What is «2D/3D Paint»?

«2D/3D Paint» - is an asset for Unity that allows you to paint on 2D and 3D objects! To use asset you need to add a component and configure a few parameters!

With «2D/3D Paint» you will be able to paint on 2D and 3D components such as: MeshFilter, SkinnedMeshRenderer, SpriteRenderer and RawImage.

Requirements

For correct work «2D/3D Paint» requires:

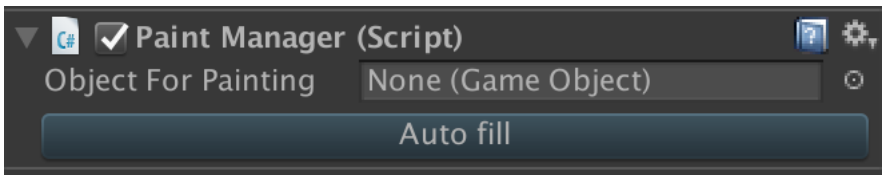
- At least 10 megabytes free on disk space;
- Unity 5.5.6 or newer;
- GameObject with supported components such as MeshFilter, SkinnedMeshRenderer, SpriteRenderer or RawImage with material and texture. Also when you are using the MeshFilter or SkinnedMeshRenderer component, make sure that their models have UV map.

Quick Start

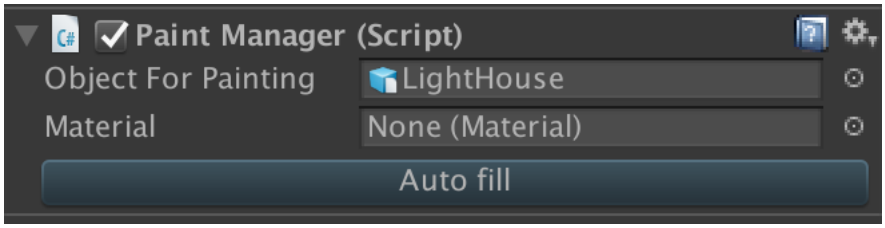
Add **PaintManager** component to the GameObject using the Unity menu «Component -> 2D/3D Paint». Select the GameObject on which you want to paint in the «Object For Painting» field. The object on which you want to paint must contain one of the supported components:

- MeshFilter
- SkinnedMeshRenderer
- SpriteRenderer
- RawImage

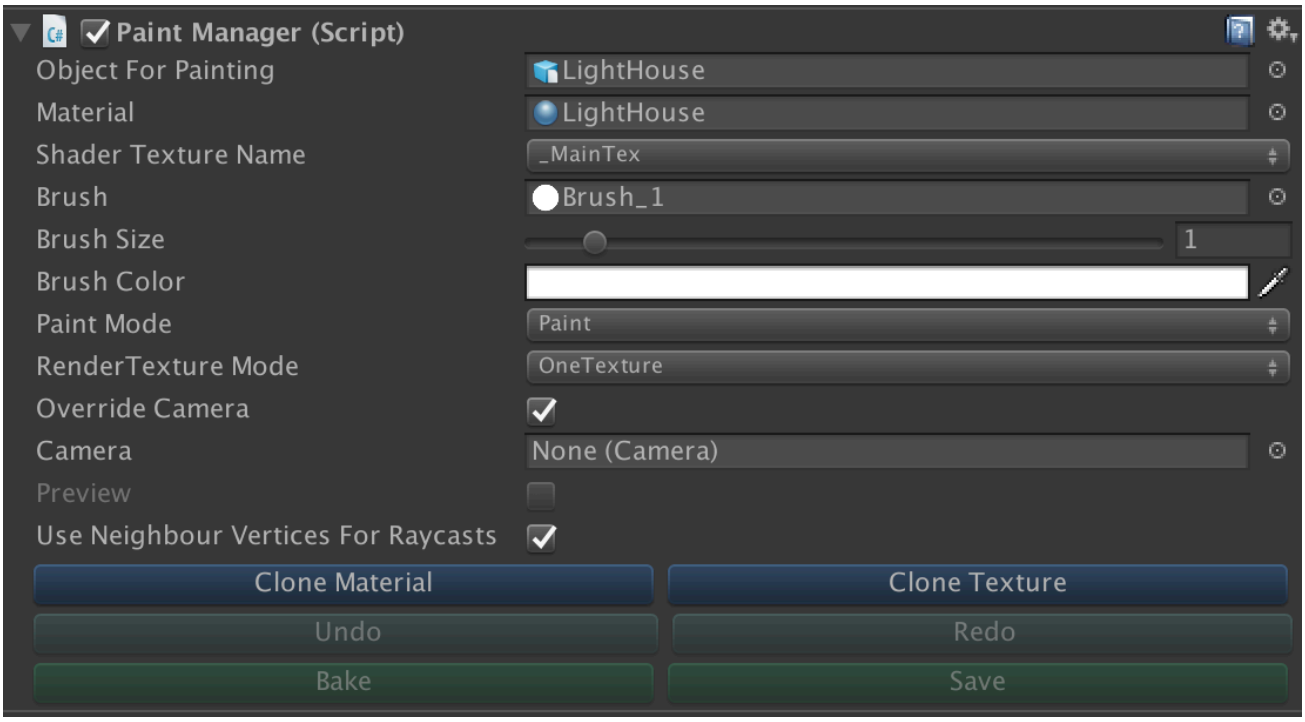
If you add a **PaintManager** component to a GameObject and any of its child objects contains an object with one of the supported components, it can automatically assign it to the «Object For Painting» field by clicking on the «Auto fill» button:



After clicking on the «Auto fill» button, the «Object For Painting» field will be filled in when one of the supported components will be found:



After that, in the same way, you can assign the material to paint «Material» by repeatedly pressing the «Auto fill» button. If the material or painting object cannot be found automatically, make sure that there is a GameObject with a supported component among the child objects, otherwise «Object For Painting» and «Material» can be assigned manually:

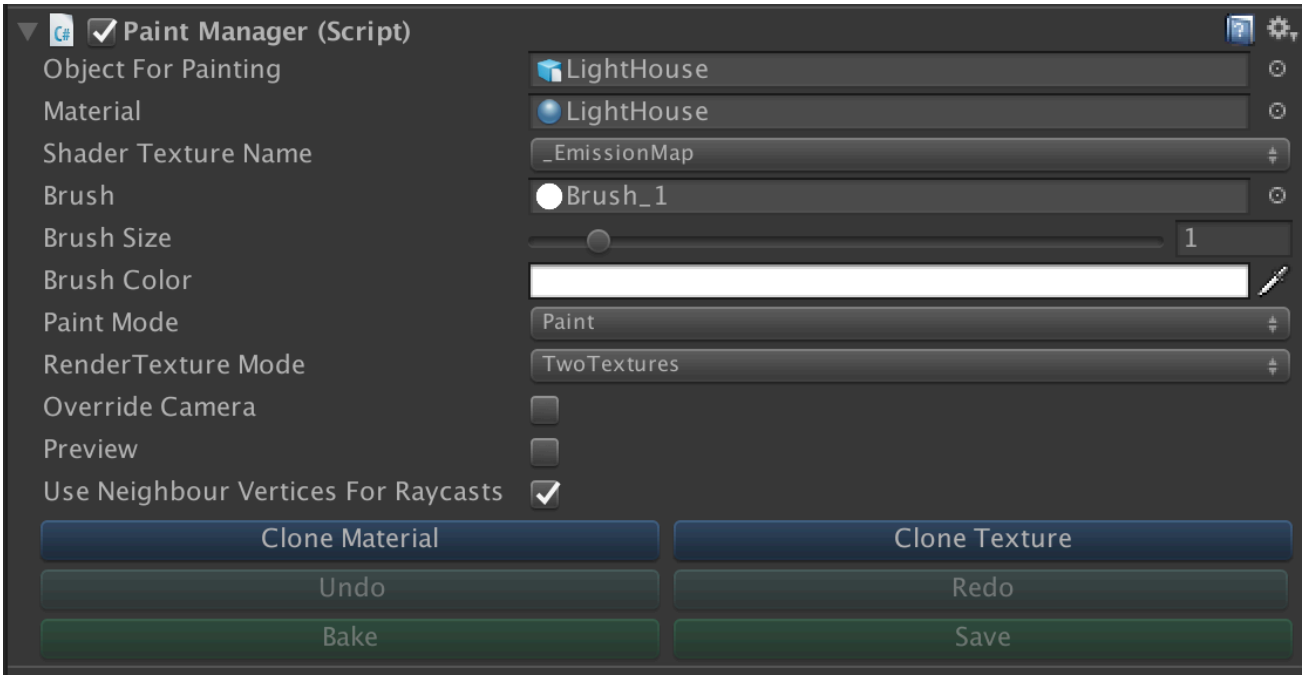


In the presence of values in the fields «Object For Painting» and «Material» will appear the rest of the settings component. Consider the existing settings:

- **Shader Texture Name** - the name of the material texture on which the painting will be performed;
- **Brush** - brush texture. Make sure that «Wrap Mode» in the settings of the brush texture equals «Clamp»;
- **Brush Size** - brush size;
- **Brush Color** - brush color;
- **Paint Mode** - paint mode. Supported painting modes:
  - Paint - painting on texture;
  - Erase - erase;
  - EraseBackground - erase background;
  - Restore - restore mode.
- **RenderTexture Mode** - Count of RenderTexture's for painting. Supported value:
  - OneTexture - one texture mode. Brush preview are not available in this mode and PaintMode.EraseBackground behaves the same way as PaintMode.Erase;
  - TwoTextures - two textures. In this mode, the first texture is using for painting, second texture is written to the source texture, and then the first texture is written. Brush preview is available in this mode.
- **Override Camera** - asset overrides the camera to determine the intersection of the ray with triangles and to work with user input, if the flag is set to false, the camera will be obtained from the Camera.main;

- **Camera** - camera for determining ray intersections with the triangles and for working with user input;
- **Preview** - show preview of the brush when user hovers over the object to be painted;
- **Use Neighbors Vertices For Raycasts** - asset uses neighbors vertices to find the intersection of the ray with triangles while lines are drawing. In the case of setting the flag the results of calculations can be inaccurate when non-convex objects are using. When we are using objects with a large number of the vertices the performance of searching the intersection of the ray with triangles is degrading. It is recommended to set the flag as true for mobile devices;
- **Clone Material** - button to copy the source material of the object to a new file, can be used only in the Unity Editor;
- **Clone Texture** - button to copy the source texture of the object to the new file, can be used only in the Unity Editor;
- **Undo** - button to undo action with the object, can be used only in the Unity Editor;
- **Redo** - button to redo action with the object, can be used only in the Unity Editor;
- **Bake** - button to save the painting results to the source file. Note that texture is stored in RAM and is not written to disk, can be used only in the Unity Editor;
- **Save** - button to save the texture of the painting results to the file, can be used only in the Unity Editor.

Let's set up the object to paint on the emission texture:

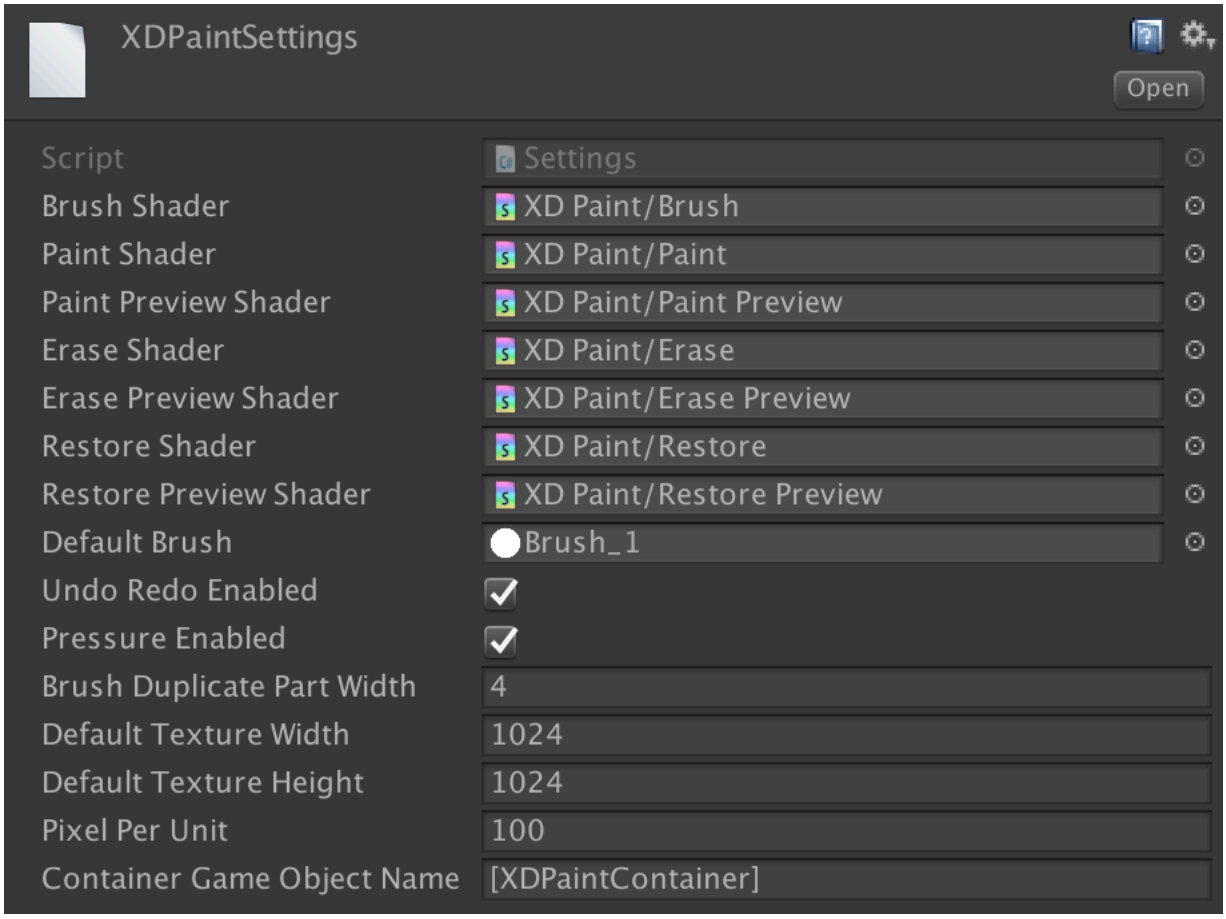


Texture of Shader with the name «_EmissionMap» was selected and the mode «TwoTextures» for «RenderTexture Mode» was chosen. Mode «TwoTextures» was set to be able to erase the painted without affecting the source texture. After switching to the game mode, the user will be able to paint on the object «LightHouse». User can use input devices for painting: a computer mouse or a touch device.

After switching to the game mode, the material and the source texture «_EmissionMap» will be copied, RenderTexture will be assigned as the new texture.

