**Oracle: Origins - Character Creation & Customization System**

**Table of Contents**

1. Character Creator Overview
2. UMA 2 Implementation
3. Creation UI Framework
4. Customization Parameters
5. Class Selection Integration
6. Companion Selection
7. Character Preview System
8. Data Persistence
9. Implementation Scripts
10. Testing & Validation

**1. Character Creator Overview**

The character creation system for Oracle: Origins provides players with a comprehensive yet intuitive interface to design their character's appearance, select their elemental class, and choose their companion. This one-time creation process establishes the foundation for the player's journey.

**Core Features:**

* Detailed appearance customization using UMA 2
* Class selection with visual preview of abilities
* Companion selection with evolution preview
* Name and background selection
* Visual previews of customization changes
* Final character confirmation and save

**Design Philosophy:**

* Streamlined process with meaningful choices
* Visual feedback for all selections
* Clear connection between aesthetics and gameplay
* "What you see is what you get" approach

**2. UMA 2 Implementation**

**UMA 2 System Setup**

1. **Required UMA Components:**
   * Create a prefab named CharacterCreatorManager with:
     + DynamicCharacterAvatar component
     + UMACustomizationController (custom script)
     + UMAExpressionPlayer for facial animations
     + UMAData component configured for high-quality rendering
2. **UMA Asset Configuration:**
   * Import required UMA content:

- UMA Essentials

- Human Male

- Human Female

- Base Clothing Set

- Base Hair Set

Create custom Oracle: Origins assets:

Copy

- Class-Specific Eye Colors

- Oracle Clothing Set

- Special Hairstyles

- Face Marking Options

1. **UMA Recipe Configuration:**
   * Create base recipes for male and female characters
   * Configure shared color tables for consistent palette
   * Set up overlay library for class-specific visual elements
   * Prepare shared material settings for performance

**UMA Race Configuration**

1. **Human Race Setup:**
   * Configure the human race with:

- Skeleton: Humanoid

- Texture Resolution: 1024 (adjustable in settings)

- Mesh Resolution: High

- Animation Controller: HumanoidAnimator

 **Class-Specific Modifications:**

* Create DNA converters for class-based appearance changes:
  + Eye color ranges specific to each class
  + Subtle skin tone variations by class
  + Class-appropriate body type tendencies

 **Performance Settings:**

* Configure UMA text atlas options:

- Atlas Resolution: 2048x2048

- MipMaps: Enabled

- Compression: Normal Quality

- Maximum Slots: 4

* + Set LOD options for character preview

**3. Creation UI Framework**

**UI Layout Design**

1. **Main Creation Screens:**
   * Welcome/Introduction Screen
   * Appearance Customization Screen
   * Class Selection Screen
   * Companion Selection Screen
   * Background/Name Screen
   * Final Review Screen
2. **UI Component Structure:**

CharacterCreationCanvas (Canvas)

├── NavigationPanel

│ ├── AppearanceButton

│ ├── ClassButton

│ ├── CompanionButton

│ ├── BackgroundButton

│ └── ReviewButton

├── ScreenContainer

│ ├── WelcomeScreen

│ ├── AppearanceScreen

│ │ ├── GenderSelection

│ │ ├── FaceCustomization

│ │ ├── BodyCustomization

│ │ ├── HairCustomization

│ │ └── ClothingCustomization

│ ├── ClassScreen

│ │ ├── ClassOptions

│ │ ├── ClassDescription

│ │ ├── AbilityPreview

│ │ └── ClassStats

│ ├── CompanionScreen

│ │ ├── CompanionOptions

│ │ ├── CompanionDescription

│ │ ├── EvolutionPreview

│ │ └── CompanionAbilities

│ ├── BackgroundScreen

│ │ ├── NameInput

│ │ ├── BackgroundOptions

│ │ └── PersonalityTraits

│ └── ReviewScreen

│ ├── CharacterSummary

│ ├── FinalAppearance

│ ├── SelectedClass

│ ├── SelectedCompanion

│ └── ConfirmationButton

├── CharacterPreviewPanel

│ ├── CharacterModel

│ ├── RotateControls

│ ├── ZoomControls

│ └── LightingOptions

└── NavigationControls

├── PreviousButton

├── NextButton

├── RandomizeButton

└── ResetButton

1. **Navigation System:**
   * Tab-based navigation between main sections
   * Breadcrumb trail showing progress
   * Next/Previous buttons for sequential flow
   * Direct tab access once visited
   * Cannot proceed to final review until all sections completed

