Complete Convai Setup and Unity Integration Tutorial

What is Convai?

Convai is a conversational AI service for games, metaverse, XR and more, designed to bring your characters to life. It's the highest-rated developer platform for crafting embodied conversational AI characters with advanced multimodal perception abilities. With Convai, you can create NPCs that can have natural conversations, understand voice commands, and take actions in your virtual world.

Table of Contents

Getting Started with Convai

Prerequisites

Before we begin, make sure you have:

- Unity 2021.3 or higher installed
- A computer with a working microphone
- Internet connection
- Basic knowledge of Unity interface

Step 1: Create a Convai Account

1. Visit convai.com
2. Click "Sign Up" and create your free account
3. Verify your email address
4. Once logged in, you'll see the Convai Dashboard

Step 2: Get Your API Key

From the dashboard, grab your API key. This key will be used to connect your Unity project to Convai's services.

1. In your Convai Dashboard, locate the API Key section
2. Copy your unique API key (keep this safe!)
3. You'll need this key later in Unity

Basic Setup

Understanding Convai Components

Convai includes several key features:

- **Speech Recognition**: Converts player speech to text
- **Natural Language Understanding**: Understands what players mean
- **Text Generation**: Creates appropriate responses
- **Text-to-Speech**: Converts AI responses to natural speech
- **Lip Sync**: Matches character mouth movements to speech
- **Actions**: Allows characters to perform tasks in the game world

System Requirements

- **Unity Version**: 2021.3.0f1 or higher (recommended: 2022.3+)
- **Platform Support**: Windows, macOS, WebGL
- **Memory**: Minimum 8GB RAM recommended
- **Storage**: At least 2GB free space for Unity project

Creating Your First AI Character

Step 1: Design Your Character in Convai Playground

1. **Log into Convai Playground**

    - Go to your Convai Dashboard
    - Click on "Character Creator" or "Create New Character"

2. **Set Basic Information**

Character Name: Assistant Amy

Character Description: A helpful virtual assistant

Voice: Choose from available voice options

Language: English (or your preferred language)

3. **Define Personality and Backstory**

   Personality: Friendly, helpful, and knowledgeable

   Backstory: Amy is a virtual assistant who loves helping users

   navigate and learn about virtual environments.

   Knowledge Base: Add specific information your character should know

4. **Configure Actions** (Optional)

   - Define what your character can do (wave, point, dance, etc.)
   - Set up custom commands they can respond to

5. **Save and Copy Character ID**

   - After creating your character, copy the Character ID
   - This unique ID connects your Unity character to the AI brain

Example Character Configuration

```
{
  "name": "Assistant Amy",
  "description": "A helpful virtual guide",
  "personality": "friendly and knowledgeable",
  "voice": "female_01",
  "actions": ["wave", "point", "explain"],
  "knowledge": "Expert in virtual environments and Unity"
}
```

Unity Integration

Step 1: Download and Install Convai Unity Plugin

Download the Convai Unity SDK from Unity Asset Store. This is the Core version of the plugin that has a sample scene for anyone to get started.

Method 1: Unity Asset Store

1. Open Unity Asset Store in your browser
2. Search for "Convai"
3. Find "NPC AI Engine - Dialog, actions, voice and lipsync - Convai"
4. Click "Add to My Assets"
5. Open Unity Hub and create a new 3D project

Method 2: Direct Download

1. Visit docs.convai.com
2. Navigate to Unity Plugin downloads
3. Download the .unitypackage file

Step 2: Import Convai Plugin into Unity

When importing the package, a prompt for using the new Unity Input System will appear. Press Yes. The project will automatically restart.

1. **Import Package**

   Assets > Import Package > Custom Package

   Select: ConvaiforUnity_vX.Y.Z.unitypackage

   Click: Import

2. **Handle Input System Prompt**

   - When prompted about Input System, click "Yes"
   - Unity will restart automatically

3. **Verify Installation**

- Check for "Convai" folder in your Project window
- Look for Convai menu in the top menu bar

Step 3: Configure Convai Plugin

In the Menu Bar, go to Convai > API Key Setup. Enter the API Key and click begin. This will create an APIKey asset in the resources folder.

