Projectile animator

# Summary

This extension allows you to animate position of a group of objects using premade textures. Objects retain collisions, and can also run their own logic. This extension uses unity jobs to maximize performance.

Contents

[Summary 1](#_Toc89453760)

[Texture painting 2](#_Toc89453761)

[Basics 2](#_Toc89453762)

[Example: 2](#_Toc89453763)

[Texture baker 3](#_Toc89453764)

[Textures import settings. 3](#_Toc89453765)

[Converting textures to a projectile animation asset 3](#_Toc89453766)

[Working with Projectile Driver in editor 4](#_Toc89453767)

[Settings 4](#_Toc89453768)

[Gizmos 6](#_Toc89453769)

[Working with Projectile Driver in code 7](#_Toc89453770)

[Passive variables 7](#_Toc89453771)

[Active variables 7](#_Toc89453772)

[Setting asset through script. 7](#_Toc89453773)

[Controlling animation 7](#_Toc89453774)

[Serialization 7](#_Toc89453775)

[Serialization 7](#_Toc89453776)

[Deserialization 7](#_Toc89453777)

# Texture painting

## Basics

All data about each projectile and its position is stored in the texture:

Each pixel can represent one projectile:

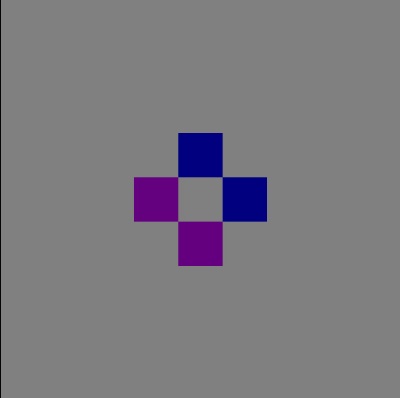
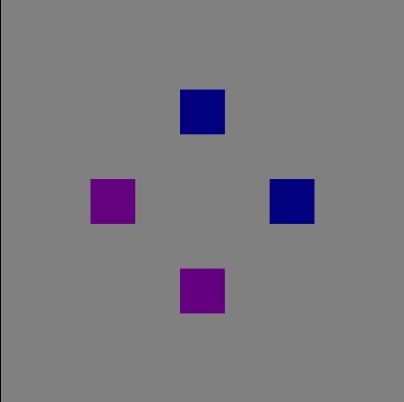
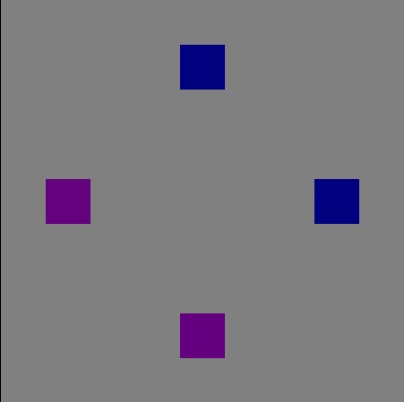
1. **Height and width** correspond with **projectile’s** **x and y position in local/world space.**
2. **Red value** corresponds with **projectile’s prefab ID.** Its minimum is 1, colors with red value = 0 **are ignored during texture reading.**
3. **Green value** corresponds with **projectile’s ID among projectiles with same prefab.**
4. **Blue value** corresponds with **projectile’s z position in local/world space.** By default, z = 0 when blue value = 127.
5. **Alpha value** points to an additional texture from which to take. More on that in Additional Texture paragraph.

## Example

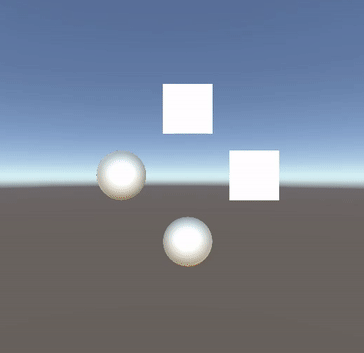
Let’s say we want to make 4 projectiles move from center, to form a cross. Top and right projectiles should be of one projectile type (for example, a cube). And bottom and left projectile should be of another type (for example, a ball). We want texture size to be 9x9 and we want to animation be done in 3 frames.

Firstly, create a palette with 4 different colors. For top and right projectiles, we use (1, 0, 127) and (1, 1, 127) for bottom and left ones we use (100, 0, 127) and (100, 1, 127).

