

Motion Matching

User Manual for Unity 2019.4.8f1

This documentation was originally written five years ago for Motion Matching 1.0, so some parts may be outdated or inconsistent. Still, it can be helpful for understanding the system. Please be patient with any such issues.

This translation is from Polish to English, so some language gaps may occur.
All the best!

Caution: I am not the author of this system. The PDF originally existed, but for some reason, the author removed it.

In the YouTube video about this system, many people were struggling to understand it and were asking for documentation.

After a bit of searching, I found this PDF online (from version 1.0). So, I'm uploading it here to help everyone out.
Hope it helps!

2 videos link --- (credit to Michael J)

1. <https://youtu.be/ihzNX3ASQx4?si=d8SwHLxC1l6npazW>
2. <https://youtu.be/vgXL4TIPa28?si=9DMmc7u7BpzrKkOq>

1. Installation:

You should paste the system folder into the project's Assets directory and then install the plugins using the package manager:

1. Burst
2. Jobs
3. Mathematics
4. Collections

2. General overview of the system.

MotionMatching is implemented in a way that makes working with it similar to working with the built-in Mecanim animation system.

Working with the system involves:

1. Preparation of AnimationData – these are assets corresponding to animations
2. Preparing MotionMatchingAnimatorController – this is an asset corresponding to the standard AnimatorController

The next step is to add the component **MotionMatching** in which we place the previously prepared controller and component **TrajectoryMaker** which, based on data collected from the user, creates the trajectory needed for the previous component.

MotionMatchingAnimatorController – has states in which AnimationData is placed. Between each one we can create transitions that occur based on conditions specified in them.

3. AnimationData as an animation equivalent.

AnimationData is a container for the animations from which it was created and for animation frame data (calculated at a set interval) such as:

1. Trajectory – indicates where the animation will move the animated object after a specified time
2. Pose – stores information about selected bones such as their local position and local velocity as Vector3
3. Belonging to the animation section

This data is calculated when AnimationData is created and cannot be changed later (unless the animations are recalculated).



Inspector AnimationData

AnimationData is created using a special editor. After its creation, it is possible to change some of its options.

Description of the AnimationData Inspector:

Basic Options:

Data Type-Animation data type. Currently there are three types:

1. Single Aimation

2. BlendTree

3. AnimationSequence

It is used by the MotionMatching Component to correctly create animations in the AnimationPlayableGraph.

Length-length of animations from which AnimationData is created. In the case of BlendTree, it is the length of the longest animation. In the case of AnimationSequence, it is the length calculated based on the appropriate options (more about this in the description **Motion Matching Data Creator**).

Motion Matching Data Frame Time-time at which the animation was sampled to calculate the data used by MotionMatching.

Motion Matching Data Frame Count-Number of calculated animation frames containing data needed for Motion Matching to work.

Blend It Yourself-MotionMatching is about constantly searching for the best place in a given set of animations. This CheckBox is responsible for the behavior at the following moment: AnimationData "A" is currently being played if in the next search I draw the same animation "A" again and **BlendToYourself** it will be **true** will blend into the animation, otherwise no blending will occur. This option should be unchecked for short looping animations such as walking or running.

Find in Yourself-if this option is unchecked in the next search for the best animation we are sure that we will not find any animation located in AnimationData. This option can be useful in some cases but mainly it should be **true**.

Sections

Each AnimationData has Sections. Sections are time intervals that can be used to mark individual animations. Their use is twofold:

1. We have a long animation lasting several minutes. Part of this animation is walking, and part is running. By creating appropriate sections in AnimationData we can later choose from the code which sections to select animations from.
2. Sections can be used in the appropriate state in the MotionMatchingAnimatorController to define dependencies (More on this in the discussion **SectionDependencies**).

AnimationData has 3 basic sections that cannot be deleted, but can be edited. These sections are:

Never checking-This section specifies frames that will never be searched by the MotionMatching system, even if frames in this section are also found in another section.

Not looking for a new pose-if the current time of the currently playing animation is within the ranges marked by this section, the next search for the animation will not be performed. It can be used, for example, to mark a quick turn animation during a run. If we mark the time in which this turn is performed and this animation is selected at that time, the next search will only occur in the animation time not included in this section.

Always-This is the basic section where the search is performed upon entering a given state.

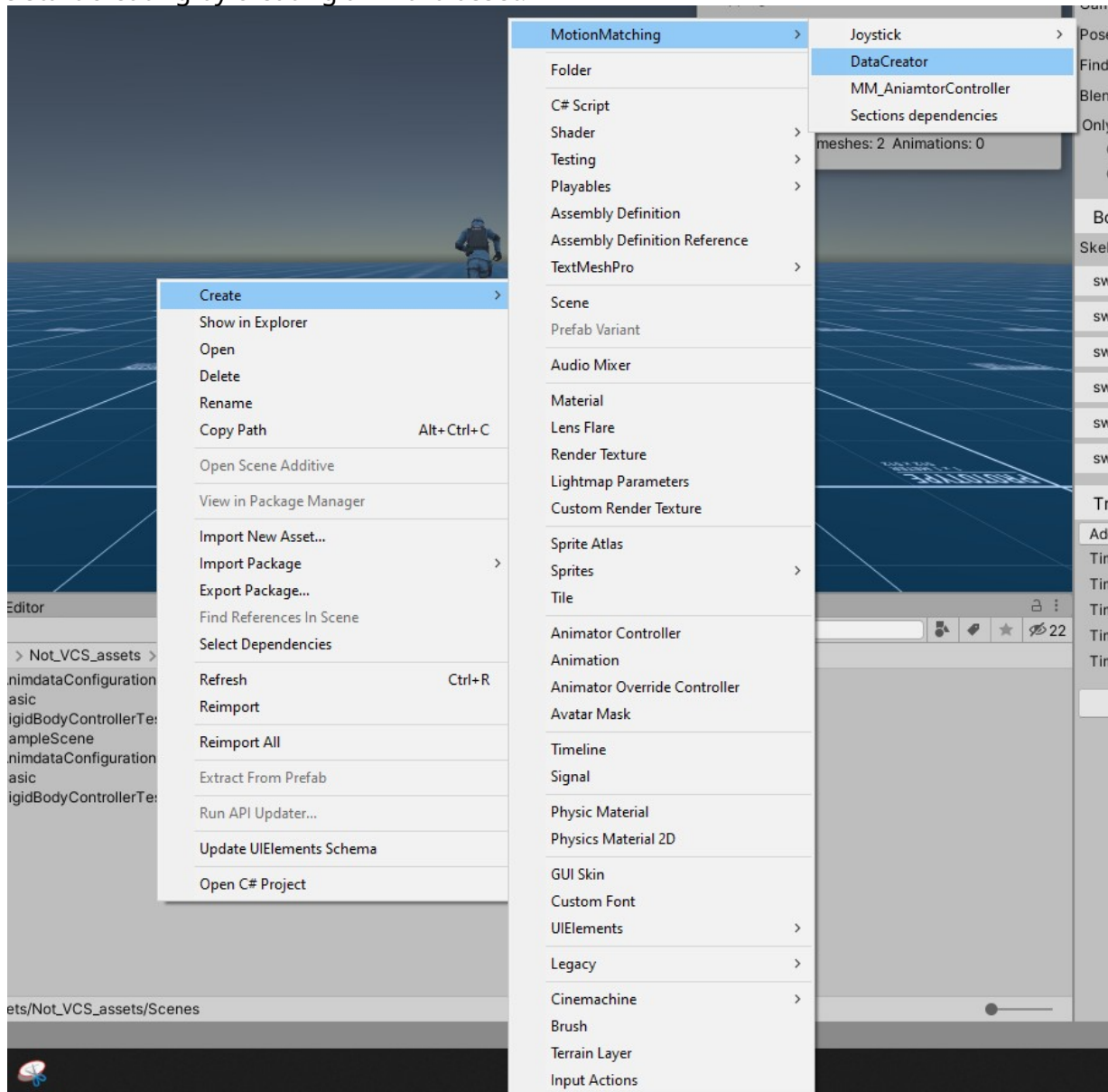
Each AnimationData can have additional 31 sections with arbitrary names. Sections can be added from the inspector. using SectionDependencies and Motion Matching Animator Controller and MotionMatchingDataEditor.

