Documentation
Character creator

Welcome to Small Scale Interactive Character Creator

Character Creator is a modular 2D character customization system designed for Unity. It allows you to visually build and animate characters by combining interchangeable gear, skin tones, weapons, backpacks, and effects. Whether you're crafting a single hero or generating dozens of unique characters, this toolset gives you the flexibility to swap, tint, preview, and export pixel-perfect spritesheets with ease.

Index

1.1 Core scripts

The core scripts in *Character Creator* manage the core functionality of character customization, gear management, and spritesheet generation.

1.2 AnimatorGearSwitcher.cs

Purpose:
This script allows dynamic switching of gear visuals and colors for a character part (e.g., head, chest, legs). It supports:

- Cycling through multiple gear options via UI buttons.
- Changing gear color using toggles or a global color swatch panel.
- Applying colors to weapons or individual gear parts.
- Randomizing gear and color for a quick visual variety.

This is a **core component** of the **Character Creator** system and is intended to be attached to individual character parts like head, chest, legs, or shoes.

Gear Visuals & Animator Control

Fields

- List<RuntimeAnimatorController> animatorControllers: List of gear variations as animator controllers.
- int currentAnimatorIndex: Tracks the current gear being shown.
- Animator animator: Used to apply the selected RuntimeAnimatorController.

Methods

- NextAnimator(): Cycles to the next gear animator in the list.
- PreviousAnimator(): Cycles to the previous gear animator.
- RandomGear(): Randomly selects a gear animator and a random color.

Color Picker System

Toggle-Based Color Selection

- Toggle[] colorToggles: UI toggle buttons linked to specific colors.

- Color[] toggleColors: Defines the actual colors used by the toggles.
- ChangeColor(int index): Applies the color from a selected toggle to the current gear.

Global Color Swatch Panel (Shared UI)

- GameObject colorSwatchPanel: Shared panel used for swatch-based color selection.
- Button[] colorSwatchButtons: Buttons representing available swatch colors.
- TextMeshProUGUI colorInfoText: Displays the name and hex value of the hovered color swatch.
- Button closeColorPickerButton: UI button to close the color picker without applying changes.
- SetupColorSwatchButtons(): Initializes swatch buttons with click and hover logic.
- SetupCloseButton(): Initializes the close button functionality.

Color Target Handling (Static System)

To allow centralized control, the following static variables are used:

- SpriteRenderer currentTarget: The currently selected single gear part.
- List<SpriteRenderer> currentWeaponTargets: All weapon sprites currently targeted.
- bool globalSwatchSetup: Prevents multiple swatch setups across different parts.
- GameObject globalColorSwatchPanel: The shared swatch panel UI reference.

Methods

- OpenColorPicker(): Sets the current gear sprite as the color target and opens the swatch panel.
- OpenWeaponColorPicker(): Targets all assigned weapon sprites and opens the swatch panel.
- CloseColorPicker(): Closes the color picker UI and clears selected targets.

Weapon Color Handling

- List<SpriteRenderer> weaponSpriteRenderers: Assigned weapon parts that need synchronized coloring.
- Allows all weapon visuals (e.g., handle, blade, glow) to be colored together.

Initialization Logic

Inside Start():

- Connects UI button callbacks.
- Applies the first gear animator controller by default.
- Initializes toggle visuals and color listeners.

- Sets up the global swatch system (once per session).
- Randomizes gear and color on startup for specified body parts (isHead, isChest, etc.).

Gear Type Flags

- isHead, isChest, isLegs, isShoes, isSkin: Boolean tags to help classify which part the script is applied to. Also used to trigger gear randomization on Start().

Usage Overview

To Use This Script:

1. Attach it to a character gear part (e.g., Head, Chest).
2. Assign:
   - A list of animator controllers (gear variations).
   - UI Buttons for next, previous, and random.
   - Toggle buttons and their corresponding colors.
   - A shared global color swatch panel with color buttons and a close button.
3. For weapons, also assign all weapon SpriteRenderers under weaponSpriteRenderers.

Gear Control

- Click **Next** or **Previous** to cycle through gear options.
- Click **Random** to assign a random gear and random color.

Color Control

- Use **Toggle buttons** or **Color Swatch Panel** to change colors.
- Gear and weapon coloring are handled separately for better customization.

