In the game, the enemy will heat up before self-destructing, so we need to simulate its material during the gradual heating process.

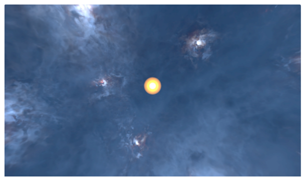
The metal will transition unevenly between red and orange as it heats up. The gradient coefficient at position (x, y) $(0 \le x, y \le 1)$ is defined as

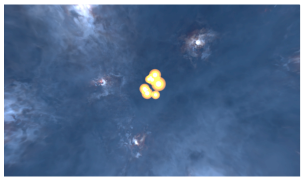
$$f(x,y) = \frac{1}{2}\left(\frac{\sum_{i=0}^{5} 2^i \sin(2^i(x + \mathrm{kx}_i))\cos(2^i(y + \mathrm{ky}_i))}{\sum_{i=0}^{5} 2^i} + 1\right)$$

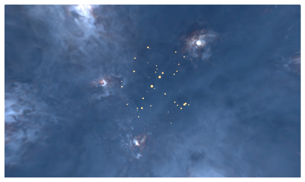
Property	Flash	Fire	Sparks
Count	1	10	50
$\vec{x_0}$	$\vec{0}$	$R \cdot \mathrm{Shell}(0.1, 0.7)$	$R \cdot \mathrm{Shell}(0.1, 0.7)$
$\vec{v_0}$	$\vec{0}$	$R \cdot \mathrm{Shell}(0.5, 3)$	$\frac{\vec{x_0}}{\lvert\vec{x_0}\rvert} \cdot \mathrm{Rand}(10, 15) \cdot R$
r_0	R	$R \cdot \mathrm{Rand}(0.5, 1.5)$	$R \cdot \mathrm{Rand}(0.02, 0.1)$
o_0	1	1	1
s	0	$\mathrm{Rand}(0, 0.1)$	$\mathrm{Rand}(0, 0.1)$
L	0.4	$\mathrm{Rand}(0.2, 0.4)$	$\mathrm{Rand}(0.2, 0.4)$
f_r	$1 - x$	$1 - x^3$	$1 - x^3$
f_v	1	1	$1 - x^3$
f_o	1	1	$1 - x^3$

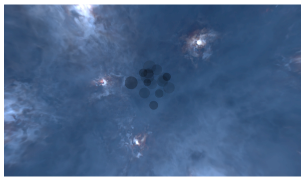
Property	Smoke	Debris
Count	20	20
$\vec{x_0}$	$R \cdot \mathrm{Shell}(0.3, 1.2)$	$R \cdot \mathrm{Shell}(0.5, 1)$
$\vec{v_0}$	$R \cdot \mathrm{Shell}(0.5, 3)$	$\frac{\vec{x_0}}{\lvert\vec{x_0}\rvert} \cdot \mathrm{Rand}(5, 10) \cdot R$
r_0	$R \cdot \mathrm{Rand}(0.3, 0.5)$	$R \cdot \mathrm{Rand}(0.02, 0.1)$
o_0	$\mathrm{Rand}(0, 0.3)$	$\mathrm{Rand}(0, 0.7)$
s	$\mathrm{Rand}(0, 0.2)$	$\mathrm{Rand}(0, 0.1)$
L	$\mathrm{Rand}(0.5, 1)$	$\mathrm{Rand}(0.3, 0.6)$
f_r	$1 - x^3$	$1 - x$
f_v	$1 - x^3$	$1 - x^2$
f_o	$1 - x^4$	$1 - x^4$

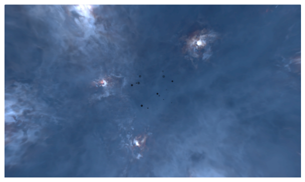
Table 2: Particle parameters.


(a) Flash.


(b) Fire.


(c) Sparks.


(d) Smoke.


(e) Debris.

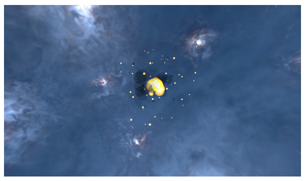

(f) Combine them.

Figure 9: The explosion.

where kx_i and ky_i are uniformly random values in the range $[0, 2\pi]$. It is easy to observe that $f(x, y) \in [0, 1]$.

