Table of Contents

The easiest way to start modding is to use existing scene asset for monkey plugin. BetterScene can detect monkey presence and expose assets it can load as scene packages. This tutorial is based on TheFlatV2 Monkey asset by Manniakk.

BetterScene internally called CustomScene, CS for short. CustomScene and BetterScene are interchangeable names.

Creating new modification package using monkey asset

Create new directory in Packages. Directory name is not important for loader, but it is good idea to use mod id as name. Recommended mod id should contain author and actual id: <author>.<id>.

Next, you need to create package manifest manifest.json. Minimal manifest may look like this:

```
{
	"type": "story",
	"id": "<your mod id>",
	"name": "<your mod display name>",
	“author”: “<author’s name>”,
	"version": "1.0.0",
	"require":{
		"monkey.the_flat_v2_evening":"0.0.0",
		"basic_poses": "1.0"
	},
	"plugins":{
```

```
            "com.thora.monkey":"1.8.7"
        },
        "mainScene": "myscene"
}
```

Package *type* can have following values:

- asset – generic asset package that can be referenced by other packages. Use-cases: graphics assets, poses, etc
- extension – this type of package allows to expand contents of specific packages. Use-cases: additional poses for postures provided by asset package; replacing content
- story – package that can be selected from menu and started

Plugins section is used to declare BepInEx dependencies. In this case package require monkey.dll, to load monkey assets.

Require section allows to declare package dependencies. Dependencies are mandatory. In this example modification requires to load package with basic poses and actual monkey asset.

BetterScene uses the following pattern for monkey mod id: monkey.<assetfilename>. For example: “theturtle.unity3d” file will be exposed as “monkey.theturtle”.

mainScene parameter controls scene id to load when story starts.

Finally, you need to define scene file myscene.scene with following content:

```
{
	"include_before":[
		"monkey"
	],
	"sequence":[
		{ "type": "destroy_go", "name":"Eviroment (2)/Staticos/Officina1 Variante/WallLogic/Statics/TranslacionTargets/"},
		{ "type": "destroy_go", "name":"Eviroment (2)/Staticos/Officina1 Variante/DeskLogic/Statics/TranslacionTargets/"}
	],

}
```

Scene file is a sequence of operations required to load final scene.

- *include_before* allows to load additional scene files before main sequence. In this case it is monkey.scene (generated for monkey asset and provided by monkey.* package).
- *sequence* parameter contains list of scene loading operators. In this case it purges vanilla GoTo targets because they have no sense for custom scene.

During scene loading BetterScene automatically cleans up references to missing GoTos to prevent crashing.

Now story can be launched from Custom Story Menu.

Troubleshooting:

main log is BepinEx/LogOutput.log

- Package is not listed – see logs for json related errors
- Loaded office instead of custom scene – see logs for scene loading errors
  - wrong mainScene id
  - json errors in scene file
  - some other errors
- Package listed but can not be launched – see errors info in package description window
  - missing dependency
  - wrong monkey package id: see logs for available package ids

Setting up scene

BetterScene defines 4 object types to describe scene:

- POI – point of interest. A place character can go to. Examples: Desk, Desk Side, Wall
- Posture – primary character pose. Other poses are defined in relation to character's posture. Examples: Sit, Stand, Bend, Lie.
- Pose Clips – exact poses available for character at specific posture
- POI Posture – exact posture implementation in scene. Examples: to *Sit* on *Table* at *Kitchen*. Where Sit is posture, Kitchen – POI, and Table – is exact point this pose available at.

Example situation:

Waving hand while sitting on kitchen table.

Kitchen is POI
Sitting on table is posture
sitting on kitchen table is POI posture
Waving hand while sitting on table is Pose Clip

Available POIs will be listed in GoTo radial menu.
Available poses and postures – Poses menu.

Implementation details:

- POI – defined as position and rotation in world space
- Posture – bone rotations, hips offset and root offset. Used as template only
- POI Posture – bone rotations, hips offset and root offset. This disposition is used as default idle pose when no Idle clip provided
- Pose Clip – collection of bone rotations and hips offsets. Root offset is applied from POI posture data

Scene Tool Overview

Every scene object is altered from Custom Pose view. BetterScene uses vanilla pose editor that has own specifics. Because of that it is recommended to use “willing” character to edit scene, otherwise there may be problems with bone movement.