Integration with Character Creator

This script works in tandem with:

- The AnimationController for applying animation state logic.
- A central UI prefab that includes gear selection and color customization tools.

It forms the interactive, modular core that allows players or designers to quickly swap and customize gear pieces visually at runtime or in the editor preview.

1.3 GearPresetManager.cs

Purpose:
This script allows users to apply predefined gear and color configurations (presets) to the character via UI toggles. It manages:

- Switching entire outfits with one click.
- Enabling/disabling specific gear components like weapons, shields, and backpacks.
- Applying both visual style (color) and animation controllers for gear parts.

Preset System Overview

GearPreset Class

A GearPreset is a data container representing a full gear setup for the character. Each preset includes:

- **Head, Chest, Legs, Shoes**:
  - RuntimeAnimatorController for gear animations.
  - Color for visual appearance.
- **Skin Color**:
  - A single Color for skin tone.
- **Weapon**:
  - A GameObject (with SpriteRenderer) to represent the equipped weapon.
- **Shield (Optional)**:
  - A GameObject and Color to define a visible shield, if any.
- **Backpack (Optional)**:
  - A GameObject and Color to define a visible backpack, if any.

Defined in the Unity Inspector using [System.Serializable].

Fields & Configuration

Preset Management

- List<GearPreset> presets:
  Collection of all available gear presets configured in the Inspector.
- Toggle[] presetToggles:
  UI toggles linked to each preset. These can optionally belong to a ToggleGroup for exclusive selection.

Gear Slot References

For each gear slot (head, chest, legs, shoes), you assign:

- Animator component to change gear visuals.
- SpriteRenderer to change gear color.

Skin Slot

- SpriteRenderer skinRenderer:
  Changes the skin tone.

Weapon, Shield & Backpack References

- List<GameObject> allWeaponObjects: All possible weapon objects. Used to disable unused ones.
- List<GameObject> allShieldObjects: All shield variations available.
- List<GameObject> allBackpackObjects: All backpacks that can be toggled.

Startup Logic

Start()

- Validates that the number of toggles matches the number of presets.
- Assigns each UI toggle to its corresponding preset using onValueChanged.

Preset Application

ApplyPreset(int presetIndex)

Applies the selected preset to the character. This method:

1. **Gear & Skin**
   - Sets animation controllers and colors for head, chest, legs, shoes, and skin.
2. **Weapon**
   - Disables all weapon sprites.
   - Enables only the one assigned in the preset.
3. **Shield**
   - Disables all shields.
   - Enables the one from the preset (if assigned) and applies its color.
4. **Backpack**
   - Disables all backpacks.
   - Enables the one from the preset (if assigned) and applies its color.

Usage Notes

- Presets are fully customizable through the Unity Inspector.
- UI toggles provide a clean interface for users to preview or select different looks.
- Shields and backpacks are **optional** — if not assigned in the preset, none will be shown.
- Color and animation can be customized per preset without affecting others.

Typical Use Case

Attach GearPresetManager to an empty GameObject in your scene.
In the Inspector:

1. Create and configure presets.
2. Assign your toggle buttons.
3. Drag in all character part references (Animator + SpriteRenderers).
4. Add all available weapons, shields, and backpacks for control.

When a toggle is selected, the character instantly updates to match the defined look in that preset.

1.4 SpritesheetGenerator.cs

This script manages the UI and logic for generating composite character spritesheets based on equipped gear, skin color, and visual effects. It dynamically creates new textures by layering sprite assets and exporting them into a combined output folder, supporting optional effects like shadows and gunfire flashes.

Serialized Fields

Gear Slots

- Animator shoesAnimator, chestAnimator, legsAnimator, headAnimator
  Controls the animation previews for each corresponding gear piece.
- SpriteRenderer shoesRenderer, chestRenderer, legsRenderer, headRenderer
  Used to determine the gear color and visibility in the preview.
- TextMeshProUGUI shoesGearNameText, chestGearNameText, legsGearNameText, headGearNameText
  Displays the selected gear animator name in the UI.

- TextMeshProUGUI shoesColorText, chestColorText, legsColorText, headColorText
  Shows the current color of each gear piece.

Weapon Setup

- Weapon[] weapons
  List of all possible weapons, each with a GameObject, Animator, and SpriteRenderer.
- TextMeshProUGUI weaponNameText, weaponColorText
  Shows the active weapon's name and color.