**UI Animation & Feedback**

1. **Transition Animations:**
   * Smooth fade between screens
   * Character model rotation when changing sections
   * Highlight effects for selected options
2. **Visual Feedback:**
   * Real-time updates to character model
   * Particle effects when changing class
   * Sound effects for selections
   * Animated highlights for recommended options

**4. Customization Parameters**

**Appearance Customization**

1. **Gender Selection:**
   * Male/Female binary choice affecting base skeleton
   * Different clothing and hair options per gender
   * Voice selection tied to gender choice
2. **Face Customization:**
   * Facial structure sliders (10 parameters):

Copy

- Face Width

- Face Height

- Jaw Shape

- Cheekbone Height

- Cheekbone Width

- Eye Size

- Eye Spacing

- Nose Shape

- Mouth Size

- Ear Size

Facial details:

- Skin Tone (12 options)

- Face Markings (8 options)

- Scars (6 options)

- Makeup/Tattoos (10 options)

- Aging (5 levels)

**Body Customization:**

* Body type sliders:

- Height

- Musculature

- Weight

- Shoulder Width

- Hip Width

Physical details:

- Body Markings

- Tattoos

- Scars

1. **Hair Customization:**
   * Hairstyle (15 options per gender)
   * Hair Color (24 color options)
   * Facial Hair (male only, 10 options)
   * Eyebrow Style (8 options)
   * Eyebrow Color (matches hair color)
2. **Clothing Customization:**
   * Starting outfit options (8 per gender)
   * Color schemes (6 per outfit)
   * Accessories (rings, necklaces, earrings, etc.)
   * Class-influenced style suggestions

**Parameter Implementation**

1. **DNA Modifiers:**
   * Set up UMA DNA for all customizable features
   * Create custom DNA converters for Oracle-specific features
   * Link DNA to UI sliders and selectors
2. **Overlay Options:**
   * Configure slot-based overlay system for:
     + Clothing items
     + Hair styles
     + Markings and tattoos
   * Set up shared materials for performance
3. **Randomization:**
   * Create balanced random selection algorithm
   * Class-appropriate randomization option
   * Category-specific randomization (face only, hair only, etc.)

**5. Class Selection Integration**

**Class Presentation**

1. **Class Display Items:**
   * Visual icon representing each class
   * Short description of class focus
   * Difficulty rating (1-5 stars)
   * Play style tags (e.g., "Aggressive", "Support", "Control")
2. **Class Details Panel:**
   * Expanded lore description
   * Starting abilities explanation
   * Recommended weapon types
   * Growth trajectory summary
3. **Visual Class Preview:**
   * Character model with class-specific aura
   * Eye color preview based on class
   * Sample ability animation on demand
   * Combat stance demonstration

**Class Selection Logic**

1. **Class-Specific Stats:**
   * Display base stat differences between classes:

Copy

Demon:

- Health: Medium-Low

- Energy: High

- Damage: Very High

- Defense: Low

Frost:

- Health: Medium

- Energy: Medium

- Damage: Medium

- Defense: High

Bolt:

- Health: Low

- Energy: Very High

- Damage: High

- Defense: Very Low

Nullify:

- Health: High

- Energy: Medium-Low

- Damage: Medium

- Defense: Medium

Regen:

- Health: Very High

- Energy: Low

- Damage: Low

- Defense: Medium-High

1. **Ability Preview:**
   * Show tier 1, 2, and 3 abilities
   * Demonstrate basic ability animation
   * Display cooldowns and effects
   * Show compatibility with companion types
2. **Class Selection Confirmation:**
   * Clear confirmation of class choice
   * Brief description of starting experience
   * Warning that class cannot be changed later

**6. Companion Selection**

**Companion Presentation**

1. **Companion Display Items:**
   * Rendered model of each companion type
   * Basic information panel
   * Personality description
   * Class synergy indicators
2. **Companion Details:**
   * Backstory and lore
   * Evolution path preview
   * Ability set description
   * Combat role explanation
3. **Evolutionary Stages Preview:**
   * Initial form (fully rendered)
   * Intermediate form (silhouette)
   * Final form (artistic concept)
   * Timeline of evolution

**Companion Selection Logic**

1. **Companion Abilities:**
   * Display unique abilities for each companion:

Copy

German Shepherd:

- Initial: Scent Tracking, Protective Bark

- Final: Pack Leader, Hunter's Instinct, Alpha's Howl

Hawk:

- Initial: Scout Ahead, Diving Attack

- Final: Burning Reconnaissance, Flame Strike, Rebirth Aura

Lynx:

- Initial: Silent Pounce, Night Vision

- Final: Shadow Strike, Void Sight, Shadow Meld

Bear Cub:

- Initial: Intimidating Roar, Powerful Swipe

- Final: Earth-Shaking Bellow, Crushing Maul, Living Fortress

**Class Synergies:**

* Highlight recommended companions for each class:

Demon pairs well with Wolf (fire synergy)

Frost pairs well with Bear (ice armor)

Bolt pairs well with Hawk (aerial attacks)

Nullify pairs well with Lynx (stealth)

Regen pairs well with any (healing bond)

1. **Companion Customization:**
   * Name your companion
   * Select basic coloration options
   * Choose personality tendency (aggressive, protective, playful)

**7. Character Preview System**

**Preview Environment**

1. **3D Preview Scene:**
   * Create dedicated lighting setup for character display
   * Neutral background with optional environments
   * Camera controls for rotation and zoom
   * Toggle between static and animated preview
2. **Animation Previews:**
   * Idle animation
   * Walking/running cycle
   * Combat stance
   * Class-specific ability preview
   * Companion interaction
3. **Environment Options:**
   * Neutral studio lighting
   * Earth environment preview
   * Titan environment preview
   * Hyperion environment preview
   * Ivides environment preview

**Technical Implementation**

1. **Camera Setup:**
   * Orbit camera system around character
   * Zoom functionality
   * Focus points for face, body, full character
   * Cinematic auto-rotation option
2. **Lighting System:**
   * Three-point lighting setup
   * Environment-specific lighting profiles
   * Rim lighting to highlight character features
   * Class-appropriate lighting tints
3. **Performance Considerations:**
   * LOD system for preview rendering
   * Simplified background for low-end systems
   * Option to disable extra effects
   * Progressive loading of preview assets

**8. Data Persistence**

**Save System Integration**

1. **Character Definition Storage:**
   * Create CharacterCreationData.cs to store all selections
   * Integrate with Easy Save 3 for serialization
   * Generate unique character ID on creation
2. **Stored Parameters:**
   * UMA DNA values and selections
   * Class selection
   * Companion selection and name
   * Character name and background
   * Creation timestamp
   * Starting loadout selections
3. **Character Loading:**
   * Character selection screen before game start
   * Preview of saved characters
   * Delete option with confirmation
   * New character creation entry point

**Implementation Example**

using System;

using System.Collections.Generic;

using UnityEngine;

using UMA;

using UMA.CharacterSystem;

namespace OracleOrigins.CharacterCreation

{

[Serializable]

public class CharacterCreationData

{

// Basic Info

public string characterID;

public string characterName;

public DateTime creationDate;

public bool isMale;

// Class Info

public int classTypeID;

public string className;

// Companion Info

public int companionTypeID;

public string companionName;

// Background

public int backgroundID;

public List<int> personalityTraits;

// UMA Data

public string umaRecipeJson;

public Dictionary<string, float> dnaValues;

public List<CustomizationSelection> customizationSelections;

// Game starting info

public int startingLoadoutID;

[Serializable]

public class CustomizationSelection

{

public string category;

public string optionID;

public Color color;

public float intensity;

}

// Constructor

public CharacterCreationData(DynamicCharacterAvatar avatar, string name, int classID, int companionID)

{

characterID = Guid.NewGuid().ToString();

characterName = name;

creationDate = DateTime.Now;

isMale = avatar.activeRace.name == "HumanMale";

classTypeID = classID;

className = GetClassNameFromID(classID);

companionTypeID = companionID;

companionName = "Companion"; // Default name

// Save UMA recipe

umaRecipeJson = avatar.GetCurrentRecipe();

// Store DNA values

dnaValues = new Dictionary<string, float>();

foreach (var dna in avatar.GetAllDNA())

{

dnaValues.Add(dna.Key, dna.Value);

}

// Other data initialized with defaults

backgroundID = 0;

personalityTraits = new List<int>();

customizationSelections = new List<CustomizationSelection>();

startingLoadoutID = 0;

}

private string GetClassNameFromID(int id)