Settings

«2D/3D Paint» has global settings. The configuration file is located at: Assets/XDPaint/Resources/XDPaintSettings.asset. The settings file is a ScriptableObject with fields. Consider the fields of the settings file:



- **Default Brush** - default brush texture;
- **Undo Redo Enabled** - undo and redo functionality is enabled;
- **Pressure Enabled** - pressure force is applied to the brush size;
- **Brush Duplicate Part Width** - the width of the duplicated part of the brush, the value affects the number of vertices and smoothness of the line while lines are drawing. The higher the value, the less the number of vertices will be drawn;
- **Default Texture Width** - the default texture width is using for objects that do not have the source texture;
- **Default Texture Height** - the default texture height is using for objects that do not have the source texture;
- **Pixel Per Unit** - pixelPerUnit field for sprites. It is using for objects that do not have the source sprite;
- **Container Game Object Name** - the name of the GameObject, for the container object with InputController and RaycastController components;

How it works

General description

«2D/3D Paint» clones the source material and replaces the source texture with the RenderTexture in which the painting takes place. Before that, the source texture is copied in the RenderTexture. The asset passes data to (BasePaintObject) using a class to handle user input(InputController). The UV coordinate is calculated in (BasePaintObject) to determine the position of the painting on the texture. Painting takes place on the previously created RenderTexture and is stored in GPU memory, which provides high performance.

Painting

«2D/3D Paint» creates a RenderTexture(in ARGB32 format) texture. The size of the RenderTexture is equal to the size of the source texture, if there is no source texture, the size for RenderTexture will be taken from the settings. For such objects as MeshFilter and SkinnedMeshRederer, the asset uses a ray-surface intersection to determine the intersection of the model triangle with the ray. The use of models with a large number of vertices can lead to loss of performance. To solve this problem, it was implemented to find neighbors triangles using them when drawing lines.

«2D/3D Paint» has 2 modes of operation with RenderTexture - one texture(RT0) and two textures(RT0 and RT1).

In the case when PaintMode is set as OneTexture, painting occurs in RT0, when initialization in RT0, the source texture is written;

When the PaintMode is set as TwoTextures painting is done in RT0, RT1 is written then the source texture, over which is superimposed RT0, thus painting mode TwoTextures happens without affecting the source texture.

Undo/redo

«2D/3D Paint» saves painting data in the following form:

- UV-positions of the painting;
- Brush sizes when painting;
- Brush texture;
- Brush size;
- Brush color;
- Painting mode;

When painting with the undo/redo is enabled, it leads to increase memory consumption due saving all painting operations.

Input

«2D/3D Paint» processes input using the InputController class. Input can be either with the mouse or by touching the screen.

Raycast

For MeshFilter and SkinnedMeshRederer objects, finding the intersection of a ray with a triangle is used to calculate the UV coordinates. These calculations are performed on the CPU, the time of which depends on the number of vertices of the model. In order to avoid high CPU loads, it should be noted that the more vertices the model contains, the more CPU time it will take to find the intersection.

The method to optimize the calculations has been added to draw lines based on the data on neighbors triangles - to draw a line from triangle «A» to triangle «B» asset can use the data on neighbors triangles to find the entry and exit positions of the triangle. This method significantly improves performance for objects with a large number of vertices, but can draw a line with inaccuracy for non-convex objects, because it does not check data about other triangles that may lie «closer» to the camera and close the verifiable(neighboring) vertices.

API Help

Content

PaintManager script

The script is a manager and the main link between all auxiliary classes. It combines the main script for painting on objects, contains instances of MaterialsController, BasePaintObject.

Main public fields, properties and methods:

public GameObject objectForPainting - GameObject of the object to be painted;
public MaterialsContainer materialsContainer - MaterialsController instance;
public RenderTexturesMode renderTextureMode - render texture mode (count of RenderTexture's);
public bool shouldOverrideCamera - override the camera;
public BasePaintObject PaintObject { ... } - property of the painted object;
public Camera Camera { ... } - property of the camera;
public bool Preview { ... } - property of preview mode;
public bool UseNeighborsVerticesForRaycasts { ... } - property of using neighbors vertices for raycasts when drawing lines;
public bool Initialized { ... } - property of the initialization status of the object;
public void Init() - object initialization;
public void Render() - invoke object rendering;
public void FillTrianglesData(bool fillNeighbors = true) - fill model data, argument - is fill data about neighbors triangles;
public void ClearTrianglesData() - remove filled information about triangles;
public void ClearTrianglesNeighborsData() - remove filled information about the neighboring triangles;
public RenderTexture GetRenderTexture() - gets RenderTexture;
public Texture2D GetResultTexture() - gets the resulting texture;
public void Bake() - write changes to the source texture, used in Unity Editor.

