Mini Project – ARVR

Conveyor Belt Sorter for Part Manufacturing

Siddharth Huddar 220929030
Sarthak VP 220929322
Aryan Pawar 220929076
Vivek Suryawanshi 220929248

Objective

- Simulate automated sorting using Unity 3D.
- Implement realistic motion via Rigidbody physics.
- Add size and color detection logic.
- Visualize through AR/VR for immersive analysis.
- Demonstrate how Unity can act as an industrial prototyping tool.

Problem Definition

Efficient and accurate sorting of parts by size and color is crucial in manufacturing. Current methods often lead to inefficiencies and increased labor costs. To address these challenges, several key issues must be considered:

- **Sorting Accuracy**: Ensuring that parts are sorted correctly to avoid misplacements.
- **Speed of Operation**: The sorting system must operate at high speeds to keep up with production demands.
- **System Reliability**: The sorting mechanisms should be reliable under varying operational conditions.
- **Flexibility**: Ability to adapt to different part sizes and colors without extensive reconfiguration.

Implementing an automated conveyor belt sorting system using Unity 3D can effectively tackle these challenges. This system will enhance the sorting process and significantly reduce errors, leading to improved operational efficiency and lower costs. By utilizing advanced algorithms and real-time monitoring, the system aims to achieve a seamless flow of operations in part manufacturing environments.

Concept Design

Overview of Conveyor Belt Sorting Design

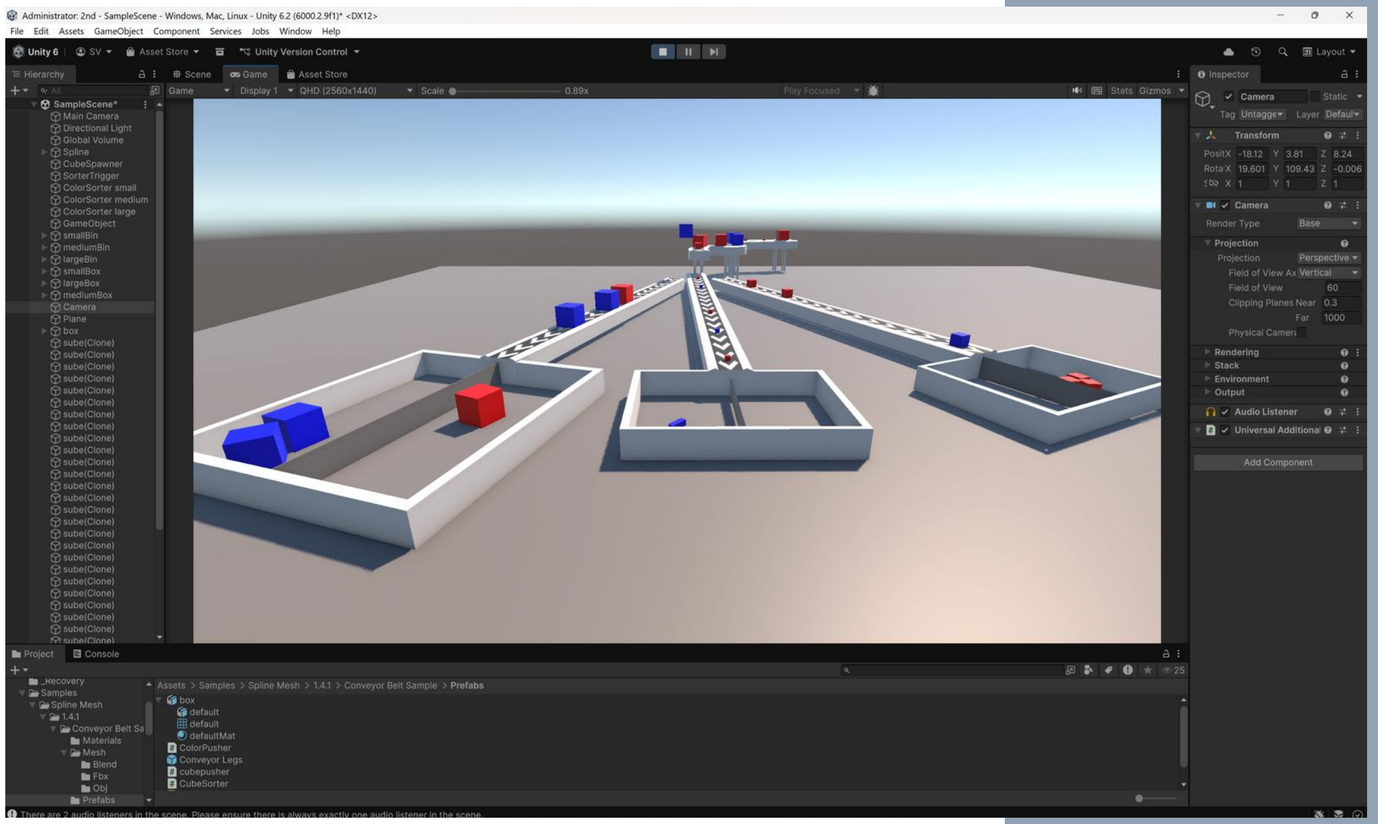
System divided into following stages:

1. **Concept Design**
2. **Environment Setup**
3. **Object Generation**
4. **Sorting Logic**
5. **Object Collection**

Workflow: Main Conveyor → Size Sorter → 3 Conveyors → Color Sorter → Color Bins

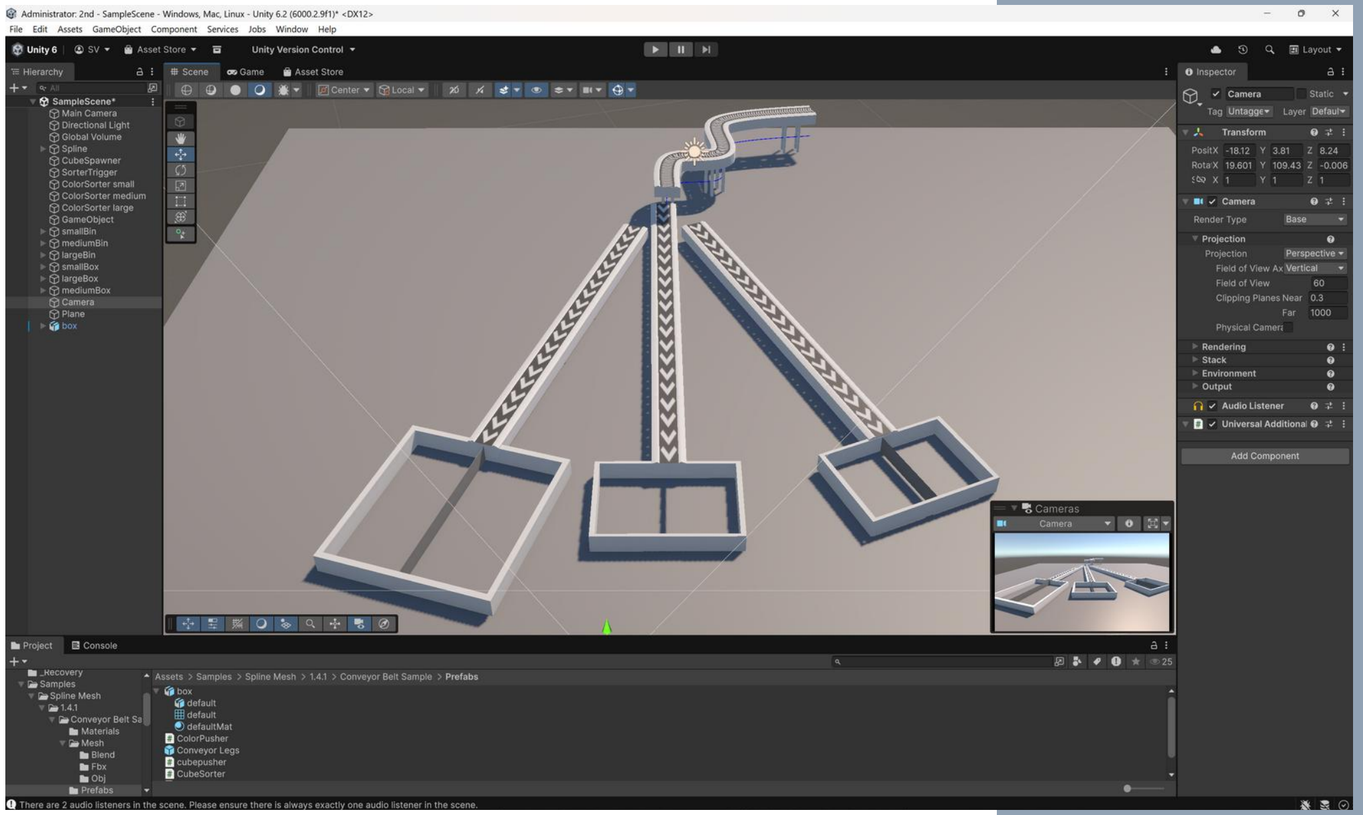
Concept Design

- Designed overall layout and sorting flow.
- Three secondary conveyors handle different sizes.
- Each has its own color sensor and bin.
- Defined pusher forces and trigger zones for realism.