Backpack and Shield Setup

- Backpack[] backpacks, Shield[] shields
  Lists of all possible backpack/shield objects and their renderers.
- TextMeshProUGUI backpackNameText, backpackColorText, shieldNameText, shieldColorText
  UI elements for displaying the selected backpack and shield info.

Skin Setup

- SpriteRenderer skinColorRenderer
  Holds the tint color for the character's body.
- TextMeshProUGUI skinColorText
  Displays the skin's hex color in UI.
- Toggle skinToggle
  If unchecked, skin will be excluded from the generated spritesheets.

Shadow Setup

- Animator shadowAnimator
  Controls the shadow animation.
- SpriteRenderer shadowRenderer
  Defines shadow color and visibility.
- TextMeshProUGUI shadowGearNameText, shadowColorText
  UI representation of the shadow layer.

GunFire Setup

- Animator gunFireAnimator, SpriteRenderer gunFireRenderer
  Gunfire visual effect.
- TextMeshProUGUI gunFireNameText, gunFireColorText
  UI showing the gunfire effect’s name and color.
- Toggle gunFireToggle
  Enables or disables gunfire overlays in the generated spritesheets.

Load Screen

- GameObject loadScreenPanel
  Shows a loading overlay during generation.
- Slider loadProgressSlider, TextMeshProUGUI currentlyGeneratingTMP
  Progress indicator and current task status text.

Public Methods

void StartCombineSpritesheets()

Kicks off the spritesheet generation coroutine using the current character configuration.

Core Coroutine

IEnumerator CombineCharacterSpritesheetsCoroutine()

Main coroutine that handles the following:

- Reads currently selected gear, colors, and effects from UI.
- Loads base and gear textures from folders based on gear names.
- Applies shadow alpha and tints to each layer.
- Performs layered texture blending using alpha compositing.
- If GunFire is enabled, overlays the effect on Attack1 and additional generated attack animations (RunAttack, etc.).
- Saves each final combined texture to an output folder with timestamp-based naming.
- Updates progress UI throughout.

Helper Methods

UpdateGearUI()

Called in Start() and Update() to constantly sync the UI with currently selected gear, color, and effects.

UpdateWeaponUI(), UpdateBackpackUI(), UpdateShieldUI()

Updates the weapon, backpack, and shield sections of the UI respectively by detecting which sprite renderer is enabled.

UpdateSkinColorUI()

Displays the currently selected skin tone as a hex color.

Utility Methods

Texture2D LoadTexture(string path)

Loads a texture from disk into memory for compositing.

Color ParseColorFromTMP(string hex)

Converts a hex color string (e.g., #FF0000) into a Unity Color.

float ParseAlphaPercentage(string percent)

Parses a percentage string (e.g., 50%) into a normalized float value (0.5).

Color AlphaBlend(Color bottom, Color top)

Blends two colors using alpha compositing (used during texture stacking).

string GetAnimatorName(Animator anim)

Returns the name of the currently loaded animator controller or fallback object name.

string GetColorHex(Color c)

Returns the hex code (e.g., #00FF00) of a given color.

string GetAlphaValue(Color c)

Returns the alpha of a color as a percentage string ("80%").

Summary

The SpritesheetGenerator is a powerful utility for previewing and exporting customized character spritesheets. It reads all selected gear and their visual states, processes them into layered textures, applies effects like shadow and gunfire, and saves out clean composite PNGs for each animation. It's designed to be used inside a character creator interface and supports both visual previews and batch export workflows.

1.5 TooltipManager.cs

Handles displaying UI tooltips when the user hovers over elements in the character creator interface. Includes customizable delays, fade-in/out animation, and positioning behavior.

Serialized Fields

Tooltip Panel Setup

- GameObject tooltipPanel
  The UI panel that will be shown as the tooltip. Should be a Canvas element in *Screen Space - Overlay* mode.
- TextMeshProUGUI tooltipHeader
  Text field for the tooltip's header or title line.
- TextMeshProUGUI tooltipContent
  Main body of the tooltip text.

Tooltip Text Settings

- string headerText
  Static or user-assigned header text to be shown on hover.
- string contentText
  Multi-line text field used for explaining the feature or UI element in detail.