{

switch (id)

{

case 0: return "Demon";

case 1: return "Frost";

case 2: return "Bolt";

case 3: return "Nullify";

case 4: return "Regen";

default: return "Unknown";

}

}

}

}

**9. Implementation Scripts**

**Character Creation Manager**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using UMA;

using UMA.CharacterSystem;

using ES3;

namespace OracleOrigins.CharacterCreation

{

public class CharacterCreationManager : MonoBehaviour

{

[Header("UMA References")]

[SerializeField] private DynamicCharacterAvatar characterAvatar;

[SerializeField] private Transform characterPreviewTransform;

[SerializeField] private Animator characterAnimator;

[Header("UI References")]

[SerializeField] private GameObject[] creationScreens;

[SerializeField] private Button nextButton;

[SerializeField] private Button backButton;

[SerializeField] private Button finishButton;

[Header("Preview Controls")]

[SerializeField] private Slider zoomSlider;

[SerializeField] private Slider rotationSlider;

[SerializeField] private Toggle animationToggle;

[Header("Creation Settings")]

[SerializeField] private int currentScreenIndex = 0;

[SerializeField] private float characterRotationSpeed = 30f;

[SerializeField] private string[] requiredScreens;

// Sub-managers

private AppearanceCustomizationManager \_appearanceManager;

private ClassSelectionManager \_classManager;

private CompanionSelectionManager \_companionManager;

private BackgroundSelectionManager \_backgroundManager;

// State tracking

private bool[] \_screenCompleted;

private bool \_characterLoaded = false;

private float \_targetRotation = 0f;

private void Awake()

{

// Get sub-managers

\_appearanceManager = GetComponent<AppearanceCustomizationManager>();

\_classManager = GetComponent<ClassSelectionManager>();

\_companionManager = GetComponent<CompanionSelectionManager>();

\_backgroundManager = GetComponent<BackgroundSelectionManager>();

// Initialize arrays

\_screenCompleted = new bool[creationScreens.Length];

// Setup UI events

nextButton.onClick.AddListener(NextScreen);

backButton.onClick.AddListener(PreviousScreen);

finishButton.onClick.AddListener(FinishCreation);

// Setup preview controls

zoomSlider.onValueChanged.AddListener(ZoomCharacter);

rotationSlider.onValueChanged.AddListener(RotateCharacter);

animationToggle.onValueChanged.AddListener(ToggleAnimation);

// Hide all screens

foreach (var screen in creationScreens)

{

screen.SetActive(false);

}

// Show first screen

if (creationScreens.Length > 0)

{

creationScreens[0].SetActive(true);

}

// Start with basic character

StartCoroutine(InitializeCharacter());

}

private IEnumerator InitializeCharacter()

{

// Create default character

characterAvatar.ChangeRace("HumanMale");

// Wait for UMA to generate

while (!characterAvatar.isActiveAndEnabled || !characterAvatar.gameObject.activeInHierarchy)

{

yield return null;

}

// Wait for the avatar to be built

yield return new WaitUntil(() => characterAvatar.BuildCharacterEnabled);

// Update avatar

characterAvatar.BuildCharacter();

yield return new WaitUntil(() => characterAvatar.built);

// Character is ready

\_characterLoaded = true;

// Notify sub-managers

\_appearanceManager.OnCharacterReady();

\_classManager.OnCharacterReady();

\_companionManager.OnCharacterReady();

\_backgroundManager.OnCharacterReady();

// Initialize default idle animation

if (characterAnimator != null)

{

characterAnimator.SetTrigger("Idle");

}

}

public void NextScreen()

{

// Mark current screen as completed

\_screenCompleted[currentScreenIndex] = true;

// Hide current screen

creationScreens[currentScreenIndex].SetActive(false);

// Move to next screen

currentScreenIndex = Mathf.Min(currentScreenIndex + 1, creationScreens.Length - 1);

// Show next screen

creationScreens[currentScreenIndex].SetActive(true);

// Update buttons

UpdateNavigationButtons();

// Update character rotation for the new screen

UpdateCharacterForCurrentScreen();

}

public void PreviousScreen()

{

// Hide current screen

creationScreens[currentScreenIndex].SetActive(false);

// Move to previous screen

currentScreenIndex = Mathf.Max(currentScreenIndex - 1, 0);

// Show previous screen

creationScreens[currentScreenIndex].SetActive(true);