- Planned modular stages to allow future upgrades or sensor additions.



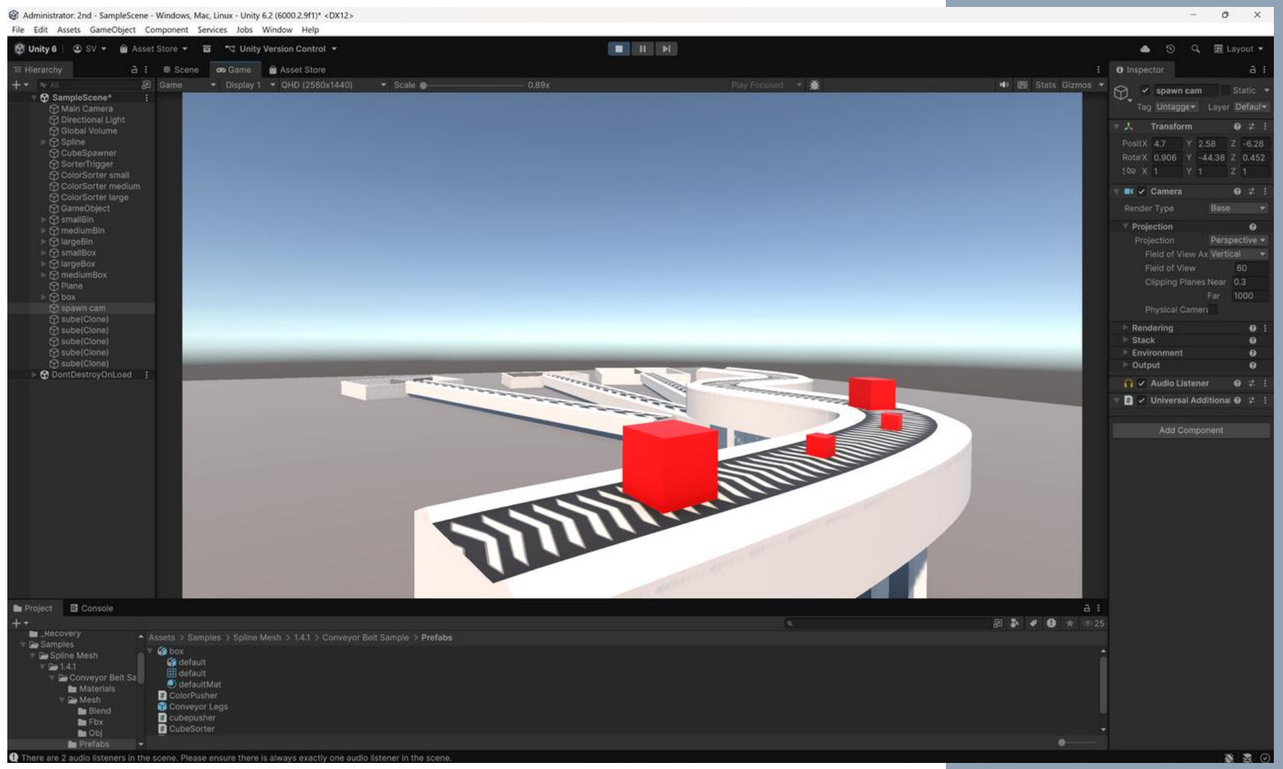
Unity Environment Setup

- Built in Unity 3D with conveyors, bins, sensors, and colliders to mimic an industrial sorting line.
- Integrated Rigidbody and Box Collider for realistic physics and collisions.
- Added materials, lighting, and textures for an industrial look.
- Optimized camera angles and layout for clear visualization of sorting motion.