Data type options – information specific to the AnimationData type is displayed here for informational purposes.

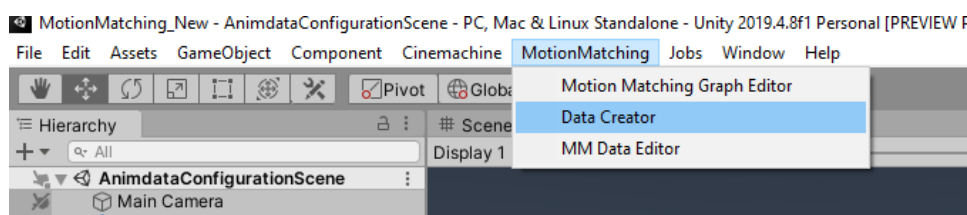
Contact Points – information about the contact points owned by AnimationData is displayed here (we add them using the Motion Matching Data Editor).

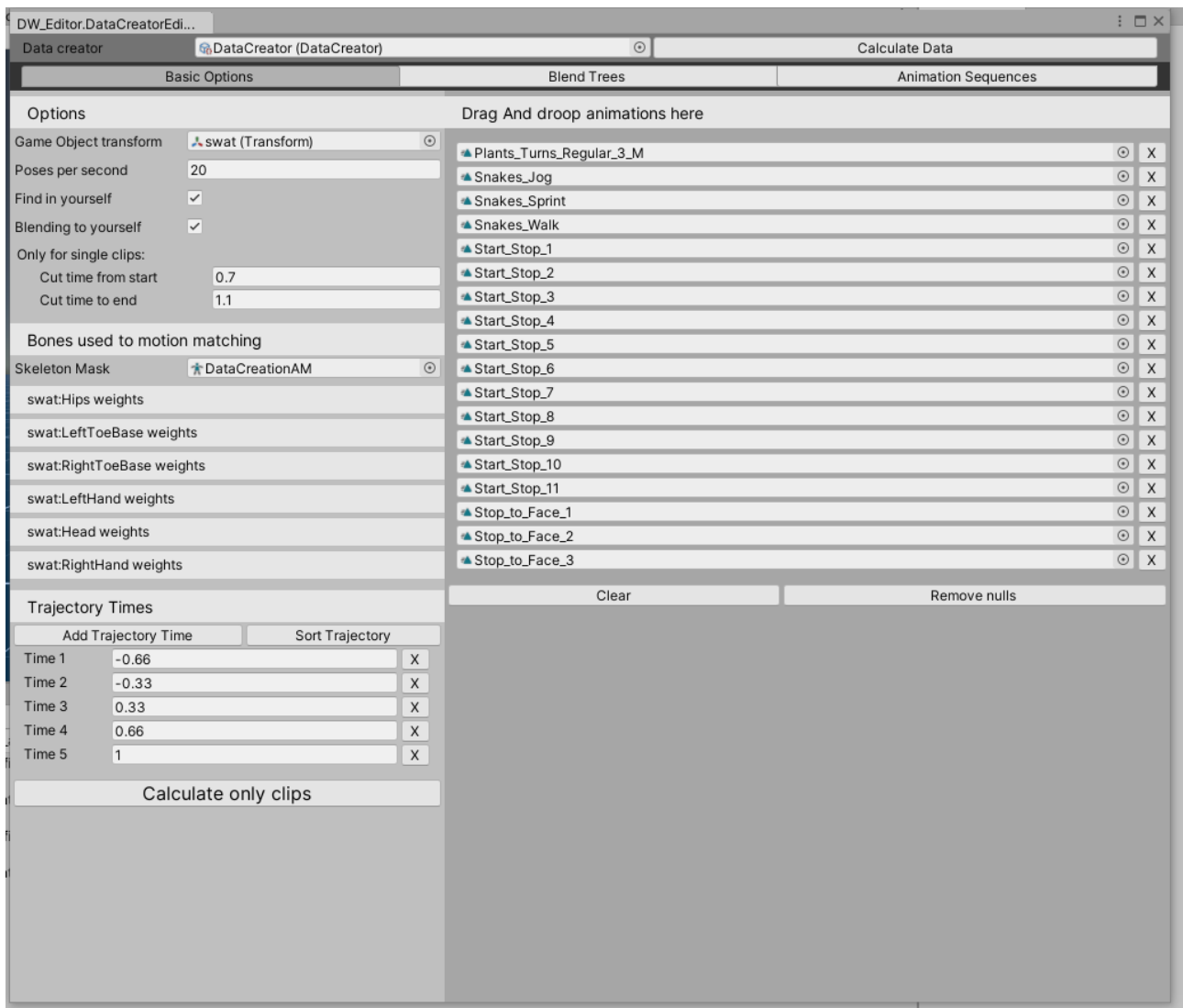
4. Creation **AnimationData**-user manual **Motion Matching Data Creator**

This is an editor that we use to create AnimationData assets. This editor allows us to create 3 types of AnimationData (SingleAnimation, BlendTree, AnimationSequence). We start creating by creating a wizard asset:



Double clicking on the created asset should bring up the Data Creator Editor. If the editor is not created, select Motion Matching from the top menu, then Data Creator Editor.





At the top of the editor there is a field for the currently open editor, and a button that will start calculating and creating all animations. Before starting the calculation, all options must be set correctly.

Below we have three types of editors to choose from:

Basic Options-in this editor menu we set the basic options with which animations are to be calculated. We add animations here, after calculating which AnimationData of type SingleAnimation will be created. We add animations by dropping selected animations on the "Drag and drop animations here" field. There should be long Mocaps and looped "cut clips" here.

On the left side we set the basic options with which the animations are to be calculated.

Game Object transform-animations are calculated by quickly playing them using the Unity AnimationPlayables system. In this field we place the object on which the animations are to be played and the calculations performed.