// Update buttons

UpdateNavigationButtons();

// Update character rotation for the new screen

UpdateCharacterForCurrentScreen();

}

private void UpdateNavigationButtons()

{

// Update back button

backButton.interactable = currentScreenIndex > 0;

// Update next button

nextButton.interactable = currentScreenIndex < creationScreens.Length - 1;

// Update finish button

finishButton.gameObject.SetActive(currentScreenIndex == creationScreens.Length - 1);

finishButton.interactable = AreAllRequiredScreensCompleted();

}

private bool AreAllRequiredScreensCompleted()

{

// Check if all required screens are completed

for (int i = 0; i < requiredScreens.Length; i++)

{

int screenIndex = System.Array.IndexOf(creationScreens,

GameObject.Find(requiredScreens[i]));

if (screenIndex >= 0 && !\_screenCompleted[screenIndex])

{

return false;

}

}

return true;

}

private void UpdateCharacterForCurrentScreen()

{

// Rotate character based on current screen

switch (currentScreenIndex)

{

case 0: // Welcome screen

\_targetRotation = 180f; // Face forward

break;

case 1: // Appearance screen

\_targetRotation = 180f; // Face forward

if (characterAnimator != null)

characterAnimator.SetTrigger("Idle");

break;

case 2: // Class screen

\_targetRotation = 180f; // Face forward

if (characterAnimator != null)

characterAnimator.SetTrigger("CombatIdle");

break;

case 3: // Companion screen

\_targetRotation = 150f; // Angle to see companion

if (characterAnimator != null)

characterAnimator.SetTrigger("Pet");

break;

case 4: // Background screen

\_targetRotation = 180f; // Face forward

if (characterAnimator != null)

characterAnimator.SetTrigger("Talk");

break;

case 5: // Review screen

\_targetRotation = 180f; // Face forward

if (characterAnimator != null)

characterAnimator.SetTrigger("Heroic");

break;

}

}

public void ZoomCharacter(float zoomLevel)

{

// Implement camera zoom

Camera.main.fieldOfView = Mathf.Lerp(20f, 60f, 1f - zoomLevel);

}

public void RotateCharacter(float rotation)

{

// Manual rotation overrides automatic rotation

\_targetRotation = rotation \* 360f;

characterPreviewTransform.rotation = Quaternion.Euler(0f, \_targetRotation, 0f);

}

public void ToggleAnimation(bool animate)

{

// Toggle animation state

if (characterAnimator != null)

{

characterAnimator.enabled = animate;

if (animate)

{

// Restart current animation

UpdateCharacterForCurrentScreen();

}

}

}

public void FinishCreation()

{

// Gather all data from sub-managers

string characterName = \_backgroundManager.GetCharacterName();

int classID = \_classManager.GetSelectedClassID();

int companionID = \_companionManager.GetSelectedCompanionID();

// Create character data

CharacterCreationData characterData = new CharacterCreationData(

characterAvatar,

characterName,

classID,

companionID

);

// Add additional data

characterData.companionName = \_companionManager.GetCompanionName();

characterData.backgroundID = \_backgroundManager.GetSelectedBackgroundID();

characterData.personalityTraits = \_backgroundManager.GetSelectedTraits();

// Save the data

SaveCharacterData(characterData);

// Proceed to game start

StartGame(characterData.characterID);

}

private void SaveCharacterData(CharacterCreationData data)

{

// Get the saved character list or create a new one

List<string> characterList;

if (ES3.KeyExists("characterList"))

{

characterList = ES3.Load<List<string>>("characterList");

}

else

{

characterList = new List<string>();

}

// Add the new character ID to the list

characterList.Add(data.characterID);

// Save the updated list

ES3.Save("characterList", characterList);

// Save the character data

ES3.Save("character\_" + data.characterID, data);

}

private void StartGame(string characterID)

{

// Set the active character ID for the game to load

ES3.Save("activeCharacterID", characterID);

// Load the game scene

UnityEngine.SceneManagement.SceneManager.LoadScene("GameStart");

}

}

}

Appearance Customization Manager

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using UMA;

using UMA.CharacterSystem;

namespace OracleOrigins.CharacterCreation

{

public class AppearanceCustomizationManager : MonoBehaviour

{

[Header("UMA References")]

[SerializeField] private DynamicCharacterAvatar characterAvatar;

[Header("UI References")]