Conveyor Materials & Textures

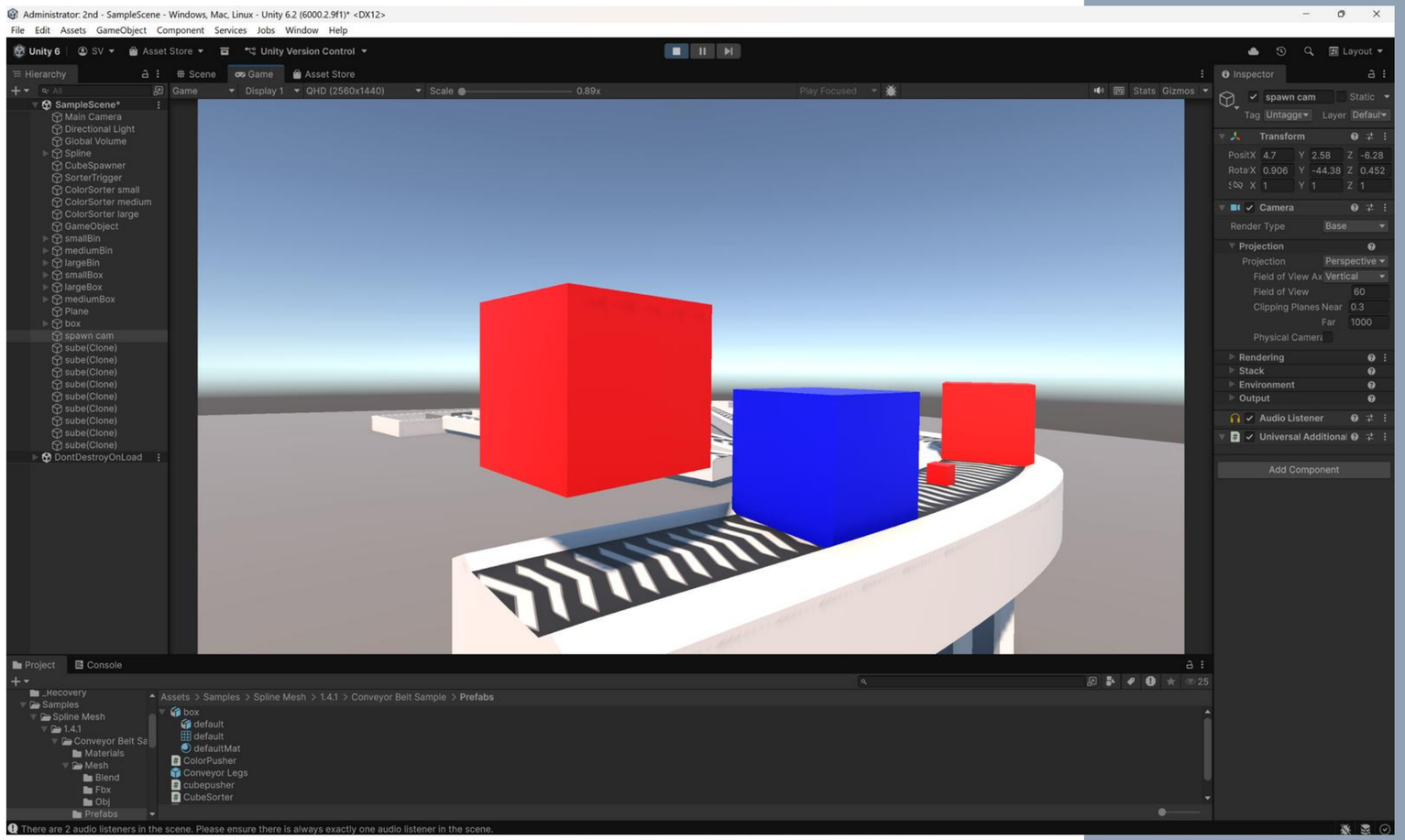
- The conveyor was modeled as a **rubber-textured plane** with adjusted friction parameters to simulate realistic motion.
- **Metallic rollers** were represented using reflective materials to create accurate texture reflections.
- The **floor was given a matte finish** to minimize unwanted light glare within the Unity environment.
- **Friction coefficients were fine-tuned** to prevent object slippage and ensure stable part movement during sorting.