In the end we have (in grey areas red = 0):

1.  2)  3) 

Which results in (standard trails added for clarity):



# Additional Textures

## Basics

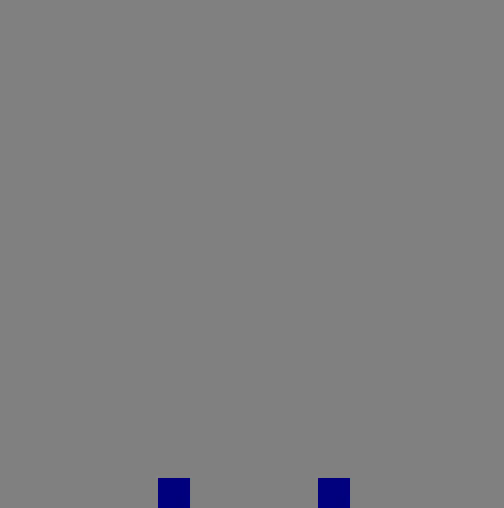
Some times you want several objects to appear at same XY or exact same position. To achieve this, you need to use Additional Textures. When alpha of a pixel does not equal to 255, data from pixel at same position on correspondent additional texture is baked.

Pay attention, that pixel that you are pointing to can’t have combinations of red and green values, that are already present on the texture you are pointing from.

Additional textures need to follow certain naming principles: their names must contain “AdditionalTexture” followed by an integer, identifying their correspondent alpha (MyNameAdditionalTexture155.png). Alpha = 255 is ignored while baking, use it for objects that are not supposed to point to any additional texture. If you want to place more than two objects at the same position, you can layer additional textures, meaning pixel on your regular texture points to a pixel on additional texture 1, which points to a pixel on additional texture 2 and so on.

## Example:

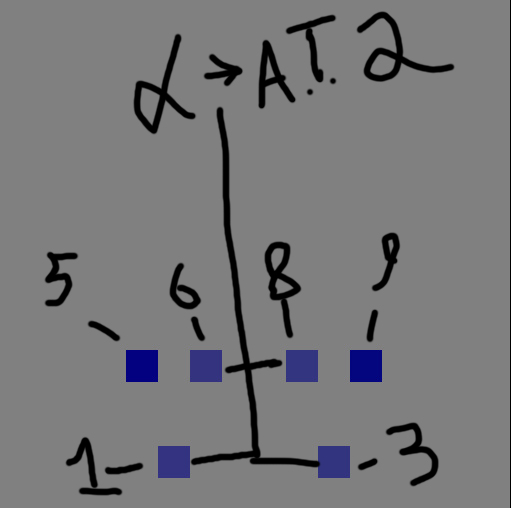
Let’s say we want to make an animation of some objects subdividing them selves into two, each time subdivision happens, we want to use new prefab after each subdivision:



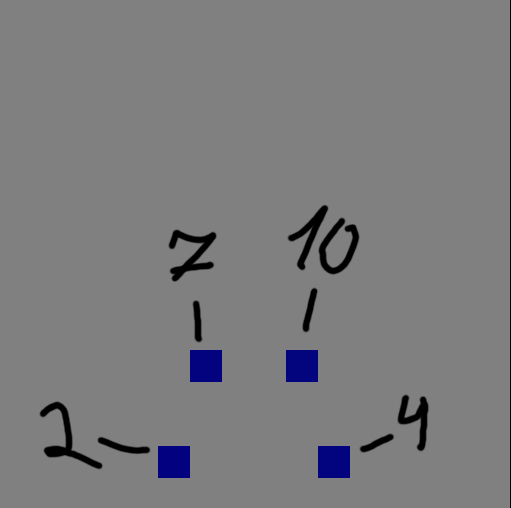
If we don’t use additional textures, “subdivision” frames would end up being blank in game, because we destroy unsubdivided object before we instantiate its subdivided parts. To fix it, we reference additional textures each frame before subdivision happens.

We need 2 additional textures (since every time an object gets subdivided, it subdivides in to 2 objects):

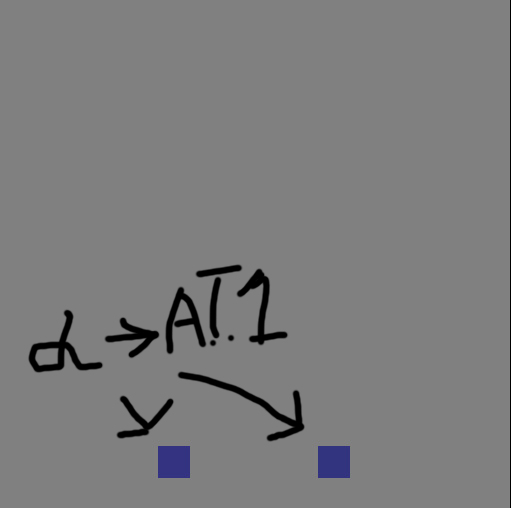
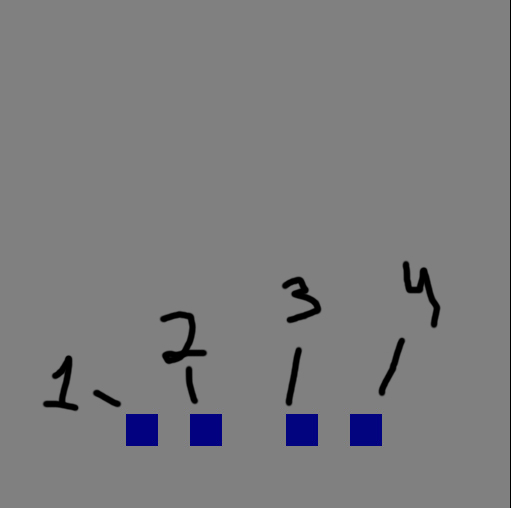
Additional Texture 1 (A.T. 1):