Poses per second-how many frames of data on which MotionMatching will operate should be created in one second of animation. The more frames, the better the results but the lower the efficiency, this value should be in the range of 10 to 30.

Find it yourself-what should be the default setting for calculated AnimationData.

Blending it yourself-what should be the default setting for calculated AnimationData.

Cut time from start-indicates what time interval should be marked in the Never checking section from the beginning of the animation.

Cut time from start-means what time interval should be marked in the Never checking from the end of Animation section – Cut time from start to the end of animation.

Cut time from start/endshould only be used for long animations from motion capture sessions that are not looped. By setting these options we can be sure that we will not be checking the animation at its beginning and end where the calculated trajectory of frames located at these places may be incorrect.

Bone used for motion matching

Skeleton mask-this is the created Avatar Mask asset with the bones selected from which the pose information will be created in the individual AnimationData frames. The bones selected should not be too many, 5 is enough. The bones of the hand, foot, neck bone and pelvis should be selected. More selected bones do not always give better results.

After placing**Skeleton mask**and**Game Object transform**which has the skeleton on which the mask was made, a list of selected bones will be displayed.

Trajectory times

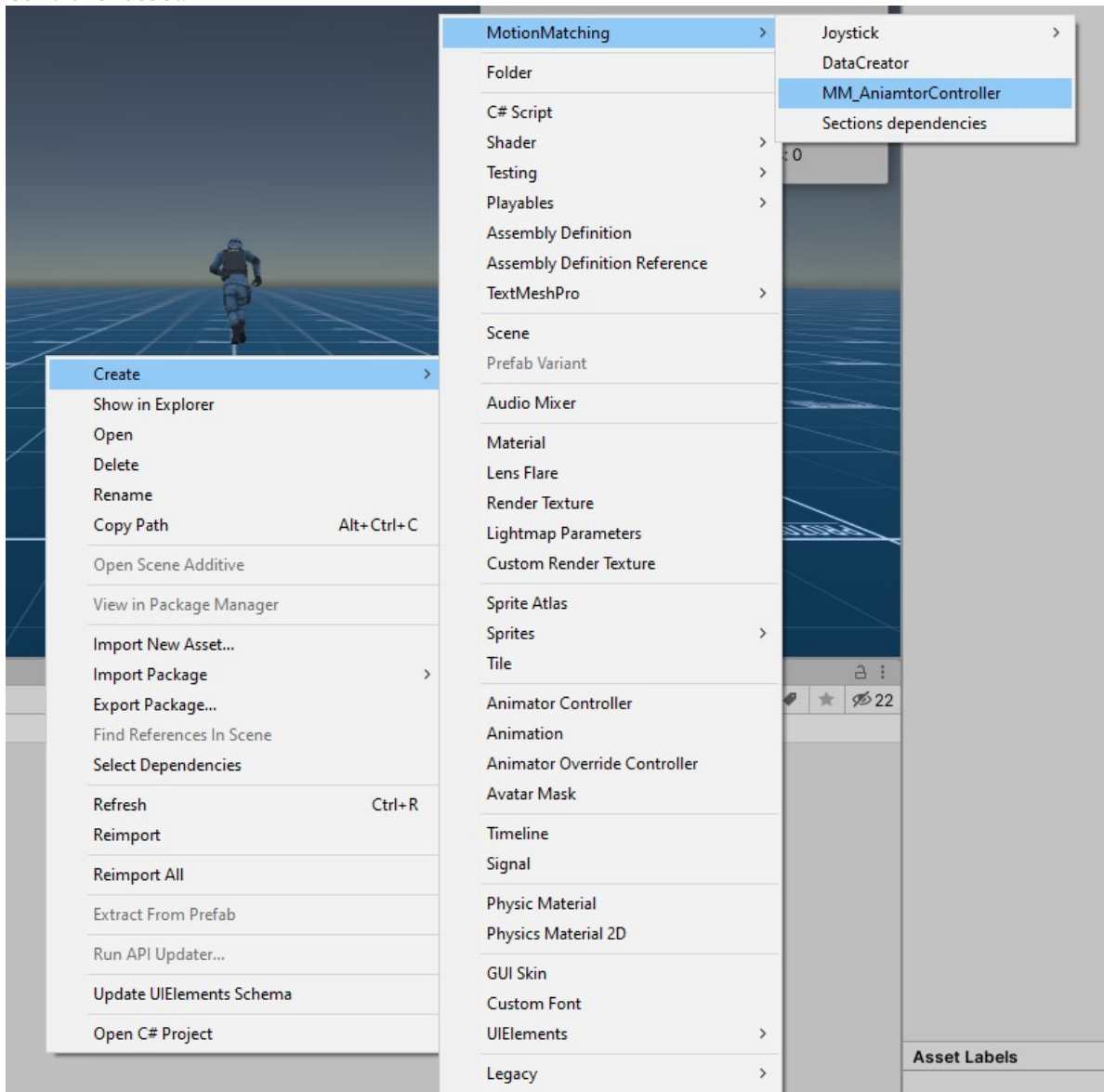
We add here the times at which the trajectory should be created in each frame of the data contained in AnimationData.

Don't overdo it with their quantity. The more points in the trajectory, the more calculations and lower efficiency, 5 points are enough, two of which should have negative time.

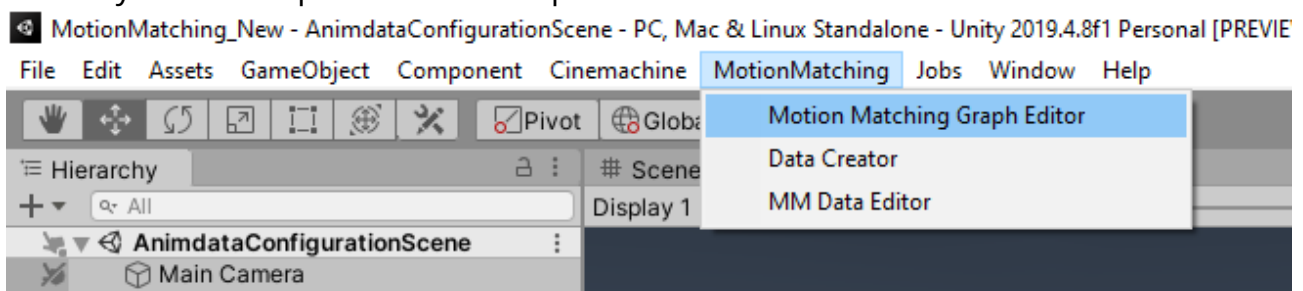
Button**Calculate Only Clips**causes only animations from Basic Options to be calculated, without animations from the menu:**BlendTrees**and**Animation Sequences**.