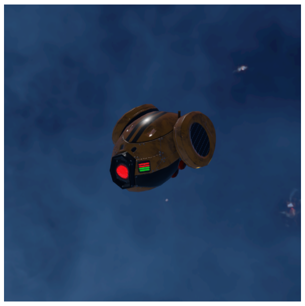
We use this coefficient to perform a gradient from orange to red, and the resulting color is denoted as $c(x, y)$.

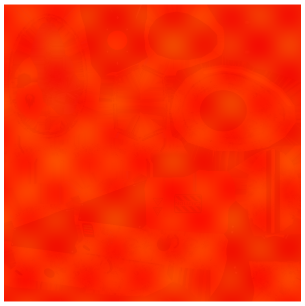
Next, we calculate the texture after heating. We extract the original texture of the enemy and then modify the color of the pixel at position (x, y) to

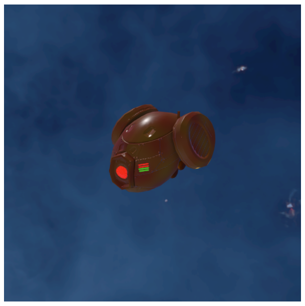
$$\mathrm{color}_{\mathrm{new}} = \mathrm{lerp}(\mathrm{color}_{\mathrm{old}}, c(x/W, y/H), 0.95)$$

where W and H are the width and height of the image, respectively. Figure 10b shows the modified texture and Figure 10c shows the enemy after the texture change.

Finally, we moderately scale up the modified enemy, change its blending mode and transparency, adjust its emissive map to match the texture, and adjust its transparency and emissive intensity in real time based on the enemy's temperature to achieve the effect of gradual heating. Figure 10d shows the final effect.


(a) The original enemy.


(b) The modified texture.


(c) After changing the texture.

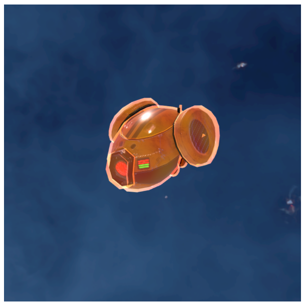

(d) The final effect.

Figure 10: Heating effect.

2.15 Multi-player mode

This part is worth 3pts.

Our game supports a two-player mode. We simply create two elements on the left and right sides of the webpage, and replicate all scenes and components in both elements. Our code structure allows this operation to be achieved by passing the elements as parameters. The effect is shown in Figure 11.

Additionally, to leverage the renderer's cache, the same 'renderer' is used for both the left and right sides, which greatly reduces memory usage.

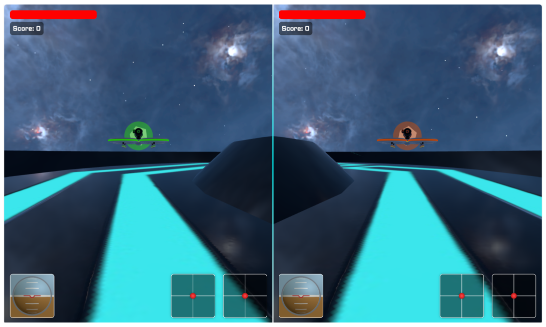


Figure 11: Multi-player modes.

2.16 Customizable character appearance

This part is worth 2pts.

We have set up two drones for the player to switch between freely.

On the code level, we have implemented logic that allows the player's color theme to be changed. After changing the theme, all colors related to the player will be updated. However, for the convenience of gameplay, the theme is pre-defined rather than being freely selectable by the player.

2.17 User manual

This part corresponds to the part of *User manual or instructions*, which is basic.

Our gameplay manual is integrated with the gameplay itself, rather than just being text-based instructions.

We have implemented a dedicated scene for tutorial purposes, which includes guidance and instructions for the next steps, helping players better understand the game mechanics. The specific effect can be seen in Figure 12.

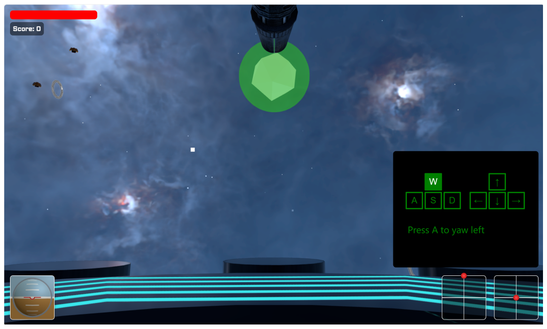


Figure 12: Manual mode.

2.18 Online access

This part corresponds to the part of *Easy installation or online access*, which is worth 2pts.

We have deployed the game on GitHub Pages, and it can be accessed at https://he-ren.github.io/ACG_Project_2024Fall/ for playing anytime.