1. **Set Up API Key**

   Menu: Convai > API Key Setup

   Enter: Your API key from Convai Dashboard

   Click: Begin

2. **Verify Setup**

   - Check that APIKey asset appears in Resources folder
   - No error messages in Console

Step 4: Add Your First AI Character

Open the demo scene by going to Convai > Demo > Scenes > Full Features. Click the Convai NPC Amelia and add the Character ID.

1. **Load Demo Scene**

   Menu: Convai > Demo > Scenes > Full Features

2. **Configure Character**

   - Select the Convai NPC in the scene
   - In Inspector, find "Character ID" field
   - Paste your character ID from Convai Playground

3. **Test Your Character**

   - Press Play
   - Approach the character in the scene
   - Speak into your microphone
   - The character should respond!

Code Examples

Basic Character Controller Script

```
using UnityEngine;

using Convai.Scripts.Runtime.Core;

public class MyConvaiCharacter : MonoBehaviour

{

    [Header("Convai Settings")]

    public string characterID = "your-character-id-here";

    public ConvaiNPC convaiNPC;


    void Start()

    {

        // Get Convai NPC component

        convaiNPC = GetComponent<ConvaiNPC>();


        if (convaiNPC != null)

        {

            // Set character ID

            convaiNPC.characterID = characterID;


            // Subscribe to events

            convaiNPC.OnCharacterResponse += HandleCharacterResponse;
```

```
        convaiNPC.OnActionReceived += HandleActionReceived;
    }
}

void HandleCharacterResponse(string response)
{
    Debug.Log($"Character said: {response}");
    // Add your custom logic here
}

void HandleActionReceived(string action)
{
    Debug.Log($"Character received action: {action}");
    // Handle specific actions like "wave", "dance", etc.

    switch(action.ToLower())
    {
        case "wave":
            PlayWaveAnimation();
            break;
        case "dance":
            PlayDanceAnimation();
```

```
            break;
        }
    }

    void PlayWaveAnimation()
    {
        // Your animation code here
        Debug.Log("Playing wave animation");
    }

    void PlayDanceAnimation()
    {
        // Your animation code here
        Debug.Log("Playing dance animation");
    }
}
```

Advanced Character Interaction Script

```
using UnityEngine;
using Convai.Scripts.Runtime.Core;
using TMPro;
public class AdvancedCharacterController : MonoBehaviour
{
    [Header("UI References")]
```

```
public TextMeshProUGUI responseText;
public TextMeshProUGUI userInputText;

[Header("Character Settings")]
public float responseDelay = 0.5f;
public bool enableProximityChat = true;
public float chatDistance = 5f;

private ConvaiNPC convaiNPC;
private Transform player;
private bool isInChatRange = false;

void Start()
{
    convaiNPC = GetComponent<ConvaiNPC>();
    player = GameObject.FindGameObjectWithTag("Player")?.transform;

    if (convaiNPC != null)
    {
        convaiNPC.OnCharacterResponse += DisplayResponse;
        convaiNPC.OnUserQuery += DisplayUserInput;
    }
```

```
}

void Update()
{
    if (enableProximityChat && player != null)
    {
        CheckProximity();
    }
}

void CheckProximity()
{
    float distance = Vector3.Distance(transform.position, player.position);
    bool wasInRange = isInChatRange;
    isInChatRange = distance <= chatDistance;

    // Enable/disable voice detection based on proximity
    if (isInChatRange != wasInRange)
    {
        convaiNPC.enabled = isInChatRange;

        if (isInChatRange)
```

```
        {
            ShowUIPrompt("You can now talk to " + convaiNPC.characterName);
        }
        else
        {
            HideUIPrompt();
        }
    }
}

void DisplayResponse(string response)
{
    if (responseText != null)
    {
        StartCoroutine(TypewriterEffect(response));
    }
}

void DisplayUserInput(string userInput)
{
    if (userInputText != null)
    {
```

```
        userInputText.text = "You: " + userInput;
    }
}

System.Collections.IEnumerator TypewriterEffect(string text)
{
    responseText.text = "";

    foreach (char letter in text)
    {
        responseText.text += letter;
        yield return new WaitForSeconds(0.02f);
    }
}

void ShowUIPrompt(string message)
{
    // Show proximity prompt UI
    Debug.Log(message);
}

void HideUIPrompt()
```

```
    {
        // Hide proximity prompt UI
    }
}
```