Switch character to Custom Pose mode. You will see Additional window on the left side called “Scene Tool”. This window is the main toolbar for every BetterScene related thing.

Bones tool is posing helper tool.

When you select bone, you can see it’s name, coordinates, and use some buttons. Brief description of this buttons:

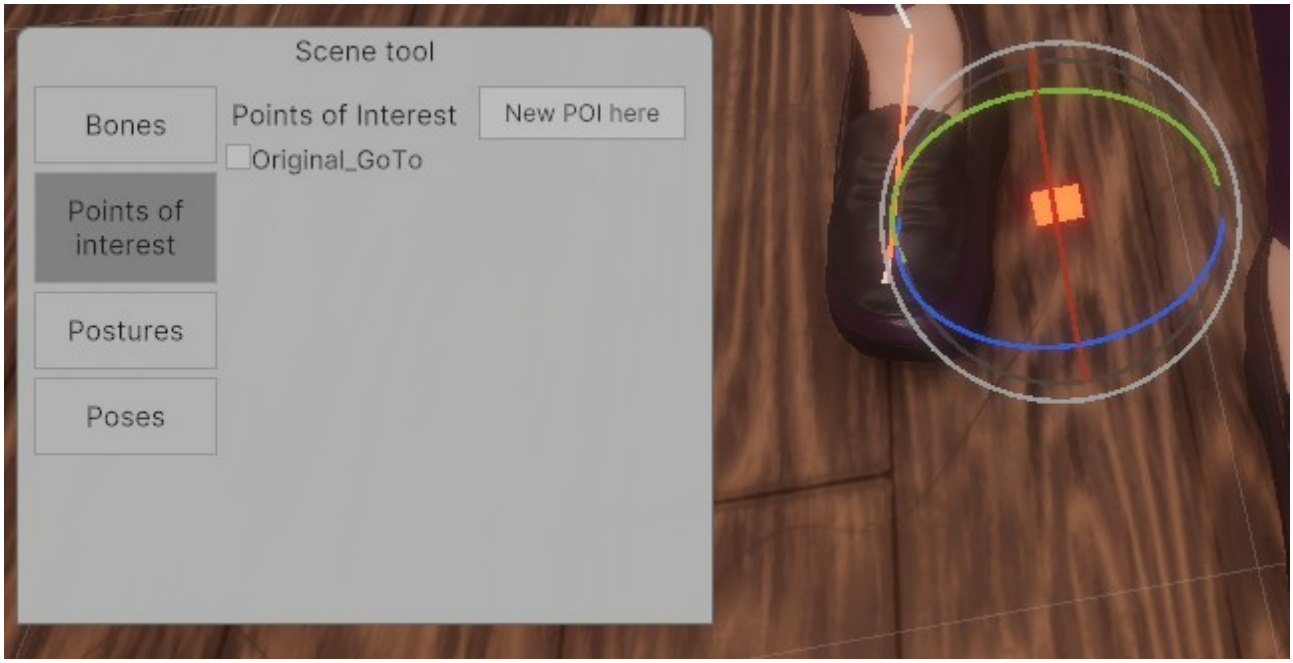
- L<>R – switch between left and right bone
- ASC – switch to ascending bone, aka parent bone
- DSC – switch to descending bone, aka child bone (single child bone only)
- FWD/BWD – add or subtract euler rotation from Rel.Rot. Field
- SYN – synchronize opposite bone’s rotation, aka synchronize direction
- MIR – mirror rotation. Same as SYN but with mirror symmetry.
- TSYN/TMIR – Tree[SYN/MIR]. Applied to selected bone and it’s subtree, aka sync limb.
- TCPY/TPST – copy/paste subtree rotations. Useful when creating multipose clips to transfer limb orientation.
- Hips locked/unlocked – allows to move(but not rotate) hip bone without dragging all hierarchy.

Other tool windows share same structure: they have “New” button to create new entity and list of objects. Each row contains object id and control button. That button can be used to perform object specific actions like “apply”, “edit” or “remove”.

Creating POI

To start, create new POI at Sofa. Switch character to custom pose mode.