Random Cube Generation

- Cubes generated at regular intervals at the spawn point.
- Randomized color and size simulate production variability.
- Spawn controlled by CubeSpawner.cs script.

Overview of the Spawn Point Mechanism



Spawning Code

```
public GameObject cubePrefab;
public float spawnInterval = 1.5f;

private void Start()
{
    StartCoroutine(SpawnCubes());
}

IEnumerator SpawnCubes()
{
    while (true)
    {
        SpawnCube();
        yield return new WaitForSeconds(spawnInterval);
    }
}
```

This sets up the spawner. A coroutine repeatedly calls SpawnCube() after every interval. It ensures cubes appear automatically during runtime.

Spawning Code

```
private readonly float[] cubeSizes = { 0.1f, 0.25f, 0.5f };

float randomScale = cubeSizes[Random.Range(0, cubeSizes.Length)];
cube.transform.localScale = new Vector3(randomScale, randomScale,
randomScale);
```

Each cube gets a random size chosen from three preset values.
This makes the conveyor system simulate size-based sorting later in the project.

Spawning Code

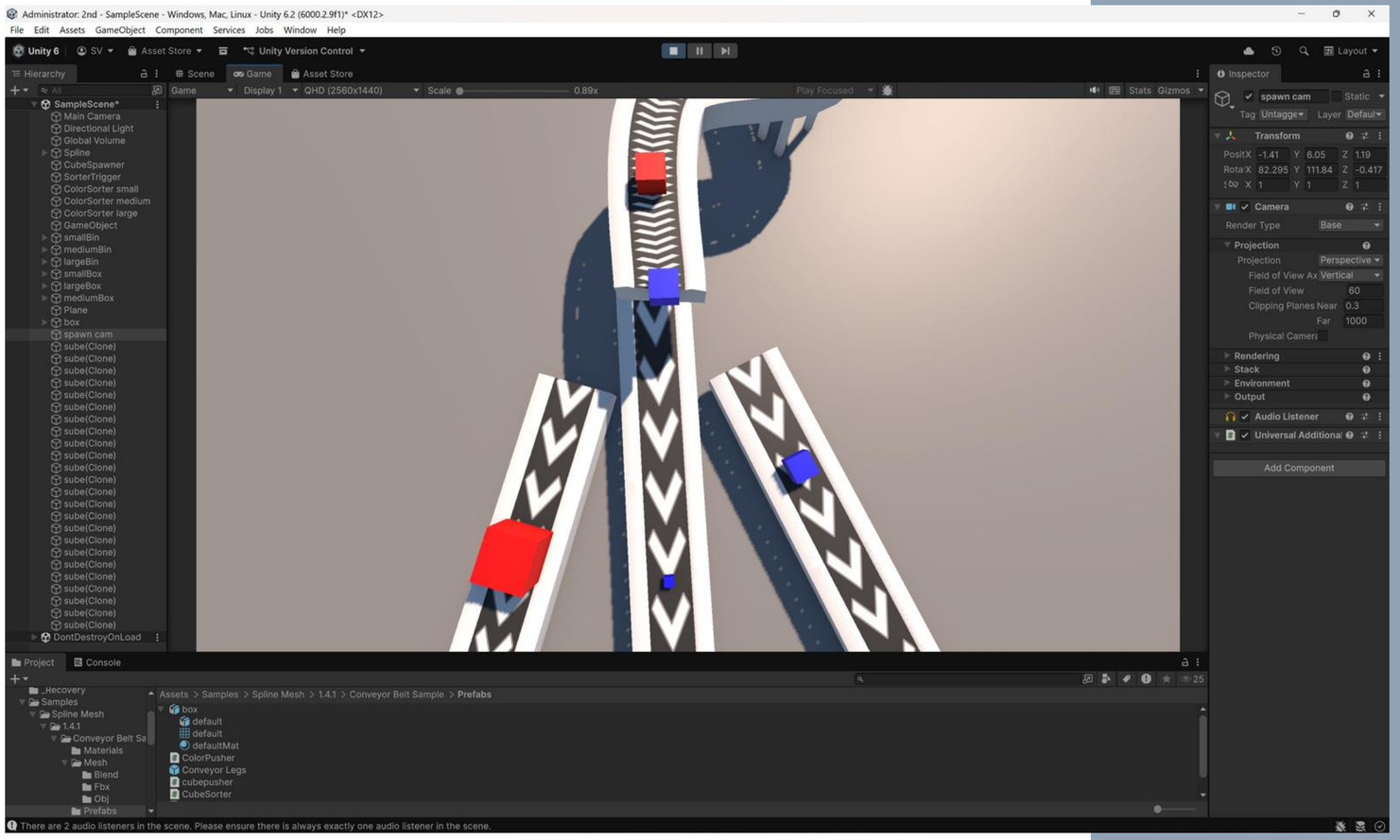
```
private readonly Color[] cubeColors = { Color.red, Color.blue };

Renderer cubeRenderer = cube.GetComponent<Renderer>();
Color randomColor = cubeColors[Random.Range(0, cubeColors.Length)];
cubeRenderer.material.color = randomColor;
```

