**CNC Machine Final code**

**XZ Movement:**

using UnityEngine;  
using Firebase;  
using Firebase.Database;  
using Firebase.Extensions;  
using System;  
  
public class **MoveObjectXZWithFirebase** : MonoBehaviour  
{  
 private DatabaseReference databaseReference;  
 private Vector3 initialPosition;  
 private Vector3 targetPosition;  
 private DateTime lastUpdateTime = DateTime.MinValue; *// Initialize with a default value* public GameObject **objectToMove**; *// Assign the bed object in the Unity Editor* public float **smoothTime** = 0.1f; *// Adjust the time it takes to reach the target position* void **Start**()  
 {  
 *// Store the initial position of the object* initialPosition = new Vector3(-1.66f, 0f, -1.98f);  
 targetPosition = initialPosition;  
  
 *// Initialize Firebase* FirebaseApp.CheckAndFixDependenciesAsync().ContinueWithOnMainThread(task => {  
 FirebaseApp app = FirebaseApp.DefaultInstance;  
 databaseReference = FirebaseDatabase.DefaultInstance.RootReference;  
 databaseReference.Child("status").ValueChanged += HandleValueChanged;  
 });  
 }  
  
 void **Update**()  
 {  
 if (databaseReference != null)  
 {  
 *// Fetch the x\_pos and z\_pos values from Firebase under the "status" node* databaseReference.Child("status").GetValueAsync().ContinueWithOnMainThread(task =>  
 {  
 if (task.IsCompleted && !task.IsFaulted)  
 {  
 DataSnapshot snapshot = task.Result;  
  
 if (snapshot.Exists && snapshot.Child("x\_pos").Exists && snapshot.Child("z\_pos").Exists)  
 {  
 *// Retrieve data from Firebase for x\_pos and z\_pos as floats* float xPosValue = ((float.Parse(snapshot.Child("x\_pos").Value.ToString()) - 12f)/2f) + initialPosition.x; *// Adjust and halve x\_pos* float zPosValue = ((float.Parse(snapshot.Child("z\_pos").Value.ToString()) - 11f)/2f) + initialPosition.z; *// Adjust and halve z\_pos  
  
 // Log the retrieved data* Debug.Log($"Retrieved data from Firebase: X = {xPosValue}, Z = {zPosValue}");  
  
 *// Update the target position* targetPosition = new Vector3(xPosValue, initialPosition.y, zPosValue);  
 }  
 }  
 });  
 }  
  
 *// Smoothly move the object towards the target position* objectToMove.transform.position = Vector3.Lerp(objectToMove.transform.position, targetPosition, smoothTime \* Time.deltaTime);  
 }  
  
 void HandleValueChanged(object sender, ValueChangedEventArgs args)  
 {  
 *// Handle additional logic for real-time updates if needed* }  
}

Y Movement:

using UnityEngine;  
using Firebase;  
using Firebase.Database;  
using Firebase.Extensions;  
using System;  
  
public class **ymovement** : MonoBehaviour  
{  
 private DatabaseReference databaseReference;  
 private Vector3[] initialPositions;  
 private Vector3[] targetPositions;  
 private DateTime lastUpdateTime = DateTime.MinValue;  
 private const float distanceThreshold = 10f;  
  
 public GameObject[] **objectsToMove**;  
 public float **smoothTime** = 0.1f; *// Adjust the time it takes to reach the target position* void **Start**()  
 {  
 initialPositions = new Vector3[objectsToMove.Length];  
 targetPositions = new Vector3[objectsToMove.Length];  
  
 for (int i = 0; i < objectsToMove.Length; i++)  
 {  
 initialPositions[i] = objectsToMove[i].transform.position;  
 targetPositions[i] = initialPositions[i];  
 }  
  
 FirebaseApp.CheckAndFixDependenciesAsync().ContinueWithOnMainThread(task => {  
 FirebaseApp app = FirebaseApp.DefaultInstance;  
 databaseReference = FirebaseDatabase.DefaultInstance.RootReference;  
 databaseReference.Child("status").ValueChanged += HandleValueChanged;  
 });  
 }  
  
 void **Update**()  
 {  
 if (databaseReference != null)  
 {  
 databaseReference.Child("status").GetValueAsync().ContinueWithOnMainThread(task =>  
 {  
 if (task.IsCompleted && !task.IsFaulted)  
 {  
 DataSnapshot snapshot = task.Result;  
  
 if (snapshot.Exists && snapshot.Child("y\_pos").Exists)  
 {  
 float yPosValue = float.Parse(snapshot.Child("y\_pos").Value.ToString());  
  
 Debug.Log("Retrieved yPosValue: " + yPosValue);  
  