[SerializeField] private Toggle maleToggle;

[SerializeField] private Toggle femaleToggle;

[SerializeField] private Slider[] faceSliders;

[SerializeField] private Slider[] bodySliders;

[SerializeField] private Dropdown hairStyleDropdown;

[SerializeField] private Dropdown facialHairDropdown;

[SerializeField] private Button[] skinToneButtons;

[SerializeField] private Button[] hairColorButtons;

[SerializeField] private Dropdown outfitDropdown;

[SerializeField] private Button randomizeButton;

[Header("DNA Settings")]

[SerializeField] private string[] faceSliderDNA;

[SerializeField] private string[] bodySliderDNA;

[Header("Customization Options")]

[SerializeField] private string[] maleHairOptions;

[SerializeField] private string[] femaleHairOptions;

[SerializeField] private string[] maleFacialHairOptions;

[SerializeField] private Color[] skinToneColors;

[SerializeField] private Color[] hairColors;

[SerializeField] private string[] maleOutfitOptions;

[SerializeField] private string[] femaleOutfitOptions;

// State tracking

private bool \_isInitialized = false;

private bool \_isUpdating = false;

private string[] \_currentHairOptions;

private string[] \_currentOutfitOptions;

public void OnCharacterReady()

{

// Initialize UI

InitializeUI();

// Mark as initialized

\_isInitialized = true;

}

private void InitializeUI()

{

// Setup gender toggles

maleToggle.onValueChanged.AddListener(isMale => {

if (isMale) ChangeGender(true);

});

femaleToggle.onValueChanged.AddListener(isFemale => {

if (isFemale) ChangeGender(false);

});

// Default to male

maleToggle.isOn = true;

\_currentHairOptions = maleHairOptions;

\_currentOutfitOptions = maleOutfitOptions;

// Setup face sliders

for (int i = 0; i < faceSliders.Length && i < faceSliderDNA.Length; i++)

{

int index = i; // Capture for lambda

faceSliders[i].onValueChanged.AddListener(value => {

if (!\_isUpdating) UpdateDNA(faceSliderDNA[index], value);

});

}

// Setup body sliders

for (int i = 0; i < bodySliders.Length && i < bodySliderDNA.Length; i++)

{

int index = i; // Capture for lambda

bodySliders[i].onValueChanged.AddListener(value => {

if (!\_isUpdating) UpdateDNA(bodySliderDNA[index], value);

});

}

// Setup hair style dropdown

PopulateHairDropdown();

hairStyleDropdown.onValueChanged.AddListener(ChangeHairStyle);

// Setup facial hair dropdown if available

if (facialHairDropdown != null)

{

PopulateFacialHairDropdown();

facialHairDropdown.onValueChanged.AddListener(ChangeFacialHair);

}

// Setup skin tone buttons

for (int i = 0; i < skinToneButtons.Length && i < skinToneColors.Length; i++)

{

int index = i; // Capture for lambda

skinToneButtons[i].GetComponent<Image>().color = skinToneColors[i];

skinToneButtons[i].onClick.AddListener(() => ChangeSkinTone(index));

}

// Setup hair color buttons

for (int i = 0; i < hairColorButtons.Length && i < hairColors.Length; i++)

{

int index = i; // Capture for lambda

hairColorButtons[i].GetComponent<Image>().color = hairColors[i];

hairColorButtons[i].onClick.AddListener(() => ChangeHairColor(index));

}

// Setup outfit dropdown

PopulateOutfitDropdown();

outfitDropdown.onValueChanged.AddListener(ChangeOutfit);

// Setup randomize button

randomizeButton.onClick.AddListener(RandomizeAppearance);

// Apply default settings

ApplyDefaultSettings();

}

private void ApplyDefaultSettings()

{

\_isUpdating = true;

// Reset DNA to defaults

for (int i = 0; i < faceSliders.Length; i++)

{

faceSliders[i].value = 0.5f;

}

for (int i = 0; i < bodySliders.Length; i++)

{

bodySliders[i].value = 0.5f;

}

// Default hair style

hairStyleDropdown.value = 0;

// Default facial hair (if male)

if (facialHairDropdown != null)

{

facialHairDropdown.value = 0;

}

// Default skin tone

ChangeSkinTone(2); // Medium tone

// Default hair color

ChangeHairColor(0); // First color

// Default outfit

outfitDropdown.value = 0;

\_