Atelier Découverte de la VR Interactive — Version avancée

Théo AVRIL

21 octobre 2025

Concept

Contexte. A-Frame permet d'écrire de la **3D en HTML**. Une scène est faite d'*entités* (`<a-entity>`) auxquelles on ajoute des *composants* (attributs/comportements). Objectif de la séance : construire une scène **interactive** (clics, animations, assets) et comprendre la logique **ECS**.

1 Objectifs

— Construire une scène VR et comprendre l'**approche entités–composants** d'A-Frame.

— Créer des **composants personnalisés** et gérer les **événements**.

— Utiliser **textures**, **lumières**, **modèles 3D** (GLB/OBJ), **animations**, **curseur/caméra**.

— Appliquer de bonnes pratiques de **performance** et **organisation**.

Concept

ECS (Entity-Component System). En A-Frame, tout est *entité* (`<a-entity>`) qui reçoit des *composants* (attributs) pour lui donner un comportement (`position`, `animation`, composants custom, etc.).

Concept

Prérequis HTTP local et assets. Servez la page en **HTTP local** (VS Code Live Server ou `python -m http.server`) : le chargement d'images/modèles peut échouer en `file://` (CORS). Placez vos fichiers (`.jpg`, `.glb`) dans le **même dossier** que `index.html`.

2 Démarrage rapide

2.1 Page HTML minimale

```
<!DOCTYPE html>
<html>
```

```
  <head>
    <meta charset="utf-8">
    <title>Atelier VR Interactif</title>
    <!-- Version testée : 1.2.0 (ou supérieure) -->
    <script src="https://aframe.io/releases/1.2.0/aframe.min.js"></script>
  </head>
  <body>
    <!-- La scène VR sera définie ici -->
  </body>
</html>
```

2.2 Scène de base + caméra et curseur (collez *à l'intérieur* de votre <body> existant)

```
<a-scene>
  <a-entity position="0 1.6 0">
    <a-camera>
      <!-- Curseur au centre : clic souris / tap -->
      <a-cursor></a-cursor>
    </a-camera>
  </a-entity>

  <!-- Primitives -->
  <a-box id="box1" position="-1 0.5 -3" rotation="0 45 0" color="#4CC3D9"
         animation="property: rotation; to: 0 405 0; loop: true; dur: 4000"
         change-color-on-click></a-box>

  <a-sphere position="0 1.25 -5" radius="1.25" color="#EF2D5E"></a-sphere>
  <a-cylinder position="1 0.75 -3" radius="0.5" height="1.5"
  ↪  color="#FFC65D"></a-cylinder>

  <a-plane position="0 0 -4" rotation="-90 0 0" width="4" height="4"
  ↪  color="#7BC8A4"></a-plane>
  <a-sky color="#ECECEC"></a-sky>

  <!-- Interaction souris Desktop plug-and-play -->
  <a-entity cursor="rayOrigin: mouse"></a-entity>
</a-scene>
```

Checkpoint

Résultat attendu : un cube bleu qui tourne, une sphère, un cylindre, un sol vert, un ciel gris.
Si la page est vide, vérifiez l'URL A-Frame et regardez la console (F12).

2.3 Interactivité : composants personnalisés

```
<script>
  // Composant 1 : couleur aléatoire au clic (toujours 6 chiffres hex)
  AFRAME.registerComponent('change-color-on-click', {
    schema: { },
    init: function () {
      this.el.addEventListener('click', function () {
        const n = Math.floor(Math.random() * 16777215);
        const randomColor = '#' + n.toString(16).padStart(6, '0');
        this.setAttribute('color', randomColor);
      });
    }
  });

  // Composant 2 : déplacement aléatoire au clic (valeurs numériques)
  AFRAME.registerComponent('jump-on-click', {
    schema: { y: {type: 'number', default: 1} },
    init: function () {
      this.el.addEventListener('click', () => {
        const rx = parseFloat((Math.random() * 4 - 2).toFixed(2));
        this.el.setAttribute('position', {
          x: rx, y: this.data.y, z: -3
        });
      });
    }
  });
</script>
```

Concept

Pourquoi un composant ? Un composant rend un comportement **réutilisable** : ajoutez `change-color-on-click` sur plusieurs objets sans recopier de code.