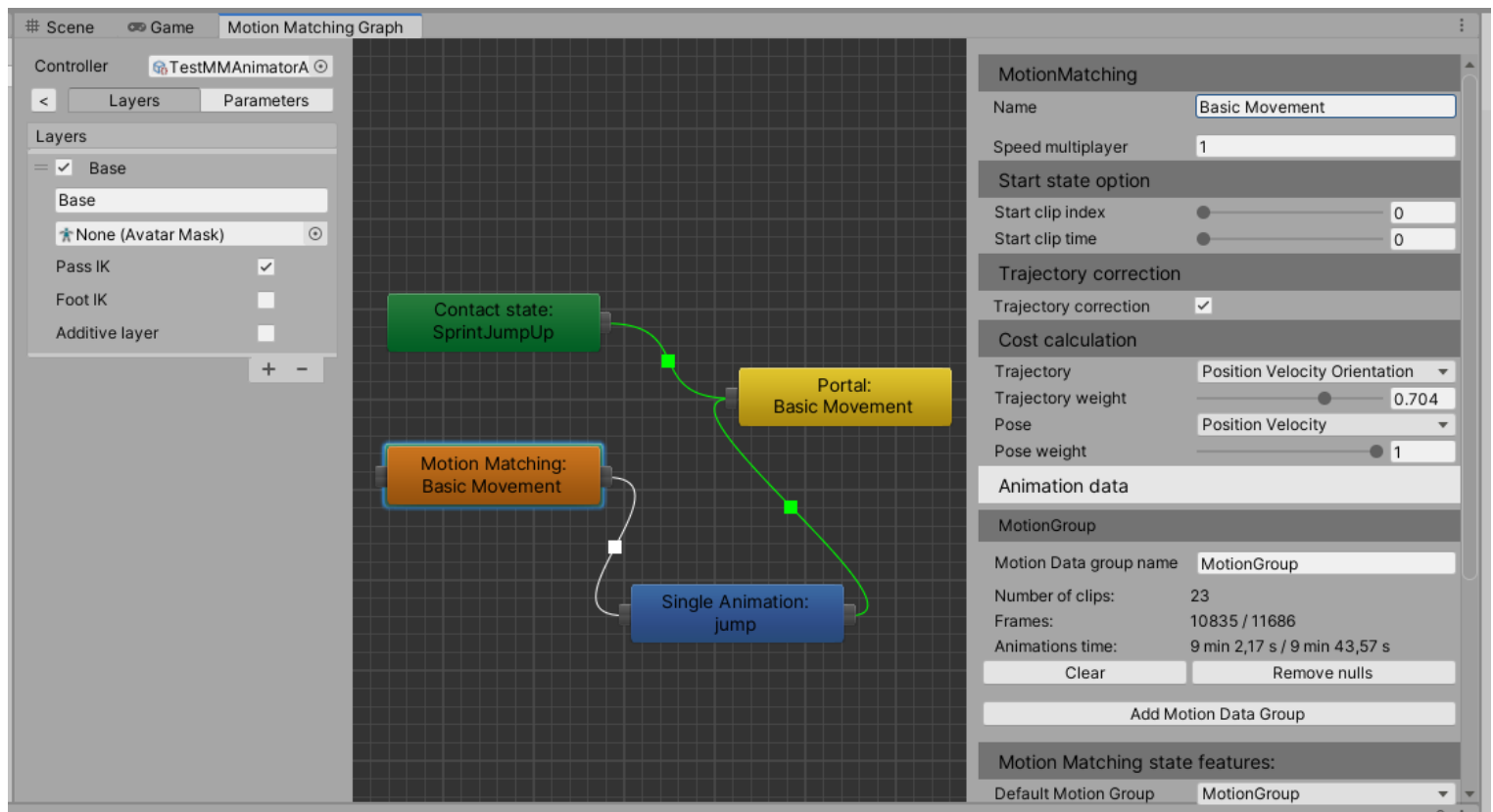
5. Motion Matching Graph Editor-Creation Motion Matching Animator Controller-equivalent of the Animator Controller.

We start working with the Motion Matching Graph Editor by creating the Motion Matching Animator Controller asset:



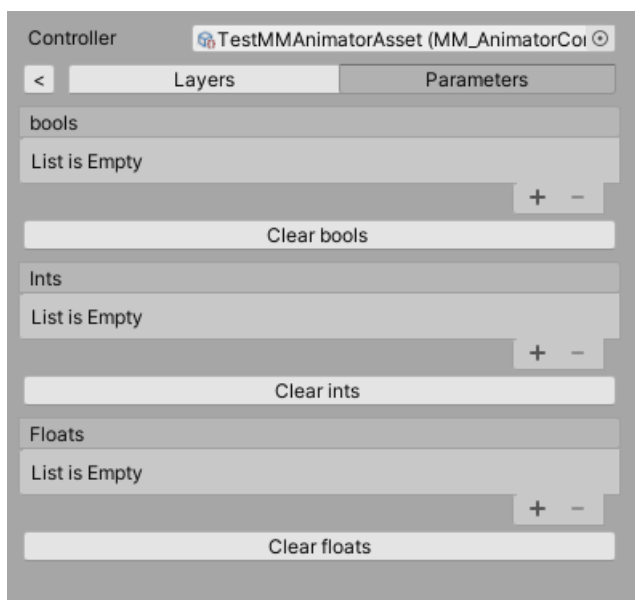
The Motion Matching Graph Editor should open by double-clicking the created asset if you do not open it from the top menu:





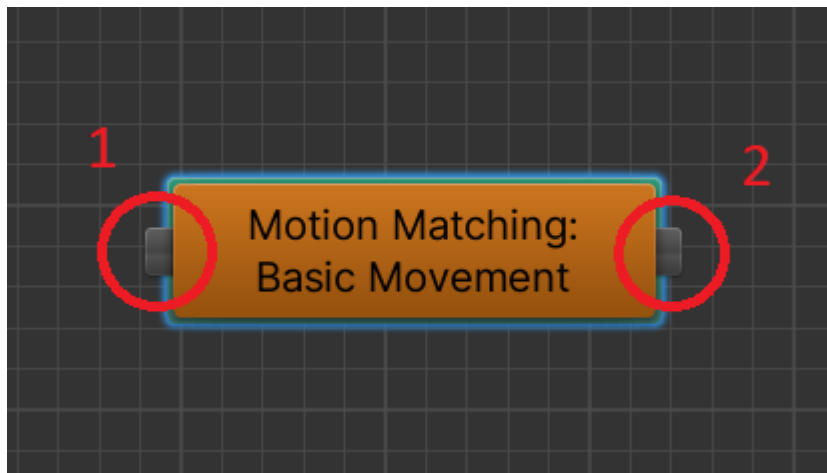
MMGE has 3 fields:

1. Left Space – it is divided into two menus:
 1. Layers – visible above, information about the layers is displayed here, we can also create new layers and set their options.
 2. Parameters – here we set parameters that will be used to set the conditions for transitions between states. Currently, it is possible to add parameters of type bool, int and float.



3. The button marked "<" allows you to collapse the menu to the left. It can be reopened by clicking very close to the left edge of the editor.

2. Graph Space – here we create a visual representation of states and transitions between them. The animation graph is created from left to right by default.