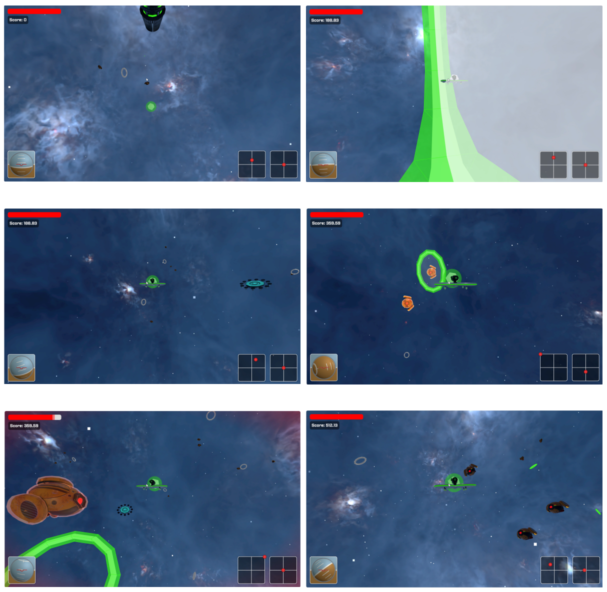


Figure 13: Screenshots.

3 Results

For technical details, please refer to the images displayed earlier. Figure 13 shows additional screenshots of the game. For more details, please visit our game website at https://he-ren.github.io/ACG_Project_2024Fall/ to play.

4 Discussion

Although we have implemented many details, the game still feels quite thin. Therefore, more rich elements need to be added in future development.

Additionally, our code structure allows for easy integration of gamepad support, which can be implemented in future development.

Moreover, because the required keybindings are very complex, it is inconvenient for two players to use a single screen and a single keyboard. Therefore, an online multiplayer mode can also be added in the future.

5 Personal Contribution Statement

The following parts of the game were implemented by me:

(1) *Scene layout: reasonable object geometry, textures, and materials* (basic).
(2) *Include animations, whether simple keyframe animations or more complex ones* (basic).
(3) *Eliminating Stuttering and Jitter by Smoothing Frame Rate Instability*. This part is not on the grading sheet, but we believe it is worth at least 1 extra point.
(4) *Collision handling* (2pts).
(5) *Control of the main game character* (basic), but given the complexity of the control algorithm we use, we believe it is worth at least 1 extra point.
(6) *Camera motion control either in third-person or first-person view* (basic).
(7) *A complex third view camera*. We believe it is worth at least 2 points.
(8) *Entity AI* This part is not on the grading sheet, but we believe it is worth at least 2 points.
(9) *Explosion simulation*. This part is not on the grading sheet, but we believe it is worth at least 1 point.
(10) *Heating effect* This part is not on the grading sheet, but we believe it is worth at least 1 point.
(11) *Multi-player mode* (3pts).
(12) *Customizable character appearance* (2pts).
(13) *Easy installation or online access* (2pts).

3D Models Used in the Game

The following 3D models are used in the game under Creative Commons licenses:

- **Parrot Camo drone** by `domiiniic`, used under `CC BY 4.0`. Available at https://sketchfab.com/3d-models/parrot-camo-drone-351867524b9b478fa406aad31d838ef4.
- **Drone** by `Kai Xiang`, used under `CC BY 4.0`. Available at https://sketchfab.com/3d-models/drone-fa5e5e3b0f6c4023b8e3d189cf55662f.
- **(FREE) Robot Drone SDC-01** by `SDC PERFORMANCE`, used under `CC BY 4.0`. Available at https://sketchfab.com/3d-models/free-robot-drone-sdc-01-666c2f8810494952863f9cc8bd273133.
- **Sci-Fi Door** by `thomass3278`, used under `CC BY 4.0`. Available at https://sketchfab.com/3d-models/sci-fi-door-33e67bdeee8b48ba9a617b238565430f.
- **CyberRoom** by `JuliaIce`, used under `CC BY 4.0`. Available at https://sketchfab.com/3d-models/cyberroom-scifi-fb9e951263f84af290bfb96ede790d01.
- **Button_ru** by `//[imai]`, used under `CC BY 4.0`. Available at https://sketchfab.com/3d-models/button-ru-71c4c4be5f1b4d5d846919e87490b03e.
- **Camera Button** by `miranda.j.rice`, used under `CC BY 4.0`. Available at https://sketchfab.com/3d-models/camera-button-c22068209db048948253a205fc5a211f.
- **Single-Channel Counting Device "SCCD2_4"** by `NeRD`, used under `CC BY 4.0`. Available at https://sketchfab.com/3d-models/single-channel-counting-device-sccd2-4-afd69a65d90f4844bfd92edb042ae132.
- **Panel (product design)** by `Mrhigh`, used under `CC BY 4.0`. Available at https://sketchfab.com/3d-models/panel-product-design-fbe1c3c2d109457486a50ff41508a512.
- **Sci-Fi Panels** by `Xavi Pujadas`, used under `CC BY 4.0`. Available at https://sketchfab.com/3d-models/sci-fi-panels-c1243c5a9e954c539b2c181c886dad62.
- **gun satellite panel Computer** by `Mehdi Shahsavan`, used under `CC BY 4.0`. Available at https://sketchfab.com/3d-models/gun-satellite-panel-computer-cc3ca5adf3704c198800dd8dc7fafbbf.
- **Technology aperture out** by `Yuki`, used under `CC BY 4.0`. Available at https://sketchfab.com/3d-models/technology-aperture-out-a47045451f1843bfa69105f84b779717.