Drag root bone to desired position and choose proper rotation. Orientation is important, it will be used as main frontal orientation(turn around action). As rule of thumb, by default character should be facing player in front of object of interest.



Then press [new POI here] and type POI ID. In this case Sofa.

> ID must be Latin alphanumeric, without spaces and special characters. There are no validations. Just don't.
>
> *POI Id is not a name*. It is identifier to reference this POI

Next, click on button near created POI and select "Edit descriptor".

Type "Sofa" in Display Name and press Save.

> Descriptors – are additional data associated with object. Descriptors are designed to be human readable json files.
>
> When Display Name is not specified, CS will use object ID as its name. It may work in case of Sofa, but it won't work in other cases, like Original_GoTo on picture above. It is recommended to assign proper Display Name for every entity.

Additionally, POI can have "parent POI":
When Kitchen_Table has Kitchen as parent POI, you can go to Kitchen_Table only from kitchen. This allows to reduce go to list.

This use case scenario will be covered later.

Creating POI Posture

BetterScene is bundled with basic poses package. It contains some poses from original game, and postures to be used with BetterScene.

Postures aren’t poses. Each posture is made with some assumptions in mind. That assumptions are related to poses posture hosts. From this perspective posture can be seen as “animation host” that contains poses available at some specific object of interest. Main role of postures is “create once apply everywhere”. This way you can create postures for common objects of interest, and just implement them at various POIs by editing root offset.

Let’s teach character to sit on sofa. Basic poses contains 2 to postures related to sitting. Both Sit posture and Climb implement different kinds on sitting pose.

Sit posture describes sitting on a small horizontal surface with very little space. Best example is chair. Character can sit, but cannot do anything else.

Climb posture is sitting on some relatively large horizontal surface. Example is bed, table, some couches. This posture suits best for scene’s sofa.

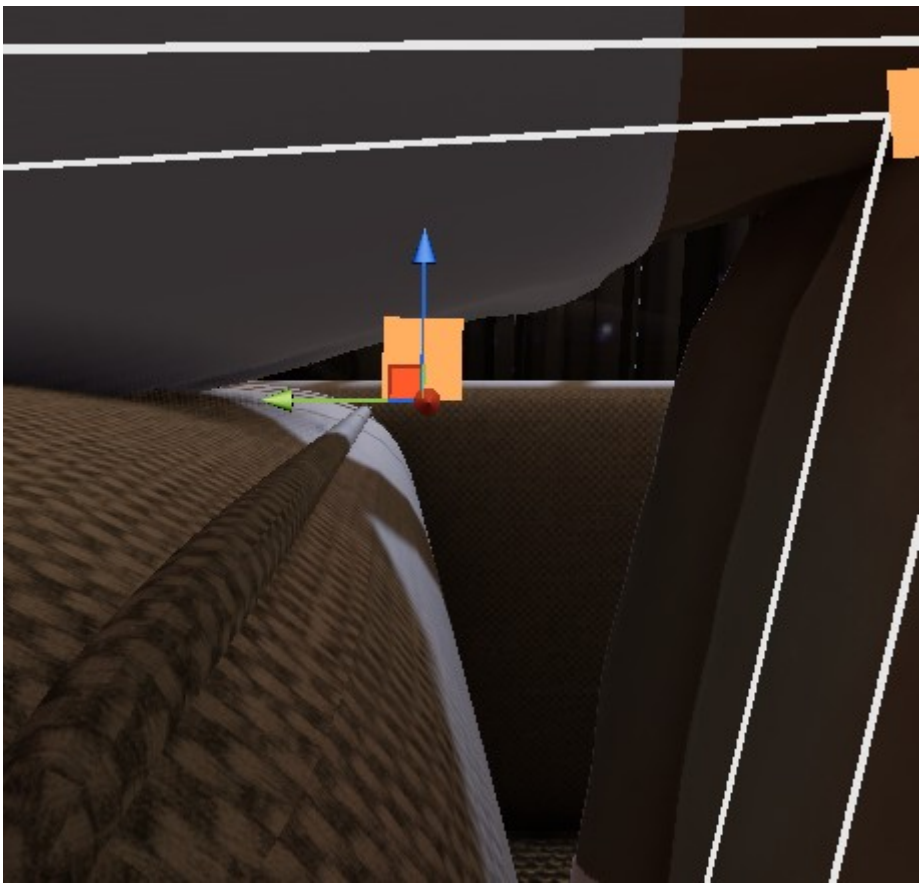
Climb posture itself is partial, it does not implement some poses related to low hanging legs due to different heights of main surface. There are two variants of climb posture: ClimbHigh – when target surface is closer to pelvis level, and ClimbLow – when it’s closer to knees. The difference is special smooth transition sequence when climbing high, and different legs angle when climbing low.