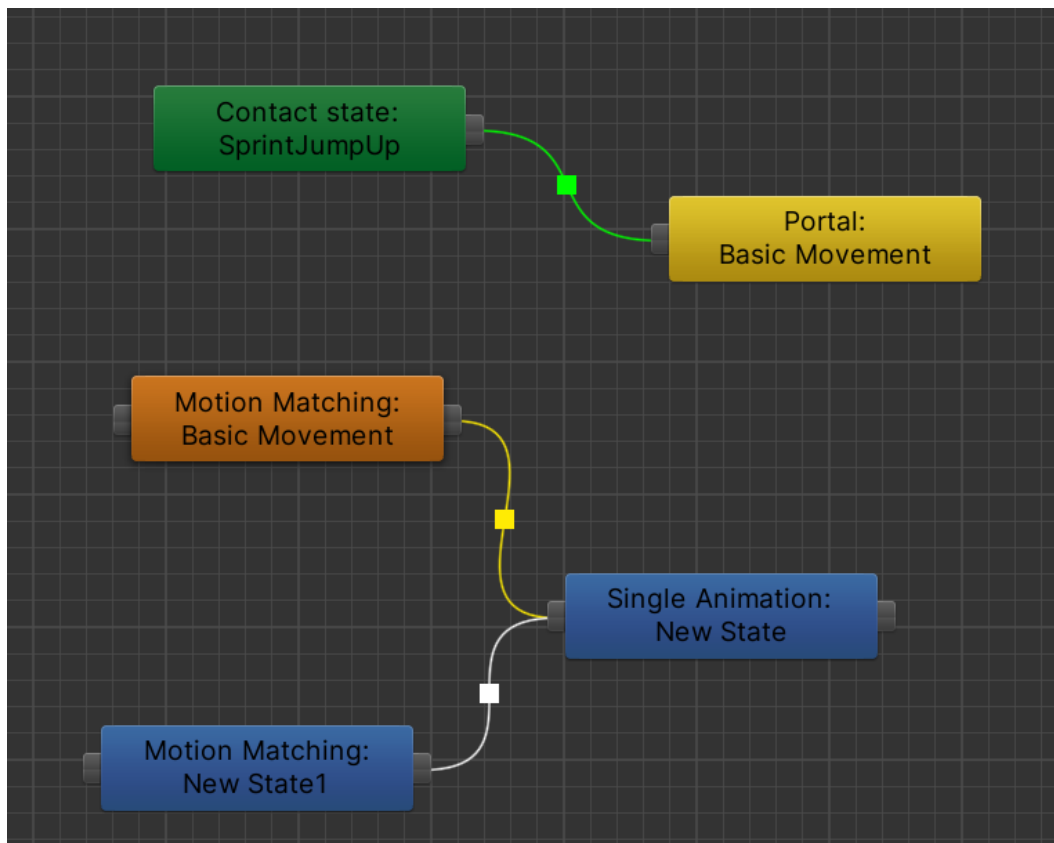


1 – state input 2 – state output

Currently we can create 4 types of states:

1. Motion Matching State – this is the state that implements the MotionMatching idea.
2. Single Animation State – this is a state in which a single animation is played (this can be changed) selected from all animations in this state based on motion matching.
3. Contact State – this is a state that can only have outputs. We can enter the state only from the code level using a special component method MotionMatching.
Contact state is divided into two subsequent states (this changes in Right Space) 1.
Contact – the character is moved between previously established contact points.

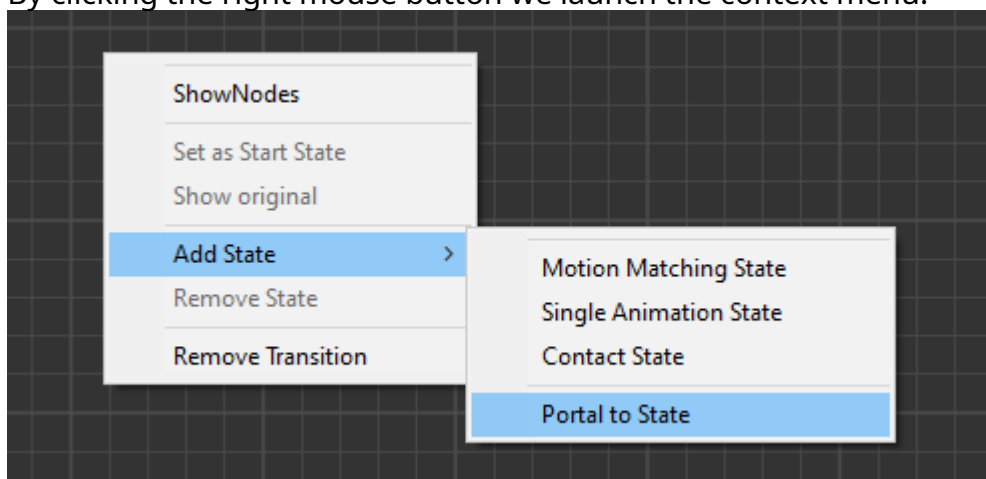
2. Impact – the animation is selected based on the provided Impact Point (currently in testing phase).
4. Portal State – this is a state that graphically represents another state. Portal state has no outputs.



The orange color means that the state is the starting state of the selected layer and the system starts working from it. The starting state can be SingleAnimation and MotionMatching.

Yellow color is for portals, green color for Contact/Impact states, and blue color for other states. Connections between SingleAnimation, Contact, and MotionMatching states are white, green color is for connections to portals. Yellow color is for the currently selected connection.

By clicking the right mouse button we launch the context menu.



ShowNodes – moves the view in Graph Space to the center of gravity of all states. Set as Start State – sets the selected state as the starting state

Show original – if the currently selected state is of type Portal, moves the view to the original state.

Add State – allows you to create states of specific types Remove

State – removes the selected state

Remove Transition – removes the selected connection

Connections between states are created by pressing the left mouse button and releasing it when the cursor indicates the state we want to connect to. We can have one connection between two different states if the state already has a connection to another state and we want to be able to return to it, we must use Portal State.

Transitions are selected by clicking on the square in their center.

3. Right Space – is the place where the options of the currently selected element in the Graph Space are displayed.

STATE PORTAL:



Portal node state selection

State name

Basic Movement

New State

New State1

We can choose which state this portal is.

States **Motion Matching**, **Single Animation** and **Contact**, some of the options are identical, and each has its own type-specific settings.

COMMON ELEMENTS:




MotionMatching

Name Basic Movement

Speed multiplayer 1

At the very beginning the type of the selected state is displayed, here we can change the name of the state and the speed at which the animations will be played.