Tooltip Timing Settings

- float delayBeforeShow
  Time (in seconds) to wait after hovering before the tooltip appears.
- float fadeDuration
  Duration for fade-in and fade-out transitions.

Tooltip Positioning

- Vector3 offset
  Positioning offset from the mouse cursor to avoid overlap.
- bool useFixedPosition
  If true, the tooltip appears at its original position instead of following the mouse.

Tooltip Display Control

- bool hideTooltip
  If false, the tooltip will never show (it's moved far off-screen). Useful for toggling tooltips via global settings.

Private Fields

- CanvasGroup tooltipCanvasGroup
  Used to control tooltip panel transparency during fades.
- Coroutine tooltipCoroutine
  Tracks the currently running tooltip coroutine (for stopping and restarting).
- Vector3 originalTooltipPosition
  Stores the tooltip's initial position, used when useFixedPosition is true.
- Vector3 offScreenPosition
  A hidden location far off-screen for hiding the tooltip without disabling the GameObject.

Unity Event Methods

void Awake()

Initializes internal references, sets the tooltip off initially, and ensures a CanvasGroup component exists for fade control.

Interface Methods

Implements Unity's IPointerEnterHandler and IPointerExitHandler.

void OnPointerEnter(PointerEventData eventData)

Triggered when the pointer enters the UI element. Starts the coroutine to show the tooltip.

void OnPointerExit(PointerEventData eventData)

Triggered when the pointer exits the UI element. Starts the coroutine to hide the tooltip.

Coroutines

IEnumerator ShowTooltipCoroutine()

- Waits for delayBeforeShow.
- Populates the header and content text fields.
- Positions the tooltip based on either mouse location (offset) or its original location.
- Gradually fades in the tooltip using CanvasGroup.alpha.

IEnumerator HideTooltipCoroutine()

- Gradually fades out the tooltip.
- Disables the panel once the fade completes.

Summary

The TooltipManager provides dynamic, animated tooltips for any UI element it's attached to. With features like:

- Delayed show
- Mouse-following or fixed placement
- Smooth fade-in/out
- Configurable text and UI behavior

…it makes tooltip integration seamless for gear buttons, swatches, color pickers, or any other interactive elements in the Character Creator.

2.1 Extra scripts

The scripts in this section are considered extras, meaning the scripts can be found in all my assets, and is only meant to showcase the asset features in the example scene. It has a lot of code that is perhaps not directly linked to the asset you have, since it’s a general purpose script. It is not meant to be used in a final game since it’s not optimized.

2.2 AnimationController.cs

The AnimationController serves as the central hub for managing all animation-related logic during character showcase. It allows for direction-sensitive movement and attacks, integrates effects like muzzle flashes and blood, and supports a wide range of gameplay-style animation demonstrations—all through simple keyboard and mouse inputs.

Serialized Fields

- Animator animator: The main animator controlling the character's animations.
- Animator muzzleAnimator: Animator used for gunfire muzzle flash effects.
- SpriteRenderer muzzleFlashRenderer: SpriteRenderer to manage the visibility of the muzzle flash.
- Color defaultColor: Sets the base color of the character's sprite.
- List<GameObject> bloodPrefabs: Visual effects for regular damage.

- List<GameObject> radiatedPrefabs: Visual effects for radiation damage.
- float rollTime: Time duration before ending the roll animation.

Public Fields

- string currentDirection: Keeps track of the character's current direction (e.g., "isEast").
- bool isCurrentlyRunning: Debug flag to check if movement input is active.
- bool isCrouching: State flag for crouching.
- bool isDying: Prevents other animations from playing if the character is dead.
- PlayerController playerController: External reference for player state (e.g., isMelee, isDead).
- bool isRadiated: Determines which damage visual effects to use.

Core Methods

Start()

Initializes references, especially the character's SpriteRenderer.

Update()

Handles keyboard and mouse input to:

- Trigger special animations (e.g., take damage, crouch, cast spell).
- Determine directional movement.
- Handle attack input.
- Block animations if the character is dead.

Utility Methods

ResetAnimator()

Fully resets the animator's state and parameters. Useful for scene reloads or respawns.

SetDefaultColor()

Resets the character's sprite color to defaultColor.

UpdateDirection(string newDirection)

Updates the animator's directional state (e.g., "isNorth"). Automatically resets attack booleans if direction changes.