Colors are randomized between red and blue.
This allows color-based sorting to be tested on the secondary conveyor.

Size Sorting Mechanism

Efficiently segregating parts by attributes

- **Size sorter placed at end of main conveyor.**
- **Detects cube size via trigger collider.**
- **Applies push force toward secondary conveyors.**



Size Sorting Code

```
public float sidePushForce = 5f;
public Vector3 smallPushDirection = new Vector3(-1f, -0.3f, 0f);
public Vector3 mediumPushDirection = new Vector3(0f, -0.3f, 1f);
public Vector3 largePushDirection = new Vector3(1f, -0.3f, 0f);
```

This script controls how cubes are pushed toward different conveyors.
Each direction corresponds to a cube size — small goes left, medium straight, large right.
The slight downward vector ensures the cube drops naturally due to gravity.

Size Sorting Code

```
private void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Block"))
    {
        Rigidbody rb = other.GetComponent<Rigidbody>();
        if (!rb) return;

        float size = other.transform.localScale.x;
```

The trigger detects when a cube enters the sorting zone.
It checks if the object is tagged as a “Block” and retrieves its Rigidbody to apply physics forces. Then it reads the cube’s size to decide which direction to push.

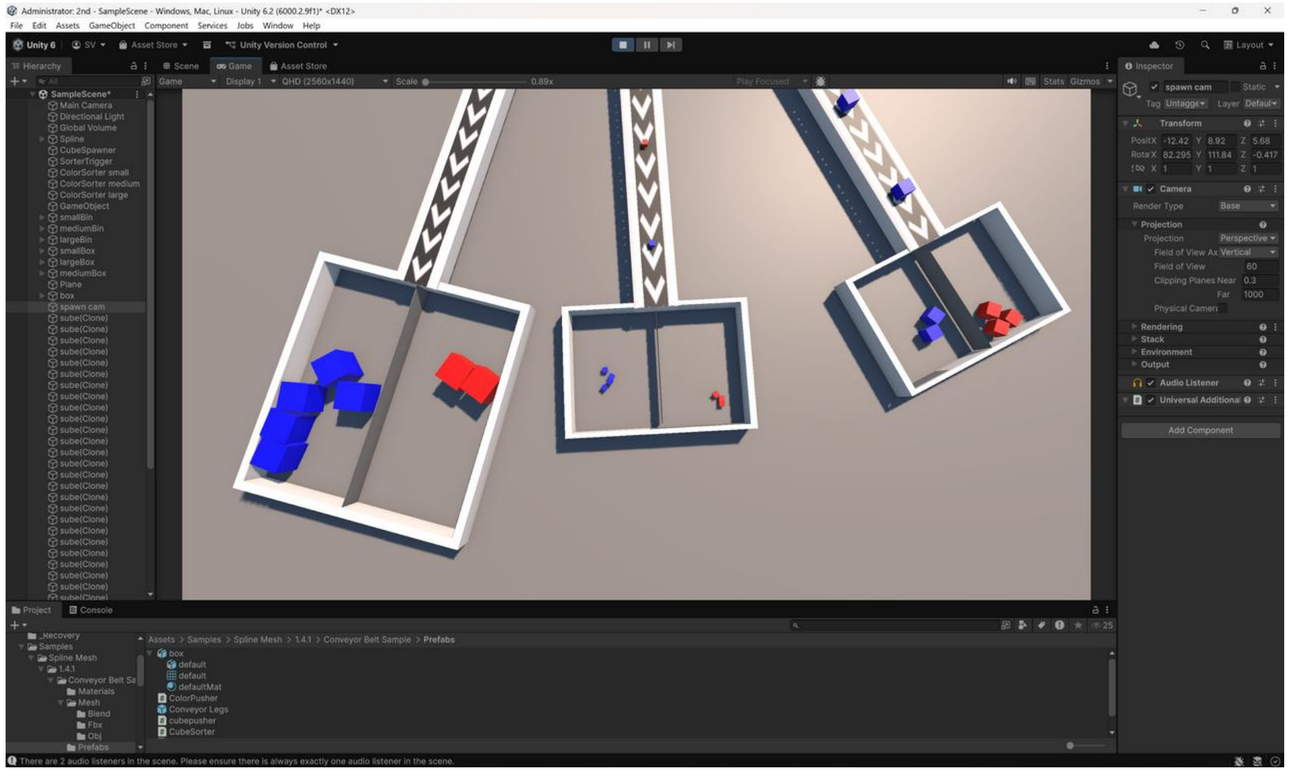
Size Sorting Code

```
if (Mathf.Approximately(size, 0.1f))
    direction = smallPushDirection.normalized;
else if (Mathf.Approximately(size, 0.25f))
    direction = mediumPushDirection.normalized;
else
    direction = largePushDirection.normalized;

rb.AddForce(direction * sidePushForce, ForceMode.Impulse);
```