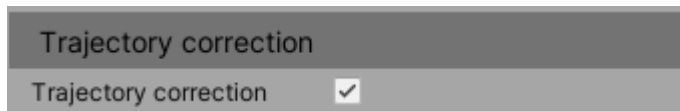


Start state option

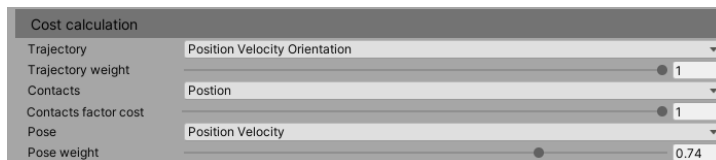
Start clip index 0

Start clip time 0

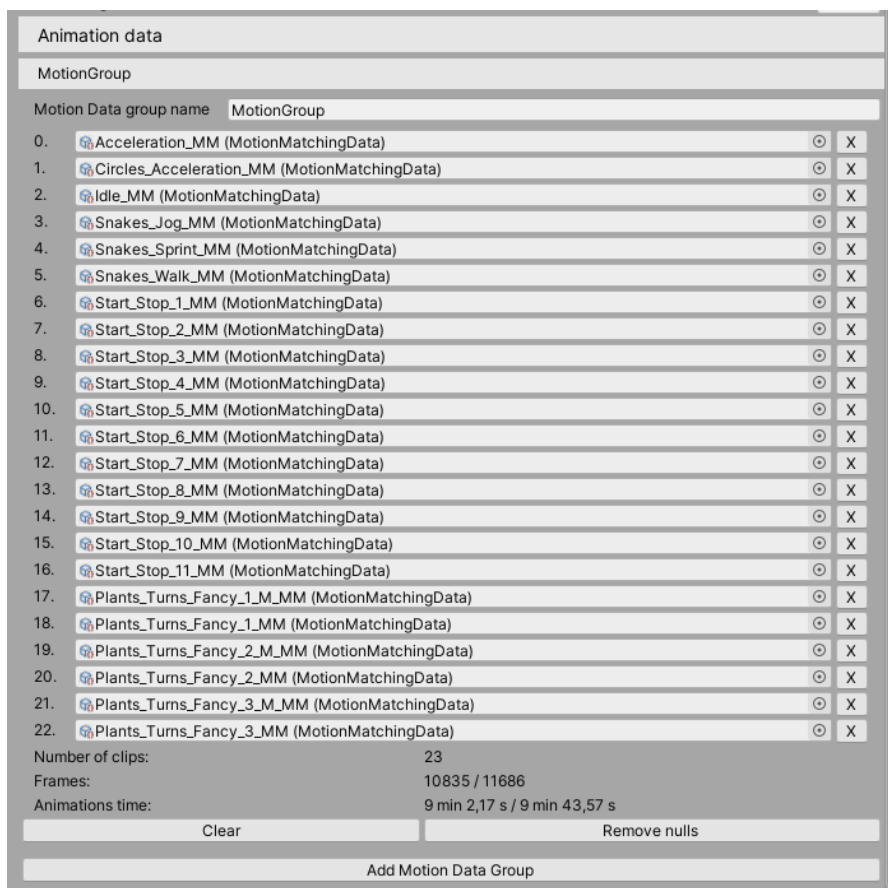
Start state option are options that appear only in the start state. Here we can set the start index of the animation, and the time at which it will start playing.



If the Trajectory Correction option is checked, the trajectory correction will take place, its options are set in the Motion Matching component and will be discussed there. Trajectory correction can be done in the Motion Matching State and Single Animation State.



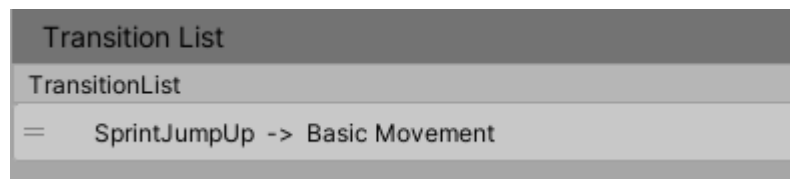
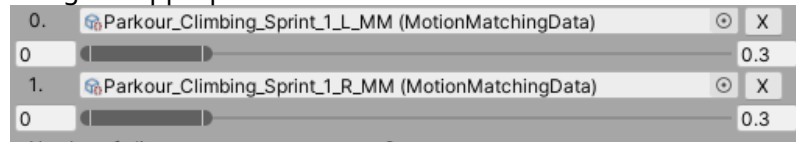
In the Cost calculation options we can select cost methods and their weights, different status types may have more or less options to set in this section.



Animation Data – here we place AnimationData assets that the selected state will use. A Motion Matching state can have several data groups that can be switched between. We can use them if we want to have animations with only normal movement in one group and animations of movement when our hero is tired in the other. We change the current group from the code level, using the appropriate method of the Motion Matching component. After clicking on this bar, we can collapse the list of animations. A summary of the given group is displayed below the list: quantity

animation, total number of frames (frames used by the state / all frames in Animation Data) and total animation time (frames used / all animation frames) Animation data is added to groups by dropping them on the bar with the group name. Single animation and Contact states can have only one group and it is not possible to add more.

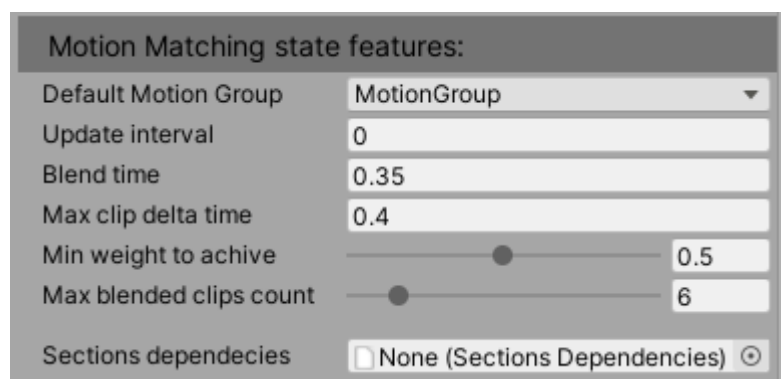
Additionally, in states of the Contact and Single Animation type we have the option of where the animation will be searched for if we decide to go to the state directly from the code, using the appropriate method.



In the Transition List section we have a list of transitions from the selected state, we can change the order in which they will be checked.

The last group of options is **State Features**, each of the state types (MotionMatching, SingleAnimation, and Contact) have different options in this section.

MOTION MATCHING STATE FEATURES



Default Motion Group-the group in which animations will be searched for after entering the state.

Update interval-how often will the search for the next best animation be performed. If it is zero the search will be performed with a frequency equal to the current frames per second at which the program is running.

Blend time-the time in which the subsequent drawn animations will blend.

Max clip delta time-If during the search we find the same animation and the difference between the time of the currently playing animation and the randomly selected one is less than or equal to this option, blending to the randomly selected animation will not occur.

Min weight to achieve-this is the weight that a random animation must reach before its weight starts decreasing again. This weight is before normalization. Setting its value to greater than zero makes the blending between animations smoother, and there is no effect of fast jumping between animations.