First of all switch game into custom pose mode, and perform “GoTo” action on Sofa POI. This action will teleport character to Sofa POI. Game detects current POI, but sometimes it may work wrong.

Then switch to Postures tool and “Apply” ClimbLow posture. After that action it is expected that character will be sitting in the air. Select root bone and drag it to edge of sofa surface.

Drag blue axis to align rootbone with sofa surface and green axis to match surface edge. Avoid dragging red axis, to prevent shifting during pose transition.

When dragging blue axis, keep an eye on body position. It is bad sign when puppet does not follow bones, it will end up with armature trembling and probably destabilize every pose in posture.



After you are fine with character's position, use "Implement" action of ClimbLow posture.

In the window you need to type missing part of ID. POI posture ID use this format: [Posture].[POI].[object of interest]. You can type here "Sofa".

> In most cases, when POI is bound to object of interest, you will end up with IDs like Sit.Table.Table. This in expected. The purpose of third part is to allow multiple targets at single POI, like Sit.Kitchen.Table/Sit.Kitchen.Sink or Sit.Sofa.Left/Sit.Sofa.Center.

Then exit custom pose mode, and relax character. Now, in poses menu you can see "Sofa" action. Click it. When character sits, its poses menu contains available poses and "Interrupt posture action". In this case it is called "Stand.Sofa".

To fix naming problems, transfer character to custom pose, and open "Edit descriptor" of new ClimbLow.Sofa.Sofa.

In this context, Display name is used for "Apply posture" action. You can type here "Sit on sofa".

Cancellation name is a "Cancel posture". You can type here "Stand up".

Orientation field is used for scenarios that require specific character orientation. Examples: Vanilla that provide different poses depending on character orientation.

Parent posture allows to nest postures. Nested posture can be applied only when character has parent posture. Example: Lie posture that require to Sit first.

Creating Poses

Basic concept of posture poses

BetterScene's posing system is custom pose based, but uses own plain-json data format. In CS poses are called Pose Animation Clips (PACs) and may contain one or more classic poses that may be played sequentially.

Vanilla custom-poses are relative to current character position. This approach makes them difficult to reuse in different contexts: a) you need precise position of character relative to object other than floor; b) you need to retarget each pose to match new environment.

Because of this reasons, all PACs are relative to *Posture*. It allows to retarget *Posture* via *POI Posture* to use existing poses in new places. Because of that, you should not modify root bone's location in PACs – all poses in posture must share same root.

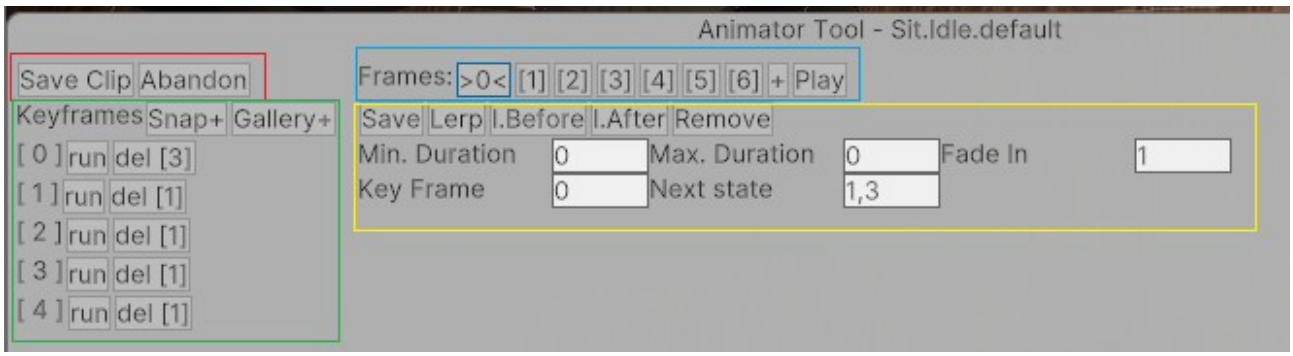
Pose editor

To cover most features of PAC, the best approach is to recreate Sit.Idle pose. This clips uses all features of editor.