DetermineDirectionFromAngle(float angle)

Converts an angle to one of 8 directional animation states (e.g., isEast, isSouthWest).

SetDirectionBools(...)

Helper method to set all directional animator booleans.

Movement Handling

HandleMovement()

Determines movement direction based on mouse and keyboard inputs. Updates animator parameters for running, strafing, and crouching in 8 directions.

SetMovementAnimation(bool isActive, string baseKey, string direction)

Sets movement animation flags for the animator (e.g., MoveNorth, CrouchRunEast).

ResetAllMovementBools()

Resets all directional movement booleans (including crouch run).

Attack Handling

HandleAttackAttack()

Controls right-click attack input. Switches between running and standing attack animations. Also handles muzzle flash direction and toggling states based on input release.

TriggerAttack(bool isRunning, string direction)

Triggers an attack animation based on the current movement state and direction.

ResetAttackAttackParameters()

Resets all attack-related animator parameters.

RestoreDirectionAfterAttack()

Restores movement states after attack animations end.

ResetAllGunFireBools()

Disables all gunfire states in the muzzle animator.

Animation Triggers (Directional Variants)

Each action (e.g., crouch, die, kick) has:

- A Trigger method that activates the appropriate directional animation.
- A coroutine that resets animation booleans after a short delay.

Examples:

- TriggerCrouchIdleAnimation() → Plays crouch idle animation.
- TriggerDie() → Plays the death animation based on current direction.
- TriggerRollAnimation() → Plays the roll animation in the current direction.
- TriggerSlideAnimation() → Plays sliding animation.
- TriggerFlipAnimation() → Plays flip/acrobatics animation.
- TriggerAttackSpinAnimation() → Performs a spinning melee attack.
- TriggerPummelAnimation() → Triggers a punch/pummel animation.
- TriggerKickAnimation() → Triggers a kick animation.
- TriggerSpecialAbility1Animation() / TriggerSpecialAbility2Animation() → Plays special ability animations.
- TriggerCastSpellAnimation() → Casts a spell based on direction.

Effects & Damage

TriggerTakeDamageAnimation()

Spawns blood or radiation effect (based on isRadiated) at the character's position.

SpawnEffect()

Randomly selects a prefab and instantiates it for the damage visual.

UpdateSpriteOrder(GameObject effectInstance)

Ensures the damage effect displays in the correct render layer after a short delay.

2.3 PlayerController.cs

This script can be found in all my assets, and is only meant to showcase the asset. It has a lot of code that is perhaps not directly linked to the asset you have, since it's a general purpose script. It is not meant to be used in a final game since its not optimized.

This script manages player behavior in the demo scene, including movement, directional control, health, shooting (projectile and melee), crouching, special abilities, and interaction with visual and audio feedback elements. It supports both ranged and melee character types with optional summoner and shapeshifter modes.

Core Components & Dependencies

- AnimationController animationController: Reference to the character's animation logic.
- Rigidbody2D rb: Handles physics-based movement.
- CircleCollider2D circleCollider: Used for collision detection.
- SpriteRenderer spriteRenderer: Used to apply visual feedback like color changes.
- AudioSource gunfireAudioSource: Plays shooting sounds.
- Slider healthSlider: UI element showing current health.
- GameObject gameOver: Game over UI shown on player death.
- TextMeshProUGUI killCountText: Displays kill count with pulsing effect.

Character Configuration Flags

- isActive: Whether the character is controllable.
- isRanged: True if character uses ranged attacks.
- isMelee: True if character uses melee attacks.
- isStealth: Makes the character semi-transparent while crouching.
- isShapeShifter: Enables shapeshifting ability.
- isSummoner: Enables summoning ability.

Combat & Abilities

- **Ranged Configuration**:
  projectilePrefab, projectileSpeed, shootDelay, bulletDamage, bulletsPerSecond, bulletLinePrefab, etc.

- **Melee Configuration**:
  meleePrefab, directional handling, and prefabs for close-range attacks.
- **Special Abilities**:
  AoEPrefab, Special1Prefab, HookPrefab, and ShapeShiftPrefab.

Health & UI

- maxHealth, currentHealth: Defines and tracks health.
- isDead: Flag to block interaction and movement after death.
- zombieKillCount: Tracks total kills.
- killCountText: UI text for score with animated pulse effect.