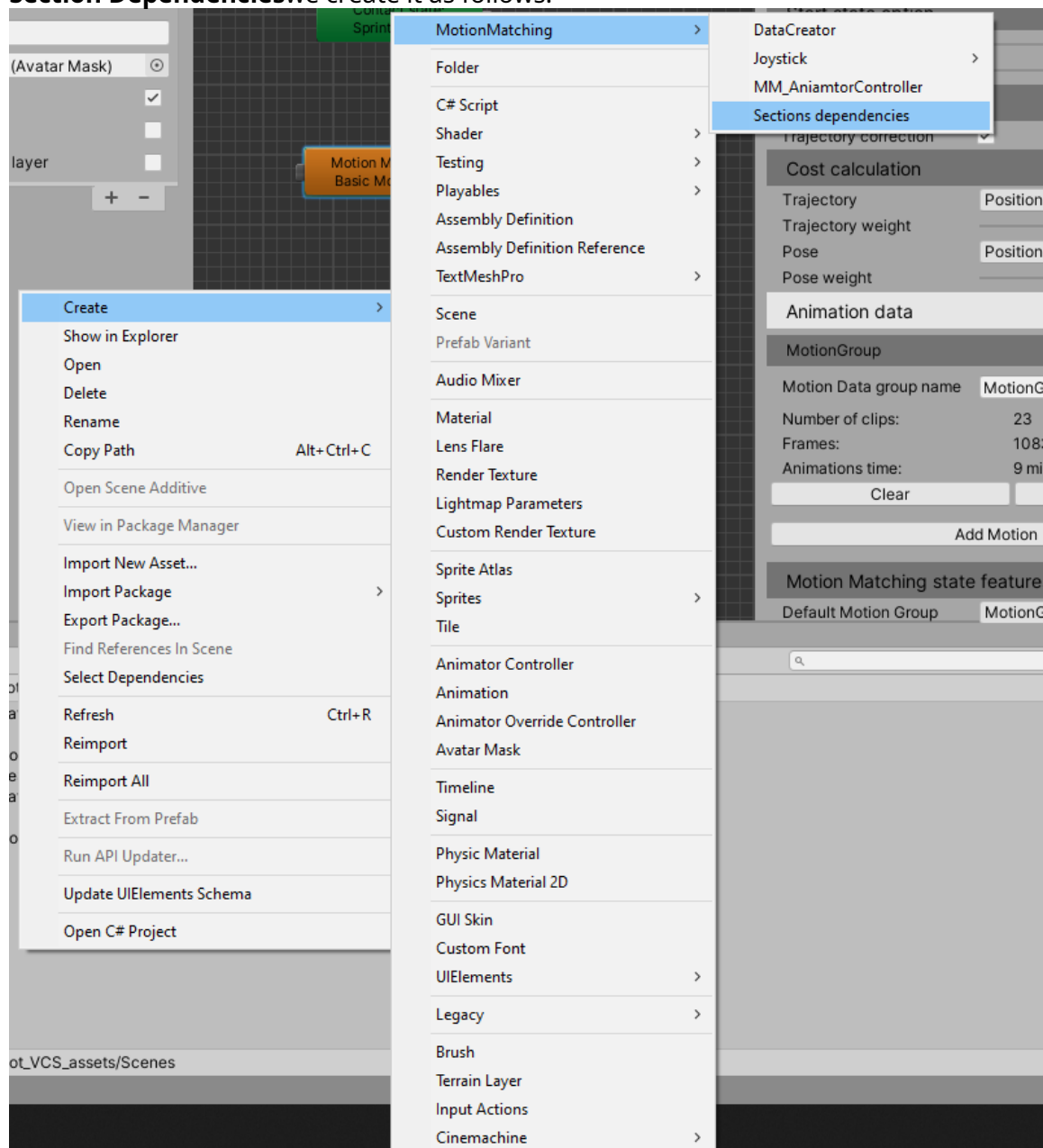
Max blended clips count-defines the maximum number of animations that can be running at the same time

blended. If currently as many animations are blended as this option, the next animation is not searched for. During testing it turned out that this value should not exceed 10, and should not be less than 4. It is possible to set a higher number.

Section Dependencies:

Motion Matching is about constantly searching the animation and choosing the best place that fits the current situation. This search involves calculating the cost to each of the data frames and choosing the one with the lowest cost.

Section Dependencies(section dependencies) is an asset that can be used to influence the cost of searched frames depending on which sections the current time of the currently played animation belongs to and which sections the searched data frames belong to. Asset **Section Dependencies**we create it as follows:



We can edit this asset by selecting it in the Project window, and editing it in the Inspector. Inspector **Section Dependencies** looks like this:

SectionsDependencies

Add Section

0. Always

Section X

Section name Section

Cost to section Cost weight

Section 1 X

Section_1 1.2 X

Section_2 0.8 X

Add section info

Section_1 X

Section name Section_1

Cost to section Cost weight

Section_2 1.2 X

Add section info

Section_2 X

Section name Section_2

Cost to section Cost weight

Section 0.95 X

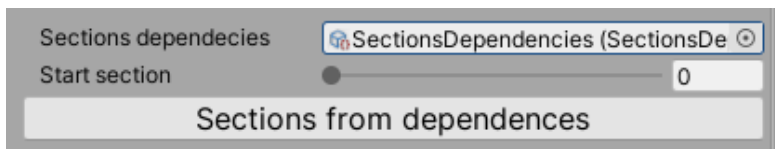
Add section info

The "Add Section" button is used to add a new section. Then, by clicking on the section name bar, we can expand the options we can set, we can change its name, and set the costs of switching to selected sections.

Using the "Add Section Info" button, we add information in which we can set the section for which the cost is to have a set weight. From the drop-down list, we can select the section for which we are setting the cost. If the section cost is less than 1, it means that the frame belonging to this section has a greater chance of being selected. Accordingly, a cost greater than 1 means that the frame belonging to this section has a smaller chance of being selected. If the frames belong to several sections at the same time, the cost of moving to these sections will be set, the final cost of moving to this frame will be the product of the section costs, e.g. the final cost of the frame belonging to Section_1 and Section_2 calculated from the Section section will be:

$$\text{FinalCost} = \text{FrameCost} * \text{CostWeight}_{\text{Section}_1} * \text{CostWeight}_{\text{Section}_2}$$

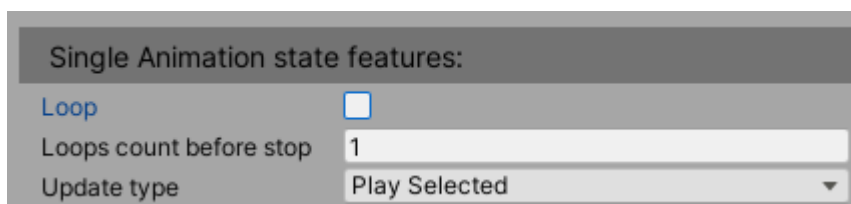
After placing the asset **Section Dependencies** Additional options will appear in the MotionMatching state features field:



HomeSection-this is the section number that will be set by default when entering the state.

Sections from dependencies-this is a button which, when pressed, will add AnimationData in sections created in the asset in any state **Section Dependencies**.

SINGLE ANIMATION STATE FEATURES:



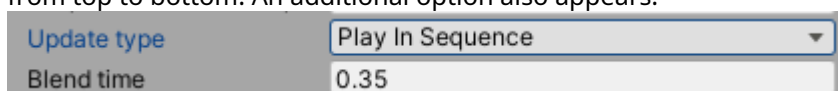
Loop-if checked the selected AnimationData will play in a loop (if the animation belonging to it is looped).