First of all create new POI at chair and implement Sit posture there. You may notice that Sit uses different root location compared to Climb. In this case root bone points to the end of "Chair surface". This way characters of different heights and sizes won't try to push their spines into chairback.

This pose has few important features: it is animated and it has hand pinning.

Switch to custom pose and in Poses tool use “Load” of Sit.Idle.default. You will get additional animator tool window. Press “[0]” button in “Frames” section.



Red section is clip control buttons. Save clip – saves. Abandon – discards changes. Both buttons will close tool.

Green section is key frames control.

- Snap – make snapshot of current pose, save it as new keyframe, and assign this keyframe to current timeline frame
- Gallery – select pose from vanilla pose gallery, and apply its rotations to current pose
- run – apply keyframe to pose
- del [N] – delete keyrfame. N is count of timeline frames that reference this particular keyframe. You cannot delete keyframes that have references.

Yellow section is frame settings. There may be multiple frames that refer to same keyframe.

- Save – saves fields data
- Lerp – interpolates current pose as average between previous and next frame. Does not create keyframe on its own. It is useful when you don’t like how poses blend and want to tune it.
- I.Before, I.After – Insert copy of current frame before or after. Auto-adjusts cross frame references.
- Remove – remove selected frame. Auto-adjusts cross frame references.
- Min/Max Duration – how long to display blended frame before moving forward. Min/Max allows to induce some randomness into animation flow. [seconds]
- Fade In – how long does it take to blend from previous frame[seconds]
- Keyframe – index of keyframe to show in this frame
- Next state – comma separated list of next frame indexes. May be empty. Randomized when multiple indices.

Blue section is frame timeline. Number buttons allow to select frame(and auto save fields changes). “+” inserts new frame. Play – plays frame sequence starting from selection. Note: results may differ vary between editor’s player and in-game player.

All changes to pose are saved into temporary clip at {scenepackage}/editor/animator.{Posture}.epac file. This allows to exit custom pose mode without losing work.

Brief guide to posing

Abandon clip, and relax character.

Enter custom pose mode. At Chair *POI* implement *ClimbLow* as MyChair.

Apply this posture to character, and in Poses tool press “New Pose Here”.

> *What you see in Poses tool depends on active character Posture.*

You will be asked to type tail of Pose Id. Pose Id consists of 3 parts: PostureId.ClipName.Variant.

- PostureId – restricts clip to specific posture
- ClipName – is an identifier of clip in posture
- Variant – is a variant id of [ClipName]. This part allows to create different variants of some clip. At this moment it may be used to provide multiple variants of clip that may be played randomly.

Type “Idle.default” in the window. As default variant of Idle clip.

> *Idle is one of special clip names. Idle is a clip that will be played automatically when posture applied. Other special name is Binding. This hidden clip is generated from posture and used as fallback when Idle clip is absent. Binding is reserved name and should not be used.*

You will get empty animation window with character in binding pose.

Step 1: first frame

Take snapshot of current pose and add first frame.

Now, drag hand bone to its position on leg and adjust elbow angle.

Use Bones/ASC to select clavicle of hand and use Bone/TMIR tool adjust opposite arm pose. You may need to fix character’s symmetry to achieve proper results.

Press Snap to take new snapshot, and remove old keyframe.

Step 2: pinning effector

Because of lack of self collisions, in this pose hand may clip through legs of characters of different complexion. You can use *effector pinning* tool with hand bone, to pin hand to object below it.

> *In this case, effector is a bone in the end of IK chain. You can change its position, and IK will solve other bones orientations.*

To use this tool, select hand bone, and press “PIN” in Bones tab. If operation was successful, button will change name to “UNPIN”. In not, you will get error. Pinning is not stable or properly implemented, so you may try to change angle and position of hand to make it work properly.