- **Sci Fi Panel Control** by Vladyslav Holhanov, used under CC BY 4.0. Available at https://sketchfab.com/3d-models/sci-fi-panel-control-fa12fa24d2664932988c2af9cce565a6.
- **Space nebula HDRI panorama 360 skydome** by Aliaksandr.melas, used under CC BY 4.0. Available at https://sketchfab.com/3d-models/space-nebula-hdri-panorama-360-skydome-270e7a54eb1e44fcbd5ddb2c1e537993.
- **Transformers: War For Cybertron Fusion Cannon** This work is based on "Transformers: War For Cybertron Fusion Cannon" by wickacik (available at https://sketchfab.com/3d-models/transformers-war-for-cybertron-fusion-cannon-57ac9f96d4eb4c01ab5ab4996e45318e), licensed under CC BY 4.0. We modified its color in the game.
- **Futuristic Roundtable** This work is based on "Futuristic Roundtable" by Andulil (available at https://sketchfab.com/3d-models/futuristic-roundtable-252bf93ec1d845288115581c5358be26), licensed under CC BY 4.0.
- **Drone** This work is based on "Drone" by ElliotGriffiths (available at https://sketchfab.com/3d-models/drone-ad0e1d85295847409e961de33c2ee0bf), licensed under CC BY 4.0.
- **TRON-ish low-poly drone** This work is based on "TRON-ish low-poly drone" by Bachklamk (available at https://sketchfab.com/3d-models/tron-ish-low-poly-drone-cad1fc9ada864e06ab69a37705656392), licensed under CC BY 4.0.

Advanced Computer Graphics Project Final Report: Drones

Instructed by *Li Yi*

Yiping Liu
Tsinghua, IIIS
liuyiping23@mails.tsinghua.edu.cn

Mengjie Zhao
Tsinghua, IIIS
zhaomj23@mails.tsinghua.edu.cn

1 Introduction

The project is a 3D competitive game named Drones. The main mechanism of the game is that players control the drones to pass level while fighting against the enemies, in which the fascinating 3D graphics, realistic physics engine, stunning and immersive UI design, high-difficulty operation experience, and intense battles are the biggest highlights.

1.1 Game Mechanism

The game includes single player mode and multiplayer battle mode. The goal of the level is to control the drone to navigate through all the mechanical rings while engaging in combat with enemies. The drone is equipped with a missle launcher that can be used to Fire bullets with tracking capabilities to attck the enemies, while the enemies attack by self-destructing when they are close to the drone. In multiplayer mode, since each ring can only be passed once, the players need to compete with each other or just kill each other to win the game. Both passing the ring and killing the enemies will give the player scores.

1.2 Game Features

Complex drone controls and game-running strategies, along with the optimization of player experience, are the two core pillars of this game.

First of all, the game features a realistic and smooth drone control system, simulating the flight control effects of the drone through dedicated algorithm design. The strategy of bullet tracking and enemy behavior are also implemented by complex algorithms.

On the other hand, this game is dedicated to delivering the best player experience. We provide tutorial levels to help players get familiar with the controls, along with a customization feature that allows players to choose their drone's appearance. Additionally, we offer an immersive and stunning UI design, along with thrilling real-time sound effects, and other detailed game effects, to create a top-notch gaming experience.