Custom Action Handler

```
using UnityEngine;
using Convai.Scripts.Runtime.Core;
public class CustomActionHandler : MonoBehaviour
{
    [Header("Action Objects")]
    public GameObject[] interactableObjects;
    public Animator characterAnimator;

    private ConvaiNPC convaiNPC;

    void Start()
    {
        convaiNPC = GetComponent<ConvaiNPC>();
        convaiNPC.OnActionReceived += HandleCustomAction;
    }

    void HandleCustomAction(string actionData)
```

```
{
    // Parse action data (JSON format from Convai)
    try
    {
        var actionInfo = JsonUtility.FromJson<ActionInfo>(actionData);
        ExecuteAction(actionInfo);
    }
    catch (System.Exception e)
    {
        Debug.LogError($"Failed to parse action: {e.Message}");
    }
}

void ExecuteAction(ActionInfo action)
{
    switch (action.actionType)
    {
        case "pickup":
            PickupObject(action.targetObject);
            break;

        case "throw":
```

```
            ThrowObject(action.targetObject, action.direction);
            break;

        case "move":
            MoveToLocation(action.targetPosition);
            break;

        case "emote":
            PlayEmote(action.emoteName);
            break;
    }
}

void PickupObject(string objectName)
{
    GameObject target = FindObjectByName(objectName);
    if (target != null)
    {
        // Pickup animation and logic
        characterAnimator.SetTrigger("Pickup");
        target.transform.SetParent(transform);
    }
```

```
}

void ThrowObject(string objectName, Vector3 direction)
{
    GameObject target = FindObjectByName(objectName);
    if (target != null && target.transform.parent == transform)
    {
        characterAnimator.SetTrigger("Throw");

        // Physics throwing
        Rigidbody rb = target.GetComponent<Rigidbody>();
        if (rb != null)
        {
            target.transform.SetParent(null);
            rb.AddForce(direction * 10f, ForceMode.Impulse);
        }
    }
}

void MoveToLocation(Vector3 position)
{
    // Use NavMesh or custom movement
```

```
    UnityEngine.AI.NavMeshAgent agent = GetComponent<UnityEngine.AI.NavMeshAgent>();
    if (agent != null)
    {
        agent.SetDestination(position);
    }
}

void PlayEmote(string emoteName)
{
    characterAnimator.SetTrigger(emoteName);
}

GameObject FindObjectByName(string name)
{
    foreach (GameObject obj in interactableObjects)
    {
        if (obj.name.ToLower().Contains(name.ToLower()))
        {
            return obj;
        }
    }
    return null;
```

```
    }
}
[System.Serializable]
public class ActionInfo
{
    public string actionType;
    public string targetObject;
    public Vector3 direction;
    public Vector3 targetPosition;
    public string emoteName;
}
```

Troubleshooting

Common Issues and Solutions

1. Assembly Validation Errors

Assembly 'Assets/Convai/Plugins/Grpc.Core.Api/lib/net45/Grpc.Core.Api.dll' will not be loaded due to errors. Ensure that Assembly Validation is disabled in Project Settings > Player > Other Settings.

Solution:

1. Go to Edit > Project Settings
2. Navigate to Player > Other Settings
3. Uncheck "Assembly Version Validation"
4. Restart Unity

2. Missing Newtonsoft JSON Errors

Error: `JsonPropertyAttribute could not be found`

Solution:

1. Open Package Manager (Window > Package Manager)

2. Change dropdown from "In Project" to "Unity Registry"

3. Search for "Newtonsoft Json"

4. Install "com.unity.nuget.newtonsoft-json"

3. Microphone Permission Issues

This may indicate issues with the microphone. Please ensure that the microphone is connected correctly and applications have permission to access the microphone.