When effector is pinned, it will follow its pin point using IK. Quality of this action may vary.

Perform pinning of other hand and take snapshot. IK pinning will be stored in keyfame.

> *Effector pinning is implemented as following of point in target collider’s local space preserving relative position and relative angle.*

Step 3: key poses

Now, when base pose is ready, you need to create another two frames, for each leg pose.

Use I.After to generate two more frames. Then select frame 1, setup leg pose, and press snap. Configure pose for second leg at frame 2.

Loop sequence by setting next frame = 0 at frame 2 and press play.

You may notice that transition between frame 1 and 2 is incorrect, and you need to add intermediate frame with base pose. Stop playback, select frame 3 and press I.Before. Assign Key Frame 0 (index of base keyframe) to frame 2.

Press playback again to see if it is playing correctly.

At this point you may set Min/Max duration and Fade In for frames. Use play button to see results.

Step 4: intermediate frame blending

This loop has flaw(s). There is a clipping problem, due to linear pose blending. To mitigate this, you can create intermediate frames that avoid clipping by editing motion path.

Select frame 0 and press I.After. Then interpolate pose with Lerp. Now, you can fix clipping by adjusting leg rotation so it won't collide with leg below. For example, you can raise it higher. Use snap to save changes and test results.

You may repeat this for other frames.

Step 5: save clip

When sequence is completed, you can use "Save clip" option to save clip. Then reset character, and use Poses→MyChair pose to test your animated pose.

Advanced Scene

Creating POI Hierarchy

Radial menu has item limit. When you get too many POIs you can start to build navigation hierarchy.

How does it work? When GoTo list generated it allows to travel to POIs without parent and POIs whose parent is current POI.

Lets arrange two groups of POIs: living room and kitchen.

Create two new POIs: Sink and Oven. (Location is not important, they will not have any future use).

Edit Oven's descriptor, and set Sink as parent POI. Save.

> *There is no way to give poi 2 names: "Kitchen" at "Livingroom", but "Sink" at "Kitchen"*

Rename Sink to Kitchen. Send character to Sofa.

> *If you have animate GoTo enabled:*
>
> *Now you may notice that character walks through furniture. Animator will use level's NavMesh if possible. When NavMesh is absent – it will travel through space.*

When character at Sofa, Oven won't be present in GoTo list.

Edit Chair's descriptor and make it child of Sofa. Now, send character back to Kitchen.

New GoTo list will contains OriginalGoto, Oven And Sofa.

> *You can rename Original_GoTo! Just edit it's descriptor. You cannot remove it, to keep things simple, but you can name it.*

Send character to Sofa to see Chair POI available.

Creating posture hierarchy

It was mentioned earlier that postures may have parent. This task may have artistic reasons, but most important are technical ones. To illustrate problem, send character to Sofa POI.

Implement Lie posture on sofa, and rename it to Lie. Later run Lie action to see how it goes.

To avoid this unnatural rotation, you can forbid to Lie without Sitting first. But first, perform Sit, and then Lie. It looks, slightly better.

Now, edit Lie.Sofa posture and set its parent to ClimbLow. After this action Lie pose will be available only from Sit.

> *As you can see, when multiple postures available at POI, you can switch between them through Poses menu.*

But switching from Sit(ClimbLow) to Lie is not very smooth yet.

Transition clips

You can set additional clips that playing when character switches between different poses.

Now you need to create clip that smoothly blends between sitting and lying pose.

Start editor and send character to Lie.Sofa posture.

Create new pose FromSit.

To start you need 2 frames, sitting and lying. To get lying, just snap current pose.

Getting sitting is more problematic, because you cannot use actual posture, due to different root offset.

Go to Postures menu and Apply ClimbLow posture(Not POI posture). Character will change its pose.

If due to bug character’s root bone will move, perform snap and press run, to fix root bone.

Now, use hip bone to approximately achieve expected sitting posture and press snap.

When you have both poses, you can create frame sequence: 0 – Sit, 1 – Lie.

Create intermediate frame through Lerp’ing and try to make this transition more smooth.

You can generate more frames and adjust timings here.