1.3 Game Engine and Architecture

The project is mainly developed based on Three.js and Cannon-es.js. The development of this game delves deep into the underlying architecture, with a significant portion of the systems designed and constructed from the foundational level.

The game is launched on the github page. The link is `https://he-ren.github.io/ACG_Project_2024Fall/`

2 Method

2.1 Signal System

This project implements a signal system with the ability to send, receive, and manage events through a listener and signal mechanism.

The signal system is implemented using a combination of event listeners and signals. The SignalTarget class manages the listeners and signals, and the AlarmClock class is an example of a timed event that triggers a callback. The system allows for deferred signal processing and supports interrupting signals, ensuring flexibility in how events are handled.

2.2 Drone Control

The drone controls are designed to interact with both the hardware (i.e., the drone itself) and the software, enabling the drone to follow specific commands, adjust its altitude, orientation, and speed, and perform necessary maneuvers.

The drone control system is mainly implemented in the Uav class defined in Uav.js and the files in the control folder. Basically, the class MechanicsJoystick2D and KeyboardControls are used to get the control input from the player. These user inputs will be translated into the control commands, and the Uav class will update the drone's states and behaviors based on the input. Meanwhile, the JoystickDisplayer class is used to display the real-time joystick control input on the screen.

As for the detailed implementation, to get a better and more realistic motion control effect, the throttle control is acted by applying a force to the drone using the applyLocalForce method in the Cannon-es.js physics engine.

Actually, the Uav class is generally responsible for most of the drone logic, including the initial- ization, update, audio control, shooting control, and so on.

2.3 Bullet Strategy

Two classes, ChaseStrategy and BulletRegular, are responsible for the bullet control.

ChaseStrategy is used to control the bullet's tracking behavior. It searched for the nearest target in the scene and obtain the direction for the bullet to chase. And the BulletRegular class will update the bullet's velocity based on the chase strategy.

Beyond tracing, the BulletRegular class also handles the launch, collision, explosion events, and other bullet-related logic.

2.4 Enemy Strategy

The autonomous enemy drone entity is implemented by the EnemyDrone class. It is designed with various functionalities, including movement, health management, collision handling, and sound effects.

Movement logic is a key part of the enemy drone's behavior. It is based upon three states: idle, chasing, and goingHome. When the enemy drone is idle, remains near its home position and roams within a predefined radius. When a target is detected within its view distance, the drone actively chases it. If the drone strays too far from its home position, it returns to its base. When the drone is destroyed, it take on a self-destruction visual effect through the temperature parameter.

2.5 UI Design

Instead of using the traditional HTML/CSS UI design, this project implements the UI design in the 3D scene using Three.js.

The UI design aims to provide a totally immersive visual and interactive experience, hence all the UI elements are real 3D interactive objects in the scene. Movement of mesh objects, articulated objects with skeletal animation, are used to create the animation and interaction effects of the UI elements. In the former case, the Tween.js library is used to create smooth mesh animations.

Besides, the text are added through texture mapping manipulation on the 3D objects, rather than using the traditional HTML text or canvas text.

2.6 Camera Control

The camera control is mainly implemented in the FollowCamera class, which supports both third- person and first-person view, and the switching between them.

In the third-person view, the method updateThirdView method calculates and updates the camera's position and orientation in third-person view. The drone's position and orientation in the scene are tracked, and quaternion operations are used to slightly offset the camera's orientation, providing a more cinematic angle. It also implements a smooth camera movement effect by using the lerp method.

In the first-person view, the camera's position and orientation are aligned with the drone, and a small pitch angle is added for more natural view.

3 Results

Below are the technical points and the corresponding results that have been implemented in the project.

3.1 Graphics Assets

- Scene layout: The object geometry, texture, and material.