Checkpoint

Résultat attendu : le cube change de couleur au clic, et peut “sauter” si `jump-on-click` est appliqué.
Sinon, vérifiez que le script est bien juste avant `</body>`.

3 Référence A-Frame (sélection utile)

3.1 Formes et objets 3D

```
<a-box> <a-sphere> <a-cylinder> <a-plane> <a-cone> <a-torus>
<a-ring> <a-circle> <a-triangle> <a-sky>
```

3.2 Attributs communs

```
position="x y z"  rotation="x y z"  scale="x y z"
color="#ff0000"   opacity="0.5"     visible="true/false"
metalness="0.5"   roughness="0.5"   src="image.jpg"
```

3.3 Lumières (rappels essentiels)

```
<a-light type="ambient" color="#BBB"></a-light>
<a-light type="directional" position="0 1 1"></a-light>
<a-light type="point" position="0 2 0"></a-light>
<a-light type="spot" position="0 2 0" angle="45"></a-light>
```

3.4 Animations

```
<a-box position="0 1 -3"
       animation="property: rotation; to: 0 360 0; loop: true; dur:
       ↪  2000"></a-box>
<!-- Easing, delay, dir, loop, from/to, etc. -->
```

3.5 Événements et curseur/souris

```
<!-- Active un rayon "souris" pour cliquer sans casque -->
<a-entity cursor="rayOrigin: mouse"></a-entity>

<script>
  // S'assurer que les éléments existent avant d'ajouter des listeners
  window.addEventListener('DOMContentLoaded', () => {
    const el = document.querySelector('#box1');
    if (!el) return;
    el.addEventListener('mouseenter', () => el.setAttribute('scale', '1.1 1.1
    ↪  1.1'));
    el.addEventListener('mouseleave', () => el.setAttribute('scale', '1 1
    ↪  1'));
  });
</script>
```

3.6 Textures et images

```
<a-assets>
  <img id="brick" src="brick.jpg">
</a-assets>
<a-box src="#brick" width="2" height="1" depth="1"></a-box>
```

3.7 Chargement de modèles 3D

```
<a-assets>
  <a-asset-item id="tree" src="modele.glb"></a-asset-item>
</a-assets>
<a-entity gltf-model="#tree" scale="0.5 0.5 0.5" position="0 0
↪  -5"></a-entity>
```

4 Bonnes pratiques et performances

— Limiter les lumières dynamiques et préférer des *materials* simples.

— Réduire le poids des modèles (GLB) et textures (JPEG/PNG optimisés).

— Grouper les entités et nommer via `id` pour les manipulations.

— Tester avec l'**inspecteur** (`Ctrl+Alt+I`) pour ajuster visuellement.

5 Mini-projets guidés

1. Galerie 3D interactive (10–20 min)

But : afficher plusieurs images comme dans une petite expo 3D ; au survol, l'image s'agrandit, au clic elle revient à la taille normale.

Préparation (fichiers) : place au moins `img1.jpg`, `img2.jpg`, `img3.jpg` dans le **même dossier** que `index.html`.

Étape 1 — Déclarer les assets Ajoute des images dans `<a-assets>` (une seule fois par page) :

```
<a-assets>
  <img id="img1" src="img1.jpg">
  <img id="img2" src="img2.jpg">
  <img id="img3" src="img3.jpg">
</a-assets>
```

Checkpoint

OK si : la page se charge sans erreur dans la console (F12). Rien ne s'affiche encore (c'est normal).