Solution:

- **Windows:** Check Windows microphone permissions
- **macOS:** System Preferences > Security & Privacy > Microphone
- **Unity:** Ensure microphone access in player settings

4. Character Not Responding

Possible Causes:

- Wrong Character ID
- API key not set correctly
- No internet connection
- Microphone not working

Solutions:

1. Verify Character ID matches Convai Playground

2. Check API key in Convai > API Key Setup

3. Test microphone in other applications

4. Check Unity Console for error messages

5. Ensure character has proximity trigger enabled

5. Animation Issues

The animation avatar that we are using might be incompatible with the character mesh. Fixing that can solve the issue.

Solution:

1. Check Avatar configuration on character model

2. Ensure animations are humanoid compatible

3. Verify Animator Controller has required parameters

4. Check for conflicting animation components

6. WebGL Build Issues

Solution:

1. File > Build Settings > WebGL

2. Player Settings > Other Settings

3. Uncheck "Assembly Version Validation"

4. Use Convai WebGL template

5. Ensure proper HTTPS hosting

7. macOS Permission Issues

MacOS's security protocols can prevent certain unsigned external DLLs, like grpc_csharp_ext.bundle, from functioning correctly in Unity.

Solution:

1. Right-click grpc_csharp_ext.bundle

2. Select "Open With" > Terminal

3. Run: sudo spctl --master-disable

4. Re-enable after testing: sudo spctl --master-enable

Debugging Tips

Enable Debug Logging

```
// Add to your character script

void Start()

{

    ConvaiNPC convaiNPC = GetComponent<ConvaiNPC>();

    convaiNPC.enableDebugLogs = true;

}
```

Check Network Connectivity

```
void TestConnection()

{

    StartCoroutine(CheckInternet());

}

IEnumerator CheckInternet()

{

    UnityWebRequest www = UnityWebRequest.Get("https://convai.com");

    yield return www.SendWebRequest();


    if (www.result == UnityWebRequest.Result.Success)

    {

        Debug.Log("Internet connection OK");

    }

    else
```

```
    {
        Debug.LogError("No internet connection");
    }
}
```

Advanced Features

1. NPC to NPC Conversations

Set Up Group NPC Controller: Add the NPC conversation manager component and enable it. Select the characters for AI to AI conversation and give them a topic for conversation.

```
using UnityEngine;
using Convai.Scripts.Runtime.Core;
public class NPCConversationManager : MonoBehaviour
{
    [Header("NPC Conversation Settings")]
    public ConvaiNPC[] conversationNPCs;
    public string conversationTopic = "discuss the weather";
    public float conversationInterval = 30f;

    void Start()
    {
        StartCoroutine(ManageConversations());
    }
```

```
    System.Collections.IEnumerator ManageConversations()
    {
        yield return new WaitForSeconds(5f); // Initial delay

        while (true)
        {
            InitiateConversation();
            yield return new WaitForSeconds(conversationInterval);
        }
    }

    void InitiateConversation()
    {
        if (conversationNPCs.Length >= 2)
        {
            // Start conversation between first two NPCs
            conversationNPCs[0].StartConversationWith(conversationNPCs[1], conversationTopic);
        }
    }
}
```

2. Custom Knowledge Integration

```
public class KnowledgeManager : MonoBehaviour
```

```
{
    [Header("Knowledge Settings")]
    public TextAsset knowledgeBase;
    public ConvaiNPC targetNPC;


    void Start()
    {
        if (knowledgeBase != null && targetNPC != null)
        {
            UpdateCharacterKnowledge();
        }
    }


    void UpdateCharacterKnowledge()
    {
        string knowledge = knowledgeBase.text;
        targetNPC.UpdateKnowledgeBase(knowledge);
    }
}
```