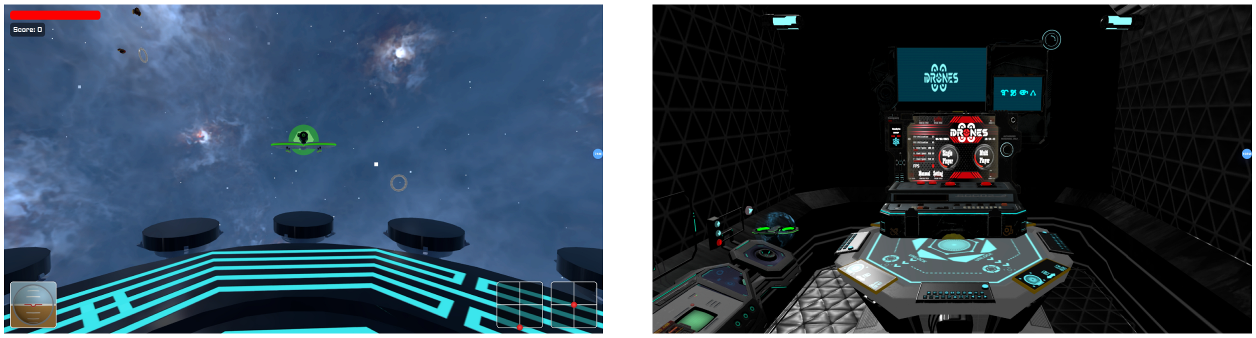


Figure 1: Scene layout in game and UI.

- Environment lighting: The ambient light, directional light, and spot light.

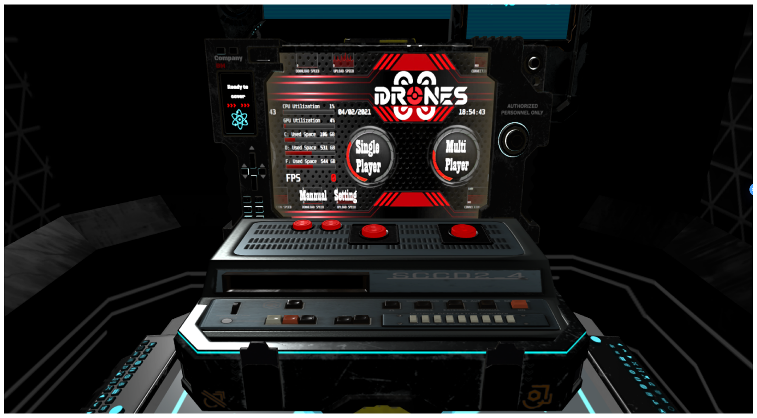


Figure 2: Environment lighting detail.

- Synchronized audio: The background music and sound effects.

3.2 Animation

- Animations.
- Articulated objects: The UI scene contains articulated objects with skeletal animation.
- Fluid simulation: The smoke and fire effects in explosions.

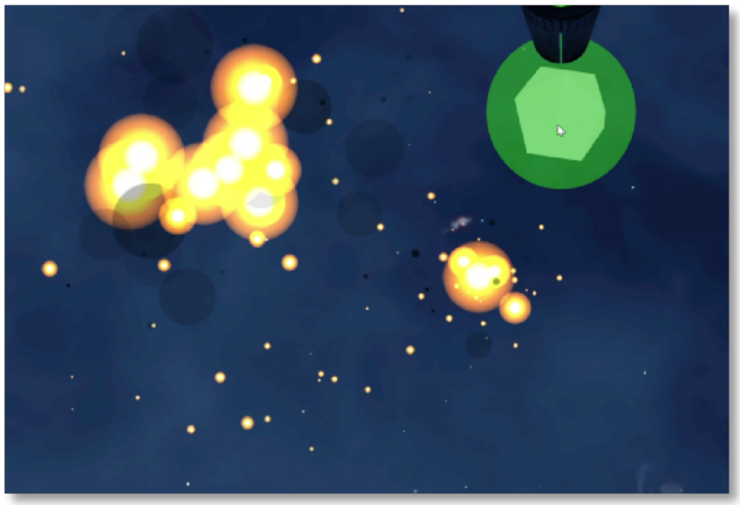


Figure 3: Explosion effect with smoke and fire.

- Collision handling: The collision detection and response between the drone, bullet, and environment.

3.3 Interactive Control

- Control of the main game character: The drone.
- Camera motion control in both third-person and first-person view.

3.4 UI Design

- Proper game start and end interface.
- Additional auxiliary interfaces: The customization interface for the drone's appearance, the manual interface.
- User-friendly layout with visually appealing design.

3.5 Other Advanced Game Features

- Multiplayer mode.

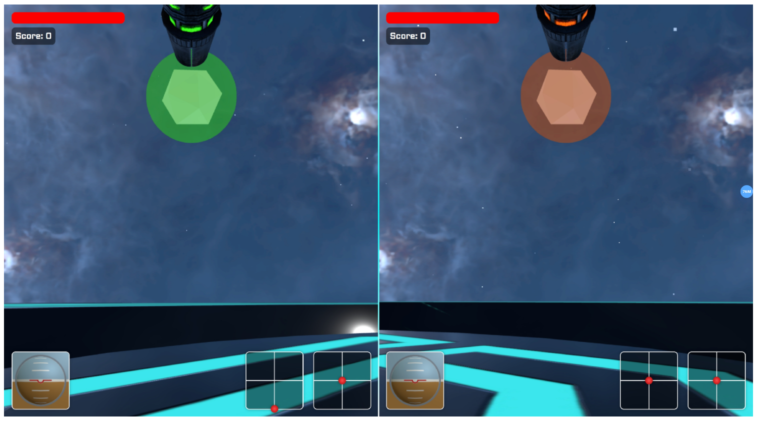


Figure 4: The multi-player mode.