MaterialsContainer script

Class for storage and management of materials. It can be used to create, configure, and manage content.

Basic public properties and methods:

public string ShaderTextureName { ... } - the name of the texture in the Shader that will be replaced by RenderTexture;
public Texture SourceTexture { ... } - source texture;
public int SourceTextureWidth { ... } - width of the source texture;
public int SourceTextureHeight { ... } - height of the source texture;
public PaintMode PaintMode { ... } - texture painting mode
public bool Preview { ... } - preview of the brush;
public Brush Brush { ... } - brush material class;
public Paint Paint { ... } - painting material class;
public void Init(IRenderComponentsHelper renderComponentsHelper, bool preview) - material container initialization;
public void InitMaterial(Material material) - setting the source material;
public void RestoreTexture() - restores the source material/texture for a painting object;
public void SetObjectMaterialTexture(Texture texture) - sets texture to paint as a texture material;

Brush script

A class for storing and managing data about brush material and its parameters.

Basic public properties and methods:

public Material Material { ... } - brush material;
public Color Color { ... } - brush color;
public Texture Texture { ... } - texture of the brush;
public float Size { ... } - brush size;
public OnChangeColor onChangeColor; - event of changing the brush color;
public OnChangeTexture onChangeTexture; - event of changing the texture of the brush;
public void Init() - initializing the brush material;
public void SetPaintMode(PaintMode paintMode) - sets the brush painting mode;

Paint script

A class for storing and managing data about painting material and its parameters.

Basic public properties and methods:

public Material Material { ... } - painting material;

public void Init(Texture sourceTexture) - painting material initialization;
public void SetColor(Color color) - set the painting color;
public void SetTexture(Texture texture) - set the painting texture;
public void SetPaintMode(PaintMode paintMode, bool preview, Texture brushTexture) - set the painting mode;
public void SetPaintTexture(Texture texture) - set texture painting to material painting;
public void SetPaintPreviewVector(Vector4 brushOffset) - set data to display the painting preview.

InputController script

Script-singleton is a user input controller and control it.

Main public fields, properties and methods:

public event OnInputPositionHit OnMouseHover - mouse hover event;
public event OnInputPositionHit OnMouseHoverWithHit - mouse hover event on objects with MeshFilter and SkinnedMeshRederer;
public event OnInputPositionHitPressure OnMouseDown - left mouse click event;
public event OnInputPositionHitPressure OnMouseDownWithHit - event of pressing the left mouse on the objects with a MeshFilter and SkinnedMeshRederer;
public event OnInputPositionHitPressure OnMouseButton - left mouse button pressing event;
public event OnInputPosition OnMouseUp - event of releasing the left key of the mouse;
public Camera Camera { ... } - camera property used for user input and raycasts.

RaycastController script

Script-singleton is a controller for performing audits of ray intersections with the triangles and the keeper of the data about all available objects to make a Raycast checks.

Main public fields, properties and methods:

public Camera Camera { ... } - property of a camera, used for raycasts;
public void InitObject(Camera newCamera, Component paintComponent, Component renderComponent, Triangle[] triangles) - initialize a new mesh object;
public void Raycast(Ray ray, out Triangle triangle) - check the intersection of ray with triangles objects, the result will be returned to out Triangle triangle;
public void RaycastLocal(Ray ray, Transform objectTransform, out Triangle triangle) - check the intersection of the ray with the triangles of the Transform objectTransform, the result will be returned to out Triangle triangle;
public void NeighborsRaycast(Triangle triangle, Ray ray, out Triangle outTriangle) - check the intersection of the ray with neighboring triangles of Triangle triangle, the result will be returned to out Triangle outTriangle;

BasePaintObject script

Base class for painting on RenderTexture. It can be declared as **CanvasRendererPaint**, **MeshRendererPaint**, or **SpriteRendererPaint**. The derived classes **CanvasRendererPaint**, **MeshRendererPaint**, and **SpriteRendererPaint** contain logic to check the painting position based on the data from the InputController and return the UV texture position for further work according to the base class logic.