In the end, make sure that last frame has empty next frame (so animation is one shot) there may be bugs. Save clip and open its descriptor.

Change clip type to “transition”.

> *Transition clips are multipurpose clips. They can be used for various purposes. They can smooth transitions between poses; create transitions inside subposes, etc.*

And add new supports transition entry. UI does not support posture transitions, so select From and To to Idle(to have fields in descriptor file) and check Reversible flag. Save everything and shutdown game. Go to {yourpackage}/poses and edit Lie.FromSit.pad.

Set “From” field to “ClimbLow”, and set “To” to “Lie”. Save and start game.

Now, when you Lie from Sit, it should play new clip.

Also, you may notice that it is not easy task to switch between different anchors, and this should be avoided.

When creating transition entry, you’ve raised Reversible flag. This flag hints system that this clip may be played in reverse when reverse transition occurs.

Sub-poses and pose sequences

If you look at Climb posture, you can see that there are contextual poses. This mechanic is implemented via subpose type.

Subposes are configured like transitions, except you fill only From field. And type display name for this transition.

It is good idea to have some naming convention to subposes, but there is no simple answer.

Lets create sequence of 3 poses to rise a hand. They will be named RaiseHand0,1,2.

Keep in mind that hand will be raised in 3 steps and send character to standing posture.

> Stand posture is special, because it is not exactly posture. At least in Idle, Stand posture is running using native animator. To enter Stand.Idle you need to interrupt custom pose mode, and as result it will shut down editor. Use game menu to get character Standing and return to custom pose mode.

For this kind of actions it is wise to create new posture to reduce list of action.

Step 1: create container posture

Create RaiseHand posture using standing pose and implement at Sofa POI.

Step 2: create first clip

Activate this posture and create clip RaiseHand0. Raise hand to 45 degrees. Save this clip.

Test results.

Step 3: create second clip

Create clip RaiseHand1. Use load action of Clip RaiseHand0 to import keyframes from previous clip. Raise hand higher and save clip.

Check results.

Step 4: create third clip

Open clip **RaiseHand1** using Load action. Raise hand higher, snap, and save this clip as **RaiseHand2**.

Check results.

Step 5: gluing clips together

~~You may see that “Sit on sofa” is present in Poses list. By default you can switch between postures on the fly. In this case you may want to hide Sitting from menu. To achieve that, set Parent Posture of ClimbLow.Sofa to Stand.~~ Does not work due to bug.

Open RaiseHand1 descriptor and change it to subpose. Add transition from RaiseHand0 and call it Higher.

Open RaiseHand2 descriptor and change it to subpose. Add transition from RaiseHand1 and call it Higher again.

Open RaiseHand0 descriptor and change display name to “Raise hand” and cancellation to “Relax”.

Test results.

Step 6: if you want to loop

You cannot directly add link from RaiseHand2 to RaiseHand0, because To operation is not implemented for subposes. And you cannot declare From link on RaiseHand0 because it has pose type.

To mitigate this, create new clip RaiseHand2_0. Save it empty, without frames.

Open descriptor, change display name to “And again”, change type to transition.

Add transition entry that maps it From RaiseHand2 to RaiseHand0. Save.

Test results.

Partial postures

Both *Posture* and *POI Posture* can have *Parent Posture* in descriptor. When you assign parent posture to POI Posture, you restrict transition. Simple posture behaves differently in this case.

When *Parent Posture* assigned to *Posture* it means that that posture also includes all poses from parent posture. ClimbLow/ClimbHigh implemented this way. They have Climb posture as parent, and share its poses.

Understanding packages

Each package consists of manifest and files. When story package is prepared to be loaded all dependencies files are merged into single filesystem.

Files search order is top-bottom: story, story-deps, stopy-deps-deps.

It means that story can override files from dependencies if their paths match.

BetterStory itself loads assets by its extension and filename, ignoring full path. In this scenario file name acts as identifier. Other components may behave differently.

This allows to regroup files in directories, but duplicate files may lead to errors.

Most scene entities will have entity file, and descriptor file. Entity file contains machine data, like coordinates, vectors, etc. Descriptor contains data that may have sense for unprepared reader.

All package dependencies are mandatory.