- Customizable character appearance: The drone's appearance.

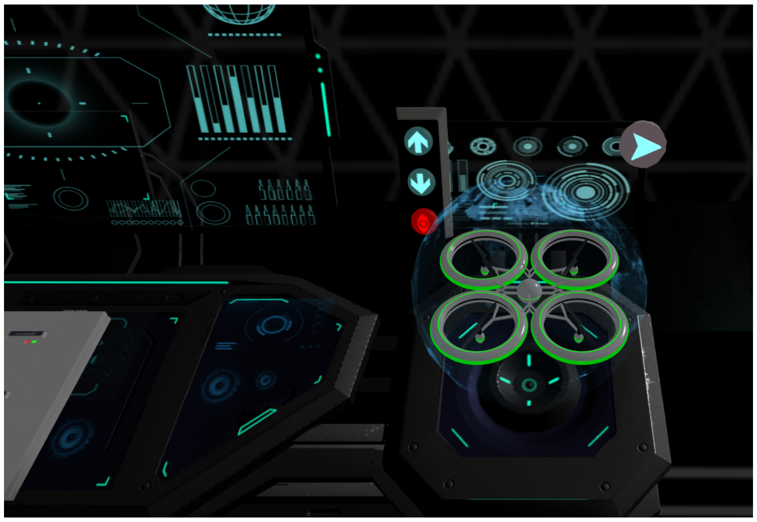


Figure 5: The customization interface.

3.6 Completion

- A fully functional, playable game.

- User mannual or instructions.

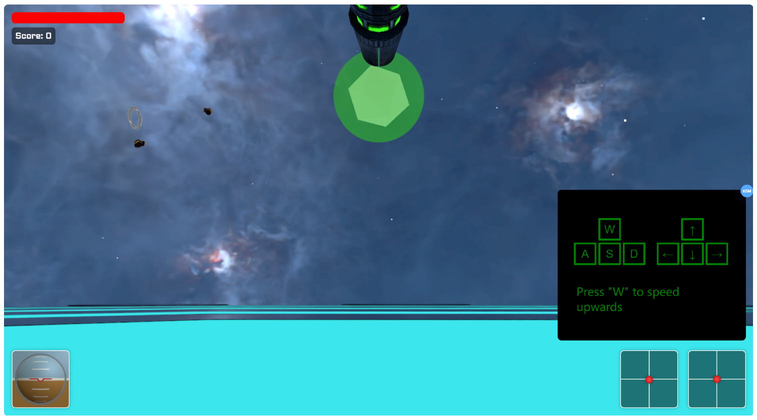


Figure 6: The mannual level.

- Easy online access: The game can be played on the web browser.

4 Discussion

First of all, one of most advanced features of this project is that it uses the keyboard to simulate the joystick control of the drone, using multiple keys for each drone to implement the fully DoF and gives the realistic operation experience. Besides, the project works on implementing highly decoupled and modularized code, which makes the project more maintainable and extensible.

The innovation point lays in the UI design, which is implemented in the 3D scene with fully interactive and realistic 3D objects instead of the traditional 2D UI design, providing a more immersive and visually appealing experience.

However, there are still some limitations in the project. For example, the level design is relatively simple and lacks diversity, and the game lacks a more detailed and comprehensive scoring system.

5 Personal Contribution Statement

My work mainly focuses on the UI interafce, mannual level, and the main game logic. In the UI interface, my works includes building up the 3D UI scene, designing the layout and the appearance of the UI objects, implementing the interactive animation and effects of the UI objects, the transition animations and logics between different UI steps, and adding the sound effects and background music. In the mannual level, I designed the mannual logic and visualizations. In the main game logic, I implement the overall game loop, including the start, mode switch, end, etc.

The extral functionalities beside the basic ones I achieved are below:

- Environment lighting.
- Synchronized audio.
- Articulated objects: Articulated objects with skeletal animation in the UI scene.
- Additional auxiliary interfaces: The manual interface, the customization interface for the drone's appearance.
- User-friendly layout with visually appealing design.