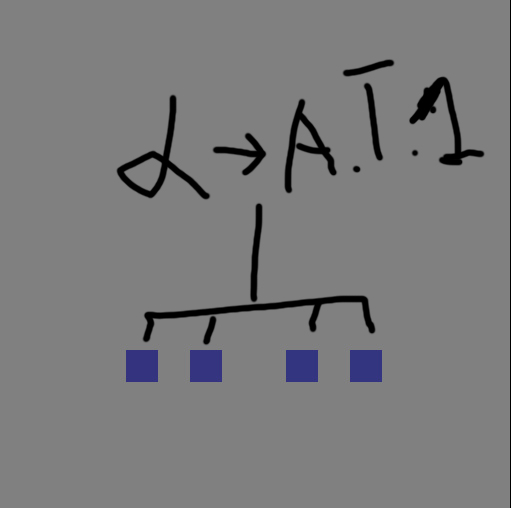
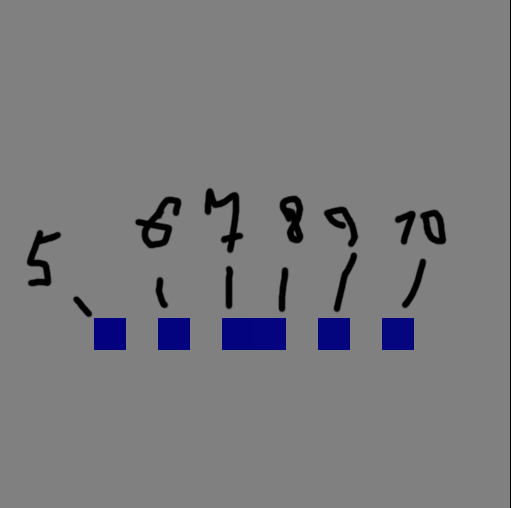


Additional Texture 2 (A. T. 2):



And our main textures (at frames of subdivision):

 -> 

 -> Texture baker

## Textures import settings.

When you import your textures to unity you need to set correct import settings:

1. Non-Power of 2 - None
2. Read/Write Enabled – true
3. Filter Mode – Point (no filter)
4. Format – RGBA 32 bit

All other settings may stay as they are.

## Converting textures to a projectile animation asset

Go to Window -> Projectile Animator -> Texture Reader

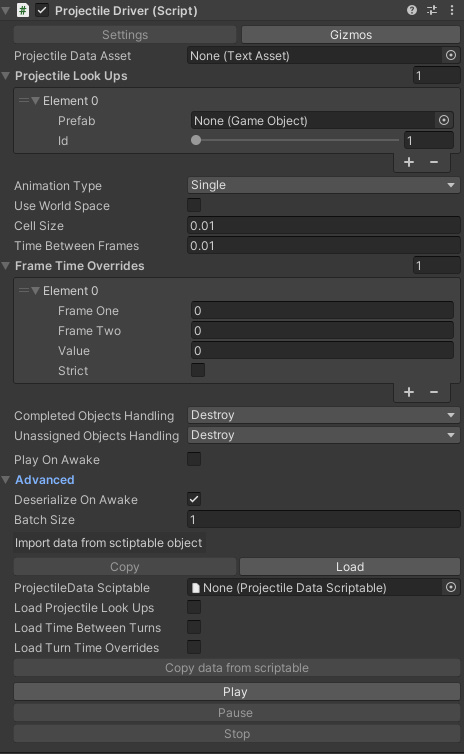
Drag textures to the middle column, if you need, you can reorder them like you want, On the right, you can set your textures center (for example, if you want your center to be the middle of your 9\*9 texture, you would type 4, 4).

If you want to use additional textures, check the corresponding toggle. **If the toggle is not checked, your additional textures, if present, will be processed like regular ones.** Order, in which your additional textures are placed does not matter.