Based on size, the cube is pushed toward its respective conveyor using an impulse force. This gives a quick, realistic motion instead of teleportation — making the sorting feel physical.

Color Sorting Mechanisms

- Each secondary conveyor has a color detection region.
- Sensor compares material color to target RGB.
- Cube pushed into corresponding bin.



Color Sorting Code

```
[Header("Push Settings")]
public float sidePushForce = 5f;

[Header("Push Directions")]
public Vector3 redPushDirection = new Vector3(-1f, -0.3f, 0f);
public Vector3 bluePushDirection = new Vector3(1f, -0.3f, 0f);
```

This script handles color-based sorting on the secondary conveyor.
Red cubes are pushed to the left, blue cubes to the right.
The downward component in each direction ensures smooth, natural motion with gravity.

Color Sorting Code

```
private void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Block"))
    {
        Renderer rend = other.GetComponent<Renderer>();
        Rigidbody rb = other.GetComponent<Rigidbody>();
        if (!rb || !rend) return;

        Color cubeColor = rend.material.color;
```

The trigger detects when a cube enters the sorting zone.
It fetches the cube's Renderer (for color) and Rigidbody (for physics).
This allows the system to apply realistic forces based on color recognition.

Color Sorting Code

```
if (cubeColor == Color.red)
    direction = redPushDirection.normalized;
else if (cubeColor == Color.blue)
    direction = bluePushDirection.normalized;

rb.AddForce(direction * sidePushForce, ForceMode.Impulse);
```

Depending on the detected color, the cube is pushed in the corresponding direction. The impulse force gives a quick, realistic push — enabling accurate color-based separation on conveyors.

Conclusion

Key findings and project insights

- Successfully implemented a dual-stage sorting system using Unity's built-in physics and trigger mechanisms.
- AR/VR visualization effectively demonstrated real-time manufacturing processes for training and analysis.
- The project offered valuable insights into automation principles, sensor integration, and motion control logic.
- Validated Unity 3D as a reliable virtual prototyping platform for simulating industrial automation systems.
- Overall, the system highlights the potential of digital simulation tools in improving design, efficiency, and learning in modern manufacturing.

Future Developments

Exploring enhancements for conveyor sorting system

- **AI-Based Visual Detection:**

Integrate OpenCV and Unity ML-Agents to enable intelligent vision-based sorting for enhanced adaptability and precision.

- **IoT Integration:**

Implement real-time monitoring and predictive maintenance through IoT connectivity between virtual and physical systems.

- **Digital Twin Linkage:**

Create a bi-directional connection with an actual conveyor system to form a complete digital twin for real-world validation.

- **Enhanced VR Interaction:**

Add hand-tracking and gesture control for immersive, user-driven operation and maintenance training.