Étape 2 — Poser 3 cadres d'images Affiche trois `<a-image>` alignées. Choisis des `position` (x, y, z) cohérentes (z négatif = devant) :

```
<a-image id="p1" src="#img1" position="-1 1.5 -3" width="1.5"
↪  height="1"></a-image>
<a-image id="p2" src="#img2" position=" 0 1.5 -3" width="1.5"
↪  height="1"></a-image>
<a-image id="p3" src="#img3" position=" 1 1.5 -3" width="1.5"
↪  height="1"></a-image>
```

Checkpoint

Résultat attendu : 3 images visibles, côte à côte, à hauteur d'yeux.

Étape 3 — Effet zoom au survol (mouseenter / mouseleave) Ajoute ce squelette JS et **complète** les TODO. L'idée : au survol, `scale` passe à `1.2 1.2 1` puis revient à `1 1 1`.

```
<script>
  window.addEventListener('DOMContentLoaded', () => {
    const frames = document.querySelectorAll('a-image'); // tes 3 images
    frames.forEach((el) => {
      el.addEventListener('mouseenter', () => {
        // TODO: agrandir légèrement l'image (scale)
        // el.setAttribute('scale', '... ... ...');
      });
      el.addEventListener('mouseleave', () => {
        // TODO: revenir à l'échelle normale
        // el.setAttribute('scale', '... ... ...');
      });
    });
  });
</script>
```

Étape 4 — Clic pour réinitialiser (ou afficher une légende) Au clic, reviens à l'échelle normale **ou** affiche une `<a-text>` sous l'image (au choix) :

```
<script>
  // ... suite du code précédent
  // TODO: Dans le forEach, ajoute un listener 'click' qui:
  // - remet scale à '1 1 1'
  // - (option) set une légende via <a-text> existante ou ajoutée avant
</script>
```

Bonus Ajoute un "cadre" : place un `<a-plane>` légèrement plus grand juste derrière chaque image (même position, z un peu plus proche de 0).

Checkpoint

Terminé si : le zoom se fait bien au survol, et le clic réinitialise (ou affiche une légende).

2. Système solaire simplifié (15–25 min)

But : un "soleil" au centre + une planète en orbite (et **bonus** : une lune). L'orbite se fait en animant la **rotation d'un parent**.

Étape 1 — Créer le soleil + lumière Place le soleil et une lumière `point` proche du soleil :

```
<!-- Soleil au centre -->
<a-sphere id="sun" position="0 1.5 -4" radius="0.5"
↪  color="#FDB813"></a-sphere>
<!-- Lumière "solaire" -->
<a-light type="point" position="0 1.5 -4" intensity="1.5"></a-light>
```

Checkpoint

Résultat attendu : une sphère jaune bien éclairée, visible devant toi.

Étape 2 — Orbite planétaire (parent rotateur) Le **truc** : on crée un `<a-entity>` parent au même endroit que le soleil, on **anime sa rotation**, et on place la planète en **décalage** sur l'axe X (rayon d'orbite approx. 1.5).

```
<!-- Parent qui tourne -->
<a-entity id="orbit1" position="0 1.5 -4"
          animation="property: rotation; to: 0 360 0; loop: true; dur: 8000;
          ↪  easing: linear">
  <!-- Planète décalée sur X : rayon d'orbite approx. 1.5 -->
  <a-sphere id="planet1" position="1.5 0 0" radius="0.2"
  ↪  color="#4CC3D9"></a-sphere>
</a-entity>
```

Checkpoint

Résultat attendu : la planète tourne autour du soleil (même si tu ne vois pas le parent).

Étape 3 — Ajouter une lune (orbite locale) Même principe : un parent secondaire centré sur la planète.

```
<!-- Dans orbit1, autour de planet1 -->
<a-entity id="orbitMoon"
          animation="property: rotation; to: 0 360 0; loop: true; dur: 3000;
          ↪  easing: linear">
```

```
  <!-- Lune décalée de planet1 (position relative) -->
  <a-sphere id="moon1" position="0.4 0 0" radius="0.06"
  ↪  color="#BBB"></a-sphere>
</a-entity>
```

Checkpoint

Résultat attendu : la planète orbite le soleil, la lune orbite la planète.