Optional dependencies called extensions, and may be selected by user before launch. Extensions cannot have dependencies due to potential of complicated dependency graph.

Packages may be bundled as zip files. In this case manifest should be in the zip's root.

Scene loader

BetterScene provides scene loader to introduce custom environment. Right now the only way to load scene is to reference it from story package manifest.

System uses scene files to load custom scenes. You can reference any *.bundle files to load assets from. Bundle files are standard Unity3d asset bundles. It is seems that game uses Unity 2021.2.7, and it is a recommended version, but other versions may work too.

When loader finishes interpretation of operations from scene file, if required, it will invoke light probes tetrahedralization. This operation will merge all scenes light probes into single grid. The unity build used by game does not merge properly overlapping light probes, and may produce dark unlit seams. It is recommended to avoid overlapping scenes with light probes and unload original scene.

Scene files

Scene file contains list of operations to load scene assets and configure them. It has following format:

```
{
          "include_before": ['scene1','scene2'],
          "include_after": ['scene1','scene2'],
          "sequence": [{operation},...],
          "interview_sequence": [{operation},...],
          "on_scene_load":{
                "unity_scene_name1":[{operation},…],
                "unity_scene_name2":[{operation},…],
          },
}
```

include_before/include_after allows to reference other scene files before or after this scene file. Primarily used to reference monkey scene to load monkey asset.

Sequence – is a sequence of operations to run when scene loaded first time.

Interview_sequence – this operations are executed when interview scene is loaded.

on_scene_load map allows to set sequence of operations to run when specific scene is loaded. Note: it won't execute them if scene is already loaded during scene file execution. This may be changed.

Generic operation object looks like this:

```
{
  "type": "operation type", "name": "referenced name",
  "target": "target name", "value": "any object",
  "disabled": false
}
```

Standard operation types

```
{
  "type": "load_scene",
  "name": "bundlename:/Path/To/Asset/file",
}
```

This operation loads scene from bundlename.bundle file. You need to provide complete path to Unity asset without ".unity" extension. Loader will log available paths if it fails to find provided name.

```
{
  "type": "unload_scene",
  "name": "SceneName",
}
```

Unload scene operator will unload Unity scene by its name. Additionally, this operation will cleanup invalid GoTo points that reference destroyed transforms.

```
{
  "type": "apply_volume_profile",
  "name": "Path/To/Volume/GO",
}
```

This operation allows to replace game's volume profiles with scene provided Volume Profile. It is expected that referenced GameObject contains Volume component. Its profile will be copied to existing in game camera volumes to achieve rendering results similar to editor. This feature does not work correctly with hand cameras. This operation has deprecated alias apply_hdrp_profile.

```
{
  "type": "dont_destroy",
  "name": "Path/To/GameObject",
}
```

Don't destroy operation moves referenced game object to HideAndDontSave scene. Main purpose of this operation is to keep Original_GoTo transform from destruction when unloading office scene.

```
{
  “type”: “load_prefab”,
  “name”: “bundlename:/Path/To/Asset/file”,
}
```
Instantiates asset from asset bundle. This operation is a placeholder, and is not implemented correctly to be used.

```
{
  “type”: “disable_go”,
  “name”: “Path/To/GameObject”,
}
```
Allows to disable GameObject by its name.

```
{
  “type”: “destroy_go”,
  “name”: “Path/To/GameObject”,
}
```
Allows to destroy GameObject

```
{
  “type”: “set_value”,
  “name”: “/Path/To/Game/Object/@ComponentType/field”,
  “value”: “123”
}
```
Set value operator allows to assign values to field and properties of component referenced by *name* field. The value field must provide objects value in json form.

Component path is interpreted this way:

- last path element is field/property name, other elements are component path
- when last component path starts with ‘@’ it is interpreted as target component class name. Otherwise it is GameObject’s field.
- Other path elements are treated as game object path

When target field found, its type is used to parse value field via JSON to object mapper. For example: string value: “test”; number value: 5; bool value: true; Vector3 value: {x: 1, y: 2, z: 3}.

End User Notes

Player cannot interact with character while internal transitions are in progress.

During first seconds of game, it is required to start/stop custom pose to initialize some game components, otherwise game does not behave properly.