Main public fields, properties and methods:

public event PaintHandler OnPaintHandler - painting event, can be used by the developer to obtain data about painting;
public bool IsPainting { ... } - property, is user is painting;
public new Camera Camera { ... } - camera property used to determine the position of the painting on the texture;
public StateKeeper StateKeeper - an instance of a class that stores data about painting States, contains painting data in the form of positions, used to undo and redo actions;
public bool Preview - preview flag when hovering the cursor over an object;
public void InitPaint(Camera camera, Transform objectTransform, IMaterialsContainer materialsContainer, IRenderTextureHelper renderTextureHelper) - initialization of an object;
public void FinishPainting() - force end painting;
public void OnRender() - render to RenderTexture;
public void RenderCombined() - render to RenderTexture, in the case when using two RenderTexture's (PaintManager.renderTextureMode as RenderTexturesMode.TwoTextures);
public void ClearTexture() - clear the texture to paint, to display changed data you must call the render method using Render();

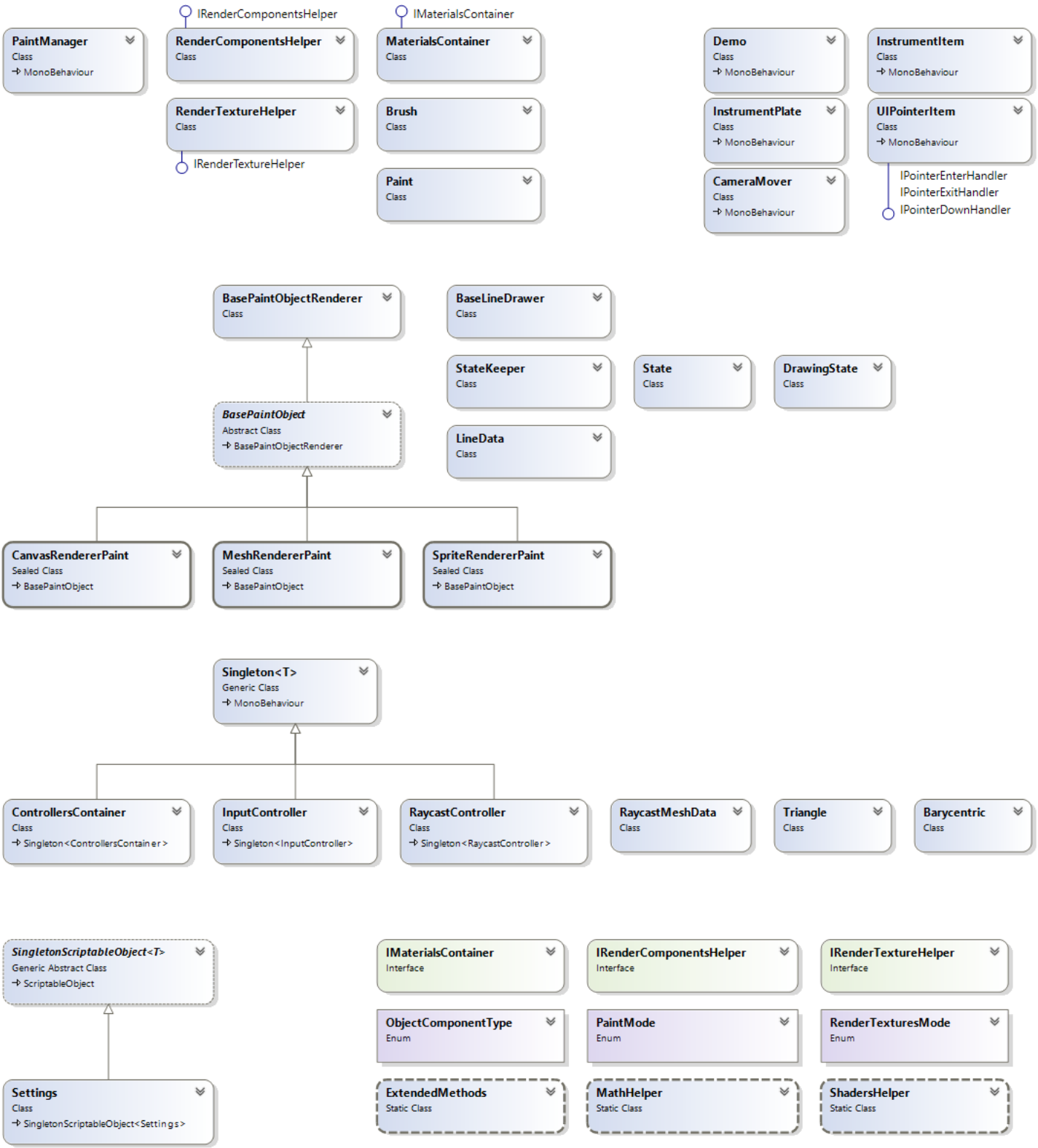
StateKeeper script

A class for storing and managing States. It contains data about painting on textures and used for undo and redo operations.

Main public methods:

public void Init(Action<int, int, Dictionary<int, State>> extraDraw, bool enabled) - initialization of the object, the first argument - action for rendering according to the passed data, the second argument-is functionality is enabled;
public void AddState(Vector2 position, float brushSize) - adds a state of the painting;
public void AddState(Vector2[] positions, float[] brushSizes) - add painting states;
public void Undo() - undo the action;
public void Redo() - redo the action;
public void Reset() - deletes all stored data for undo and redo;
public bool CanUndo() - checks if undo is possible;
public bool CanRedo() - checks if redo is possible.

Class diagram



Tips