Key Functionalities

Movement

Update()

Main loop handling input and movement state updates, ability triggers, animation states, and prefab instantiations for actions.

FixedUpdate()

Executes actual movement using physics (Rigidbody2D.MovePosition()).

HandleMovement()

Handles input-based movement in all 8 directions including backwards and strafe movement, accounting for crouching and stairs.

SnapAngleToEightDirections(float angle)

Snaps a direction angle to one of eight compass angles (e.g., 0°, 45°, 90°, ...).

Health

TakeDamage(int damageAmount)

Reduces health, updates health slider, plays take damage animation, and checks for death.

Die()

Handles player death: disables collider, plays death animation, disables movement, shows game over UI, and restarts the scene after delay.

RestartSceneAfterDelay(float delay)

Waits a set time before reloading the current scene.

Combat – Ranged & Melee

HandleShooting()

Plays a shooting sound when right-clicking.

HandleZombieDamage()

Uses raycasting to apply damage to enemies along the shooting path. Supports piercing shots and creates a visual tracer using a LineRenderer.

ShowShotLine(List<Vector2> hitPoints)

Draws a visual tracer (bullet trail) along the raycast path.

ShootProjectile(Vector2 direction)

Spawns and propels a projectile in a specific direction.

DelayedShoot()

Fires a projectile after a configured delay (used for timing attacks).

Quickshot()

Fires five projectiles in quick succession in the current direction.

CircleShot()

Fires projectiles in 8 directions around the player.

Abilities

DeployAoEDelayed()

Spawns an AoE prefab either at the player or mouse position depending on summoner status.

DeploySpecial1Delayed()

Spawns a special ability prefab with a delay based on the character type.

DeployHookDelayed()

Summoner: Spawns the hook at mouse position.
Non-summoner: Spawns hook in facing direction.

ShapeShiftDelayed()

Triggers shapeshift animation with optional effect instantiation.

Status & Customization

SetArcherStatus(bool status)

Enables or disables ranged combat mode.

SetMeleeStatus(bool status)

Enables or disables melee mode.

SetActiveStatus(bool status)

Toggles whether the player character is active.

FlashGreen()

Temporarily flashes the character green (can be used for status effects like healing or buffs).

Visual Effects

PulseTextEffect(TextMeshProUGUI text)

Adds a temporary scaling animation ("pulse") to the kill counter UI when a zombie is killed.

Collision Events

OnTriggerEnter2D(Collider2D other)

Sets isOnStairs = true when entering a "Stairs" collider (affects movement angles).

OnTriggerExit2D(Collider2D other)

Resets isOnStairs when exiting stairs.

Summary

The PlayerController script is the main driver of character interaction in the demo scene. It handles:

- **Movement logic** for 8-directional gameplay.
- **Combat** with melee and ranged attacks.
- **Health management** with UI feedback.
- **Support for multiple character types** (ranged, melee, stealth, summoner, shapeshifter).
- **Sound and VFX integration** for immersive feedback.

Designed for the **Character Creator** system, it enables fully featured and interactive demo characters ready for real-time preview and testing.

2.4 SmoothCameraFollow.cs

This script smoothly follows a target (typically the player character) with optional look-ahead based on movement direction. It ensures a responsive and cinematic camera feel for 2D games.

Serialized Fields

Target & Offset

- Transform target: The object the camera will follow.
- Vector3 offset: The camera's position offset relative to the target.

Smooth Movement

- float smoothTime: Controls how quickly the camera reaches its target position.
- Vector3 velocity: Used internally by Vector3.SmoothDamp to track camera movement velocity.

Look-Ahead Settings

- bool enableLookAhead: Enables/disables the camera looking ahead in the direction the target is moving.
- float lookAheadDistance: How far ahead the camera should look based on target movement.
- float lookAheadSpeed: Speed at which the look-ahead adjustment is applied.
- Vector3 currentLookAhead: Internally tracks the current look-ahead offset.
- Vector3 lastTargetPosition: Stores the target’s position from the previous frame to calculate movement direction.

Unity Event Methods

void Start()

Initializes the lastTargetPosition to the target’s position, preventing large jumpy camera offsets on the first frame.

void LateUpdate()

Runs every frame **after** all Update() calls, ensuring the camera follows the latest target position. Handles:

- Calculating how much the target moved.
- Computing the look-ahead offset if enabled.
- Calculating the desired camera position based on target, offset, and look-ahead.
- Smoothly interpolating the camera’s current position toward the target using Vector3.SmoothDamp.