Étape 4 — Ajuster vitesses et distances Modifie `dur` (vitesse d'orbite) et `position.x` (rayon) pour obtenir un rendu lisible. **Bonus :** ellipse approximative en animant aussi `position` de la planète (lentement) entre deux valeurs.

3. Jeu "attrape-la-boîte" (15–25 min)

But : cliquer la boîte pour marquer des points ; à chaque clic, elle "saute" ailleurs. Afficher le score via `<a-text>`.

Étape 1 — Cible + score Place une boîte cible (légère rotation pour la repérer) et un texte de score :

```
<a-box id="target" position="0 1 -3" rotation="0 45 0"
↪  color="#4CC3D9"></a-box>
<a-text id="score" value="Score: 0" position="-0.6 2 -3"
↪  color="#FFFFFF"></a-text>
```

Checkpoint

Résultat attendu : la boîte et le texte "Score : 0" sont visibles.

Étape 2 — Squelette du script Ajoute ce squelette JS, puis **complète** les TODO. Objectif : au clic, *score++*, mise à jour du texte, et **repositionnement** de la boîte dans une zone raisonnable devant toi.

```
<script>
  window.addEventListener('DOMContentLoaded', () => {
    const target = document.querySelector('#target');
    const scoreText = document.querySelector('#score');
    let score = 0;

    function randomInRange(min, max) {
      return parseFloat((Math.random() * (max - min) + min).toFixed(2));
    }

    function moveTarget() {
      // TODO: choisis des bornes raisonnables
      // x: [-2, 2], y: [0.6, 2], z: [-4.5, -3]
```

```
      const newPos = {
        x: randomInRange(-2, 2),
        y: randomInRange(0.6, 2),
        z: randomInRange(-4.5, -3)
      };
      // TODO: applique la nouvelle position
      // target.setAttribute('position', newPos);
    }

    // 1) Au clic: +1 point, mettre à jour le texte, déplacer la cible
    target.addEventListener('click', () => {
      // TODO: incrémenter score
      // score = ...
      // TODO: mettre à jour l'affichage
      // scoreText.setAttribute('value', 'Score: ' + score);
      // TODO: déplacer la cible
      // moveTarget();
    });

    // 2) Option: au survol, légère mise en évidence
    target.addEventListener('mouseenter', () => {
      // TODO: agrandir un peu la boîte (ex: 1.1 1.1 1.1)
    });
    target.addEventListener('mouseleave', () => {
      // TODO: revenir à 1 1 1
    });

    // Démarrage: placer une première fois la cible
    moveTarget();
  });
</script>
```

Checkpoint

Résultat attendu : chaque clic fait +1 et déplace la boîte ; le score s'affiche correctement.

Bonus (au choix)

— **Timer :** ajoute un compte à rebours via `setInterval` ; quand il atteint 0, désactive le clic.

— **Difficulté :** diminue l'intervalle des positions (zone plus large, ou plus éloignée) à chaque point.

— **Effet :** ajoute une animation `animation="property: rotation; ..."` pour faire tourner la boîte plus vite après chaque clic.

6 Dépannage (FAQ rapide)

— **Page vide** : vérifier l'URL A-Frame ; regarder la console (F12).

— **Pas de clic** : le composant est-il chargé *avant* `</body>` ? Activer `cursor="rayOrigin: mouse"`.

— **Modèle non visible** : vérifier chemin `src`, `scale` trop petit, ou erreurs CORS ; servir en HTTP local.

7 Ressources

— Documentation : `https://aframe.io/docs/`

— Exemples : `https://aframe.io/examples/`

— Composants : `https://www.npmjs.com/search?q=aframe-component`

— Modèles 3D : Sketchfab, TurboSquid

— Textures : Textures.com, Freepik

— Éditeur en ligne : `https://glitch.com/~aframe`