Here are tips which will help in its configuration and use «2D/3D Paint»:

- Some of the objects with the component type SkinnedMeshRederer may not work properly because of its scale. Make sure that the SkinnedMeshRederer object has a scale of (1, 1, 1). This restriction does not apply to the bones of the object or its parents;
- Using a custom brush, make sure to set their textures «Wrap Mode» set as «Clamp» in order to avoid duplication brushes with preview mode on;
- Use the «Use Neighbors Vertices For Raycasts» for **PaintManager** if it is possible. Using this flag allows you to draw lines for the objects such as MeshFilter and SkinnedMeshRederer using less CPU time, but there may be inaccuracies with painting on non-convex objects. Please note that generation of data may take a few minutes, it depends on count of vertices and takes up disk space;
- Field «Triangles» of **PaintManager** contains data about the vertices of the model. Data can take up disk space and contain the indices of vertices, the number(ID) of a triangle and the indices of the vertices of the neighboring triangles. Use the context menu to add and/or delete data. If the flag «Use Neighbors Vertices For Raycasts» is set to false, then there is no need to use the data of neighboring triangles and this data will be deleted. This field does not show by Inspectors in Unity Editor, but can be viewed using Debug mode of Inspector window of Unity Editor;
- Field «Override Camera» for **PaintManager** sets the camera for the objects **InputController** and **RaycastController**, which are singleton-objects. Thus, it should be understood that the «Camera» field in **PaintManager** can overwrite the field values of singleton objects in the case when two or more **PaintManager** with different camera settings are used at the same time;
- «2D/3D Paint» supports Unity Lightweight Render Pipeline, for LWRP use LWRP-compatible shaders for object for painting.

Please let me know if you have any questions, ideas or suggestions.
E-mail: unitymedved@gmail.com