Then press “Bake” button and decide where you want to save the file. File is saved as .pdb.txt. You should treat it like your usual text asset. When you are done, the textures are no longer required for this file to work, so you can delete them, if you want.

# Working with Projectile Driver in editor

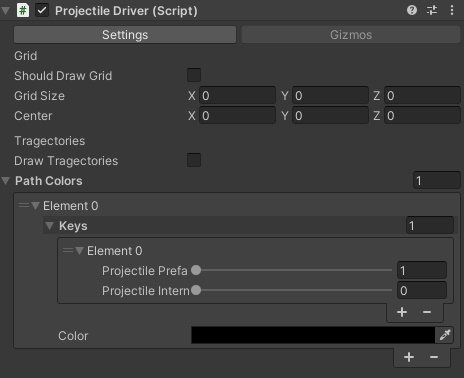
## Settings



From top to bottom:

1. Projectile data asset: here you can drag text asset, that you created following previous paragraph.
2. Projectile Look Ups: this is a list, where script will find what prefabs use for each prefab id (corresponds with red color).
3. Animation type: how do you want your animation to run. Single: run one time then stop; Repeat: run from beginning to end, then restart; Ping-pong: run to the end and then run backwards.
4. Use World Space: do you want to transfer positions coded in textures to local space or world space?
5. Ratio of converting distance between to pixels to local/world units.
6. Time between frames: default time, it takes to interpolate between current and next frame.
7. Frame time Overrides: you can add different time for interpolation between certain frames. If set to strict, works only when frame one is changed by frame two, otherwise works also if frame one changes frame two.
8. Completed Objects Handling: how to handle objects, that have completed the animation (for example, if Stop() was called or animation with single AnimationType has completed). Destroy – destroys instantiated objects. Ignore – leaves them at their last know position.
9. How to handle objects that are not present on the next frame?
10. Play On Awake
11. Deserialize On Awake: if selected, deserializes provided text asset on Awake. **If you use Mono as your scripting backend,** i**t’s recommended that you deserialize on awake at least in ONE of instances of Projectile driver, because first deserialization takes much more time than consequent ones (even on ALL other projectile drivers, that will work fast even if they didn’t deserialize anything themselves).**
12. Batch size: the batch size used by IParallelJobFor.Schedule(). The more projectiles you have, the higher you want to set this number. Experiment for optimal performance.
13. Import data from ProjectileDataScriptable: here you can drag a projectile data scriptable file, which stores different settings. Then choose types of data that you want to copy from scriptable and press “Load data from scriptable” button. Desired data will be copied. The scriptable itself **is not saved to the projectile driver.**
14. If you want to load data from scriptable object at runtime, switch to “Load data from scriptable object” and follow the steps described above.
15. Play: calls Play(). Works in edit mode.
16. Pause: calls Pause(). Works in edit mode.
17. Stop: calls Stop(). Works in edit mode.

## Gizmos



From top to bottom:

1. Should draw grid: check if you want to draw 3d grid. Draws in world space if Use World Space is checked (in settings), uses local space otherwise.
2. Grid size
3. Center
4. Draw trajectories: Draws paths for each projectile from loaded text asset.
5. Path Colors: you can assign a unique color to each projectile key, or assign same color for multiple projectile keys

# Working with Projectile Driver in code

## Passive variables

These variables can be changed while animation is running, but changes will take effect **only when you restart the animation.**

1. ProjectileLookUps (though if there are new projectiles being instantiated during the animation, they will be instantiated with new prefabs)

## Active variables

Anything else can be changed even while animation is running, and it will take effect immediately.

If you want to change animation type in inspector in play mode while animation is running, pause the application first, otherwise it may cause an error (does not occur if you change it though code or in edit mode).

## Setting asset through script.

When you want to set new asset call ChangeAsset(YourNewAsset.text).

You can also use already deserialized data ChangeAsset(YourDeserializedData).

If you want to use very big asset, you can DeserializeAssetToQueue(string). This will run deserialization on a separate thread, and when it’s done, add deserialized data to DeserializedFrameDatas.

## Controlling animation

Play/unpause animation with Play(). You can also play from certain frame Play(int frame, int order: 1 to play forward, -1 to play backwards).

Pause: Pause(). When animation is paused **changing passive variable will still have no effect.**

Stop: Stop(). Stops animation, applying CompletedObjectsHandling to each active projectile, driven by this animation.

# Serialization

Serialization and deserialization are done with FrameDataSerializer static class.

## Serialization

You can serialize data with SerializeFrameData ().

You can save your data with SaveFrameData().

## Deserialization

You can deserialize data with DeserializeFrameData().