 TimeSpan timeDifferenceY = DateTime.Now - lastUpdateTime;  
  
 if (timeDifferenceY.TotalSeconds > 1.0)  
 {  
 yPosValue -= 10f; *// Adjust the value as needed* float distanceToMove = Mathf.Abs(yPosValue);  
  
 for (int i = 0; i < objectsToMove.Length; i++)  
 {  
 targetPositions[i] = initialPositions[i];  
 targetPositions[i].y = initialPositions[i].y - distanceToMove;  
 }  
  
 lastUpdateTime = DateTime.Now;  
 }  
 }  
 }  
 });  
 }  
  
 *// Smoothly move each object towards its target position* for (int i = 0; i < objectsToMove.Length; i++)  
 {  
 objectsToMove[i].transform.position = Vector3.Lerp(objectsToMove[i].transform.position, targetPositions[i], smoothTime \* Time.deltaTime);  
 }  
 }  
  
 void HandleValueChanged(object sender, ValueChangedEventArgs args)  
 {  
 *// Handle additional logic for real-time updates if needed* }  
}

**Material Removal: (main)**

using UnityEngine;  
using Firebase;  
using Firebase.Database;  
using Firebase.Extensions;  
  
public class **CubeGenerator** : MonoBehaviour  
{  
 public GameObject **cubePrefab**; *// Reference to the Cube Prefab* public int **gridSizeX** = 3; *// Number of cubes along the X-axis* public int **gridSizeY** = 3; *// Number of cubes along the Y-axis* public int **gridSizeZ** = 3; *// Number of cubes along the Z-axis* public float **mainCubeSizeX** = 3f; *// Desired size of the entire structure along the X-axis* public float **mainCubeSizeY** = 3f; *// Desired size of the entire structure along the Y-axis* public float **mainCubeSizeZ** = 3f; *// Desired size of the entire structure along the Z-axis* private GameObject mainCube; *// Reference to the main cube* private int destroyedCubes = 0; *// Counter for destroyed cubes* private const int **totalCubes** = 30; *// Predefined fixed total number of cubes (independent of grid size)* private DatabaseReference databaseReference;  
  
 void **Start**()  
 {  
 FirebaseApp.CheckAndFixDependenciesAsync().ContinueWithOnMainThread(task =>  
 {  
 if (task.Result == DependencyStatus.**Available**)  
 {  
 FirebaseApp app = FirebaseApp.DefaultInstance;  
 databaseReference = FirebaseDatabase.DefaultInstance.RootReference;  
 Debug.Log("Firebase initialized.");  
 }  
 else  
 {  
 Debug.LogError("Could not resolve all Firebase dependencies: " + task.Result);  
 }  
 });  
  
 mainCube = gameObject; *// Assuming this script is attached to the main cube* GenerateCubes();  
 }  
  
 void GenerateCubes()  
 {  
 float cubeSizeX = mainCubeSizeX / gridSizeX; *// Size of each smaller cube along the X-axis* float cubeSizeY = mainCubeSizeY / gridSizeY; *// Size of each smaller cube along the Y-axis* float cubeSizeZ = mainCubeSizeZ / gridSizeZ; *// Size of each smaller cube along the Z-axis  
  
 // Calculate the starting position relative to the main cube's position* Vector3 startPosition = new Vector3(-mainCubeSizeX / 2f + cubeSizeX / 2f,  
 -mainCubeSizeY / 2f + cubeSizeY / 2f,  
 -mainCubeSizeZ / 2f + cubeSizeZ / 2f);  
  
 *// Generate cubes based on the grid size (actual cube creation)* for (int x = 0; x < gridSizeX; x++)  
 {  
 for (int y = 0; y < gridSizeY; y++)  
 {  
 for (int z = 0; z < gridSizeZ; z++)  
 {  
 *// Calculate the position of the new cube relative to the main cube* Vector3 position = startPosition + new Vector3(x \* cubeSizeX, y \* cubeSizeY, z \* cubeSizeZ);  
  
 *// Instantiate a new cube at the calculated position* GameObject newCube = Instantiate(cubePrefab, mainCube.transform.position + position, Quaternion.identity);  
  
 *// Set the scale of the new cube* newCube.transform.localScale = new Vector3(cubeSizeX, cubeSizeY, cubeSizeZ);  
  
 *// Make the new cube a child of the main cube* newCube.transform.parent = mainCube.transform;  
  
 *// Add a tag to the new cube* newCube.tag = "Smallcube";  
 }  
 }  
 }  
  
 *// Optionally, disable the main cube's renderer if you don't want it visible* Renderer mainCubeRenderer = mainCube.GetComponent<Renderer>();  
 if (mainCubeRenderer != null)  
 {  
 mainCubeRenderer.enabled = false;  
 }  
 }  
  
 public void CubeDestroyed()  
 {  
 destroyedCubes++;  
 LogCompletionPercentage();  
 }  
  
 void LogCompletionPercentage()  
 {  
 *// Ensure destroyedCubes never exceeds totalCubes* int clampedDestroyedCubes = Mathf.Clamp(destroyedCubes, 0, **totalCubes**);  
  