3. Lip Sync and Facial Animation

The model needs at least two animations: one for talking and one for Idle. Create an animator controller with the two animations and add a 'Talk' Boolean to trigger the animation.

```
public class LipSyncController : MonoBehaviour
```

```
{
    [Header("Lip Sync Settings")]
    public SkinnedMeshRenderer faceMesh;
    public string[] blendShapeNames = {"jawOpen", "mouthSmile"};

    private ConvaiNPC convaiNPC;

    void Start()
    {
        convaiNPC = GetComponent<ConvaiNPC>();
        convaiNPC.OnCharacterSpeaking += HandleSpeaking;
        convaiNPC.OnCharacterStopped += HandleStopped;
    }

    void HandleSpeaking(float[] lipSyncData)
    {
        // Apply lip sync data to blend shapes
        for (int i = 0; i < lipSyncData.Length && i < blendShapeNames.Length; i++)
        {
            int blendShapeIndex =
faceMesh.sharedMesh.GetBlendShapeIndex(blendShapeNames[i]);
            if (blendShapeIndex >= 0)
```

```
            {
                faceMesh.SetBlendShapeWeight(blendShapeIndex, lipSyncData[i] * 100);
            }
        }
    }

    void HandleStopped()
    {
        // Reset blend shapes
        for (int i = 0; i < blendShapeNames.Length; i++)
        {
            int blendShapeIndex =
faceMesh.sharedMesh.GetBlendShapeIndex(blendShapeNames[i]);
            if (blendShapeIndex >= 0)
            {
                faceMesh.SetBlendShapeWeight(blendShapeIndex, 0);
            }
        }
    }
}
```

4. Multiplayer Support

```
using Mirror;

public class MultiplayerConvaiNPC : NetworkBehaviour
```

```
{
    private ConvaiNPC convaiNPC;

    [SyncVar]
    public string currentResponse;

    void Start()
    {
        convaiNPC = GetComponent<ConvaiNPC>();

        if (isServer)
        {
            convaiNPC.OnCharacterResponse += HandleResponse;
        }
    }

    [Server]
    void HandleResponse(string response)
    {
        currentResponse = response;
        RpcBroadcastResponse(response);
    }
```

```
    [ClientRpc]
    void RpcBroadcastResponse(string response)
    {
        // Handle response on all clients
        Debug.Log($"NPC Response: {response}");
    }
}
```

Performance Optimization

1. Reduce API Calls

```
public class ConvaiOptimizer : MonoBehaviour
{
    [Header("Optimization Settings")]
    public float minTimeBetweenRequests = 2f;
    public int maxConcurrentRequests = 3;

    private float lastRequestTime;
    private int activeRequests = 0;

    public bool CanMakeRequest()
    {
```

```
        return Time.time - lastRequestTime >= minTimeBetweenRequests &&
               activeRequests < maxConcurrentRequests;
    }

    public void RegisterRequest()
    {
        lastRequestTime = Time.time;
        activeRequests++;
    }

    public void UnregisterRequest()
    {
        activeRequests--;
    }
}
```

2. Memory Management

```
public class ConvaiMemoryManager : MonoBehaviour
{
    void OnApplicationPause(bool pauseStatus)
    {
        if (pauseStatus)
        {
            // Pause Convai services
```

```
            ConvaiNPC[] npcs = FindObjectsOfType<ConvaiNPC>();
            foreach (var npc in npcs)
            {
                npc.enabled = false;
            }
        }
        else
        {
            // Resume Convai services
            ConvaiNPC[] npcs = FindObjectsOfType<ConvaiNPC>();
            foreach (var npc in npcs)
            {
                npc.enabled = true;
            }
        }
    }
}
```

Best Practices

1. Character Design

- Keep character personalities consistent
- Provide clear backstories
- Set appropriate knowledge boundaries

- Use specific voice and speech patterns

2. Performance

- Limit concurrent conversations
- Use proximity-based activation
- Cache frequent responses
- Optimize animation blending

3. User Experience

- Provide clear visual feedback
- Handle network errors gracefully
- Offer text alternatives for voice
- Test with different accents and languages

4. Security

- Never hardcode API keys in builds
- Use environment variables for keys
- Implement rate limiting
- Validate user input

Next Steps

Here are some ideas for expanding your project:

1. **Create Multiple Characters**: Add different personality types
2. **Implement Actions**: Make characters interact with objects
3. **Add Memory**: Let characters remember past conversations
4. **Build Scenarios**: Create guided experiences
5. **Deploy**: Build for your target platform

For more advanced features and updates, visit:

- Convai Documentation
- Convai Community Forum
- Convai YouTube Channel

Happy developing! 🚀