Features & Behavior

- **Smooth Following**: Prevents jittery movement and creates a smooth trailing effect when the target moves.
- **Look-Ahead Mode** *(optional)*:
  When enabled, the camera shifts slightly ahead in the direction of movement, making gameplay feel more dynamic and anticipatory.
- **Offset Support**: Ensures the camera isn’t locked exactly on the target, allowing for artistic framing of the scene (e.g., showing more screen space in front of the player).
- **Z-Lock for 2D**: Keeps the camera’s z-axis locked to avoid visual glitches in 2D setups.

Usage Notes

- Attach this script to the main camera in your 2D scene.
- Assign the player character to the target field in the Inspector.
- Adjust smoothTime, offset, and lookAheadDistance to fine-tune the camera feel.

3.1 Tips and Tricks

There’s a lot you can do with Character Creator and some features might not be immediately obvious. Here are a few helpful tips to make the most of the tool:

3.2 List of tips and tricks

1. **Disable Tooltips:** You can toggle tooltips on or off using the setting found in the Settings panel.
2. **Customizable Shadow:** Adjust the character’s shadow transparency to your liking. Setting it to 0% will completely remove the shadow from the final spritesheet.
3. **Disable Gunfire Effects:** To exclude gunfire effects from your output, simply turn off the gunfire toggle in the Settings panel. This will not only remove the muzzle flash from the "Attack1" animation but also prevent auto-generation of related attack animations (e.g., RunAttack, StrafeLeftAttack, etc.), allowing you to add your own muzzle flash if needed.
4. **Color Customization:** Most gear parts can be individually customized with color pickers. Some also include preset color buttons for quick selection.
5. **Modular Rendering:** You can enable or disable any gear part when generating a spritesheet. For example, if you want to render just the base character, turn off all gear, weapons, shields, and backpacks—then choose a skin color and render.
6. **Unarmed Characters:** Want your character to be unarmed? Just click the active weapon toggle again to disable it—perfect for NPCs and non-combatants.
7. **Presets for Inspiration:** Use the preset buttons to quickly apply ready-made gear sets. These are curated by the developer to showcase unique outfits and styles.
8. **Optional Equipment:** Shields and backpacks are entirely optional. Leave them disabled if not needed.
9. **Editable Preset Colors:** Each gear part (Head, Chest, Legs, Shoes) has 5 preset color buttons. You can customize these presets by modifying their corresponding SpriteRenderer colors in the Inspector (outside of Play mode).
10. **Custom Swatch Colors:** The color picker panel's swatch buttons can also be customized. Just select each button in the Inspector (while not in Play mode) and change its SpriteRenderer color to whatever you like.

3.2 List of Animations

Below is a list of all available animations and how they are commonly used:

- **Attack1** – The standard ranged weapon attack (ideal for guns).
- **Attack2** – A throwing animation, useful for grenades, molotovs, or similar.
- **Attack3** – First melee animation: a powerful overhand weapon swing.
- **Attack4** – Second melee animation: a backhand swing, perfect for combos.
- **CrouchIdle** – A stationary crouched pose; ideal for sneaking or hiding.
- **CrouchRun** – Moving while crouched, useful for stealth gameplay.

- **Die** – The death animation; used when a character is defeated.
- **Idle** – Default idle stance with weapon equipped.
- **Idle2** – Aiming stance; great for characters holding guns.
- **Idle3** – Casual idle with arms relaxed at the sides. Perfect for unarmed NPCs.
- **Run** – Running forward. Adds gunfire when enabled.
- **RunBackwards** – Moving backward. Gunfire muzzle added if enabled.
- **StrafeLeft** – Strafing left. Includes gunfire if enabled.
- **StrafeRight** – Strafing right. Includes gunfire if enabled.
- **Walk** – Casual walking. Great for NPCs or patrol behavior.
- **TakeDamage** – A flinch or damage reaction. Can also be used for dodging.
- **Taunt** – A general-purpose action animation. Suitable for reloads, interacting with objects, using tools (e.g., chainsaws, lockpicks), opening chests, crafting etc.