 *// Calculate the percentage based on the clamped destroyed cubes* float percentage = ((float)clampedDestroyedCubes / **totalCubes**) \* 100f;  
  
 *// Log the completion percentage* Debug.Log($"Destroyed Cubes: {clampedDestroyedCubes} ({percentage:F2}% complete)");  
  
 *// Send the percentage to Firebase under the 'status' node* SendCompletionToFirebase(percentage);  
 }  
  
 void SendCompletionToFirebase(float percentage)  
 {  
 if (databaseReference != null)  
 {  
 string roundedPercentage = percentage.ToString("F2");  
 *// Change this line to store the percentage under "status/percentage"* databaseReference.Child("status").Child("percentage").SetValueAsync(percentage).ContinueWithOnMainThread(task =>  
 {  
 if (task.IsCompleted)  
 {  
 Debug.Log("Percentage sent to Firebase successfully.");  
 }  
 else  
 {  
 Debug.LogError("Failed to send percentage to Firebase: " + task.Exception);  
 }  
 });  
 }  
 }  
  
}

**(Sub code):**

using UnityEngine;  
  
public class **DestroyOnCollision** : MonoBehaviour  
{  
 public float **destructionProbability** = 0.5f; *// Probability of a cube being destroyed (between 0 and 1)* private CubeGenerator cubeGenerator;  
  
 void **Start**()  
 {  
 *// Find the CubeGenerator in the scene* cubeGenerator = FindObjectOfType<CubeGenerator>();  
 }  
  
 private void **OnTriggerEnter**(Collider other)  
 {  
 if (other.gameObject.CompareTag("Smallcube"))  
 {  
 if (Random.value < destructionProbability)  
 {  
 *// Destroy the small cube* Destroy(other.gameObject);  
  
 *// Update the count of destroyed cubes in the CubeGenerator* if (cubeGenerator != null)  
 {  
 cubeGenerator.CubeDestroyed();  
 }  
 }  
 }  
 }  
}

**Percentage of completion:**

using UnityEngine;  
using Firebase;  
using Firebase.Database;  
using Firebase.Extensions;  
using System.Collections.Generic;  
  
public class **CubeCuttingProgress** : MonoBehaviour  
{  
 *// Reference to Firebase database* private DatabaseReference databaseReference;  
  
 *// Total cubes to track in the layer* private const int **TotalCubesInLayer** = 30;  
  
 *// Count of cubes cut* private int cubesCut;  
  
 void **Start**()  
 {  
 *// Initialize Firebase* FirebaseApp.CheckAndFixDependenciesAsync().ContinueWithOnMainThread(task => {  
 FirebaseApp app = FirebaseApp.DefaultInstance;  
 databaseReference = FirebaseDatabase.DefaultInstance.RootReference;  
 });  
 }  
  
 *// Call this method when a cube is cut successfully* public void OnCubeCut()  
 {  
 cubesCut++;  
 UpdateCutPercentage();  
 }  
  
 *// Calculate percentage and send data to Firebase* private void UpdateCutPercentage()  
 {  
 float percentageCompleted = (float)cubesCut / **TotalCubesInLayer** \* 100;  
 Debug.Log($"Percentage Completed: {percentageCompleted}%");  
  
 *// Send percentage to Firebase* SendPercentageToFirebase(percentageCompleted);  
 }  
  
 *// Function to send data to Firebase  
 /\* private void SendPercentageToFirebase(float percentage)  
 {  
 // Assuming we are storing the data under a node named "CuttingProgress"  
 databaseReference.Child("status").SetValueAsync(percentage).ContinueWithOnMainThread(task => {  
 if (task.IsFaulted)  
 {  
 Debug.LogError("Failed to send data to Firebase: " + task.Exception);  
 }  
 else  
 {  
 Debug.Log("Successfully sent data to Firebase.");  
 }  
 });  
 }\*/* private void SendPercentageToFirebase(float completePercentage)  
 {  
 *// Assuming we are storing the data under a node named "CuttingProgress"* Dictionary<string, object> updateData = new Dictionary<string, object>();  
 updateData["status"] = completePercentage;  
  
 databaseReference.UpdateChildrenAsync(updateData).ContinueWithOnMainThread(task => {  
 if (task.IsFaulted)  
 {  
 Debug.LogError("Failed to send data to Firebase: " + task.Exception);  
 }  
 else  
 {  
 Debug.Log("Successfully sent data to Firebase.");  
 }  
 });  
 }  
  
}