References

- **Parrot Camo drone** by [domiiniic], used under [CC BY 4.0](https://creativecommons.org/licenses/by/4.0/). Available at https://sketchfab.com/3d-models/parrot-camo-drone-351867524b9b478fa406aad31d838ef4.
- **Drone** by [Kai Xiang], used under [CC BY 4.0](https://creativecommons.org/licenses/by/4.0/). Available at https://sketchfab.com/3d-models/drone-fa5e5e3b0f6c4023b8e3d189cf55662f.
- **(FREE) Robot Drone SDC-01** by [SDC PERFORMANCE], used under [CC BY 4.0](https://creativecommons.org/licenses/by/4.0/). Available at https://sketchfab.com/3d-models/free-robot-drone-sdc-01-666c2f8810494952863f9cc8bd273133.
- **Sci-Fi Door** by [thomass3278], used under [CC BY 4.0](http://creativecommons.org/licenses/by/4.0/).
  Available at https://sketchfab.com/3d-models/sci-fi-door-33e67bdeee8b48ba9a617b238565430f.
- **CyberRoom** by [JuliaIce], used under [CC BY 4.0](https://creativecommons.org/licenses/by/4.0/).
  Available at https://sketchfab.com/3d-models/cyberroom-scifi-fb9e951263f84af290bfb96ede790d01.
- **Button_ru** by [imai], used under [CC BY 4.0](https://creativecommons.org/licenses/by/4.0/). Available at https://sketchfab.com/3d-models/button-ru-71c4c4be5f1b4d5d846919e87490b03e.
- **Camera Button** by [miranda.j.rice], used under [CC BY 4.0](https://creativecommons.org/licenses/by/4.0/).
  Available at https://sketchfab.com/3d-models/camera-button-c22068209db048948253a205fc5a211f.
- **Single-Channel Counting Device 'SCCD2_4'** by [NeRD], used under [CC BY 4.0](https://creativecommons.org/licenses/by/4.0/). Available at https://sketchfab.com/3d-models/single-channel-counting-device-sccd2-4-afd69a65d90f4844bfd92edb042
- **Panel (product design)** by [Mrhigh], used under CC BY 4.0.
  Available at: https://sketchfab.com/3d-models/panel-product-design-fbe1c3c2d109457486a50ff41508a512
- **Sci-Fi Panels** by [Xavi Pujadas], used under CC BY 4.0.
  Available at: https://sketchfab.com/3d-models/sci-fi-panels-c1243c5a9e954c539b2c181c886dad62
- **gun satellite panel Computer** by [Mehdi Shahsavan], used under CC BY 4.0.
  Available at: https://sketchfab.com/3d-models/gun-satellite-panel-computer-cc3ca5adf3704c198800dd8dc7
- **Technology aperture out** by [Yuki], used under CC BY 4.0.
  Available at:
  https://sketchfab.com/3d-models/technology-aperture-out-a47045451f1843bfa69105f84b779717
- **Sci Fi Panel Control** by [Vladyslav Holhanov], used under CC BY 4.0.
  Available at: https://sketchfab.com/3d-models/sci-fi-panel-control-fa12fa24d2664932988c2af9cce565a6
- **Space nebula HDRI panorama 360 skydome** by [Aliaksandr.melas], used under CC BY 4.0.
  Available at:
  https://sketchfab.com/3d-models/space-nebula-hdri-panorama-360-skydome-270e7a54eb1e44fcbd5ddb2c1e5
- **Transformers: War For Cybertron Fusion Cannon** by [wickacik], used under CC BY 4.0.
  Available at:
  https://sketchfab.com/3d-models/transformers-war-for-cybertron-fusion-cannon-57ac9f96d4eb4c01ab5ab
  Note: Color modified in the game.

- **Futuristic Roundtable** by [Andulil], used under CC BY 4.0.
  Available at:
  `https://sketchfab.com/3d-models/futuristic-roundtable-252bf93ec1d845288115581c5358be26`
- **Drone** by [ElliotGriffiths], used under CC BY 4.0.
  Available at: `https://sketchfab.com/3d-models/drone-ad0e1d85295847409e961de33c2ee0bf`
- **TRON-ish low-poly drone** by [Bachklamk], used under CC BY 4.0.
  Available at:
  `https://sketchfab.com/3d-models/tron-ish-low-poly-drone-cad1fc9ada864e06ab69a37705656392`