Loops count before stop-the number of loops the selected AnimationData will perform before it stops

The above two options may not work!! to be verified.

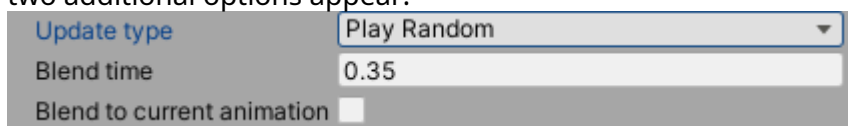
Update type – this is the type of animation updates of a given state. There are currently 3 types of updates:

1. Play Selected – one animation selected at the input is played.
2. Play in Sequence – animations are played in sequence according to their arrangement from top to bottom. An additional option also appears:



Blend time-this is the blending time between animations.

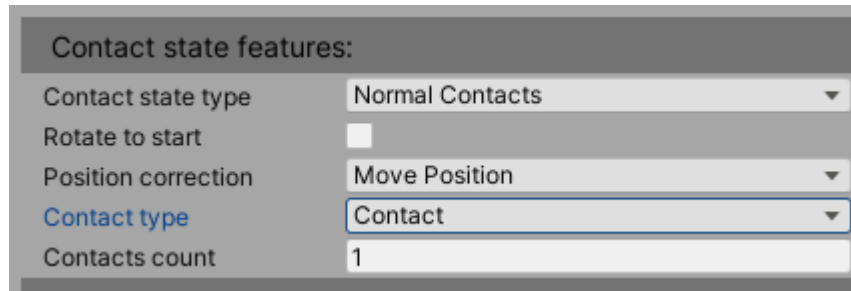
3. **Play Random**-after the random animation in the Single Animation State finishes playing, the next animation is randomly selected to play. The animation is not randomized using Motion Matching. When this type of state update is selected, two additional options appear:



Blend time-this is the blending time between animations.

Blend to current animation-if checked, the randomly drawn animation may be the same one that is currently playing. Otherwise, the same animation will definitely not be drawn.

CONTACT STATE FEATURES



The screenshot shows a settings panel titled "Contact state features:". It contains five rows of controls:

- Contact state type**: A dropdown menu currently showing "Normal Contacts".
- Rotate to start**: A checkbox that is currently unchecked.
- Position correction**: A dropdown menu currently showing "Move Position".
- Contact type**: A dropdown menu currently showing "Contact".
- Contacts count**: A text input field containing the number "1".

Contact state type-here we can choose the type of contact status which was already mentioned earlier. The current types are **Normal Contacts** and **Impacts**.

How Contact State works:

After entering Contact State, the best animation is selected based on the trajectory cost, pose and contact point cost. Then, based on the Contact State Features settings, the animated object is moved appropriately between the transferred contact points. However, before we start using Contact State, we must first prepare the Animation Data that are placed in the state, i.e. add and calculate the contact points. This is done using **Motion Matching Data Editor** (more on this in the description of the Motion Matching Data Editor).

We can only enter Contact State from the code level. This is because when entering Contact State we have to pass to it the appropriate number of contact points depending on the state settings and Animation Data placed in the state.

Important !!!!!!!

All Animation Data placed in the Contact state must have the same number of contacts.

Rotate to start-if checked after entering the state the animated object will be set in the appropriate rotation to the starting point of contact.

Position Correction-this is how the object will be moved between the contact points. There are currently two methods:

Move position-the object is moved based on the delta between the contact point passed to the state and the contact point in the selected animation, calculated every frame. It is much smoother than the second method, but can sometimes cause the moving object to jump.

Lerp position-the object is moved using the Vector3.Lerp function. The object's movement may not be smooth, but it does not cause the animated object to jump

Contacts count-number of contact points used. This number refers to the points between the first and last contact in AnimationData. The first point is called Start, and the last one is Land. The contacts in between are called Contact, and this option applies to them.

Contact type-is the way the cost of contact points will be calculated and the way the object will be moved into the contact state. There are currently 5 types, and the possibility

disabling shifting (in the future it will be possible to implement your own method):

1. Start Contact – the object is moved to the first contact and then between them.
2. Contact – the object is moved only between contacts
3. Contact Land – the object is moved between contacts and finally to the last point.
4. Start Land – the object is moved first to the start and then to the last point (Land).
5. Start Contact Land – the object is moved first to the start, then between the contacts, and then to the last point (Land).

The number of contacts (Contact) is set by the option **Contacts count**.

What should the table of contact points that we pass in the method look like?

SwitchToContactState component **MotionMatching?**

The contact list being transferred should always have a start point, i.e. a point with index 0 and an end point located at the last place in the list. If they are not used, which results from the state settings (e.g. **Contact Type** is equal **Contact**) the number of contacts transferred should be the number of contacts plus 2 (start and end).

State Transition Options

New State -> New State1

Transition options

= New option + -

Common options

Blend time 0.3

From state options

Check transition on max lenght ☐

Parkour_Climbing_Sprint_1_L_MM (MotionMatchingData) 1.2175 1.7

Parkour_Climbing_Sprint_1_R_MM (MotionMatchingData) 1.21 1.7

To state options

jumpOver_LH_MM (MotionMatchingData) 0 0.9956

JumpOver_RH_MM (MotionMatchingData) 0 1.0122

Option Conditions

Bool conditions

= New bool IS_TRUE + -

Int conditions

= New int GREATER 54 + -

Float conditions

= New float LESS_EQUAL 0.4 + -

At the very top we have the option to set the number of options that the created transition should have. Each option can have different settings.

Common options

This section contains settings for all possible transition types. Currently there are two 4 types of transitions between state types:

1. Single Animation → Single Animation
2. Single Animation → Motion Matching
3. Motion Matching → Single Animation
4. Motion Matching → Motion Matching

(Contact states are considered a Single Animation here).

Currently, I can set one option here, namely the blending time between states.

From State Option Section

These options apply to the state we are leaving, they only appear here if the state is of type SingleAnimation and Contact. In this section we can choose in what time interval of each animation the conditions set below are to be checked. If the option **Check transition on max Length** is checked, the conditions will start to be checked when the time of the currently playing animation is greater than this time reduced by the previously set time **blend time**.

To State option section

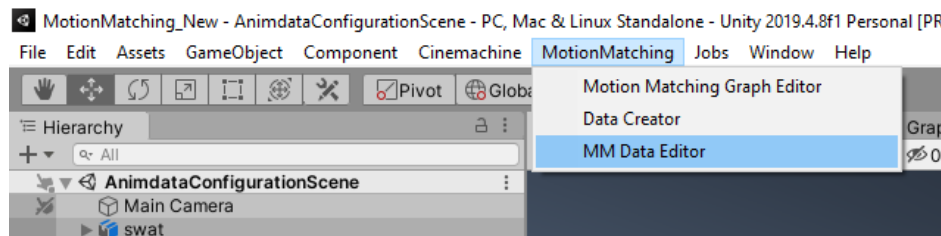
These options apply to the state we are leaving, they only appear here if the state is of type SingleAnimation and Contact. In this section we can choose in what time interval of each animation the best frame will be searched for to start blending.

Option Condition

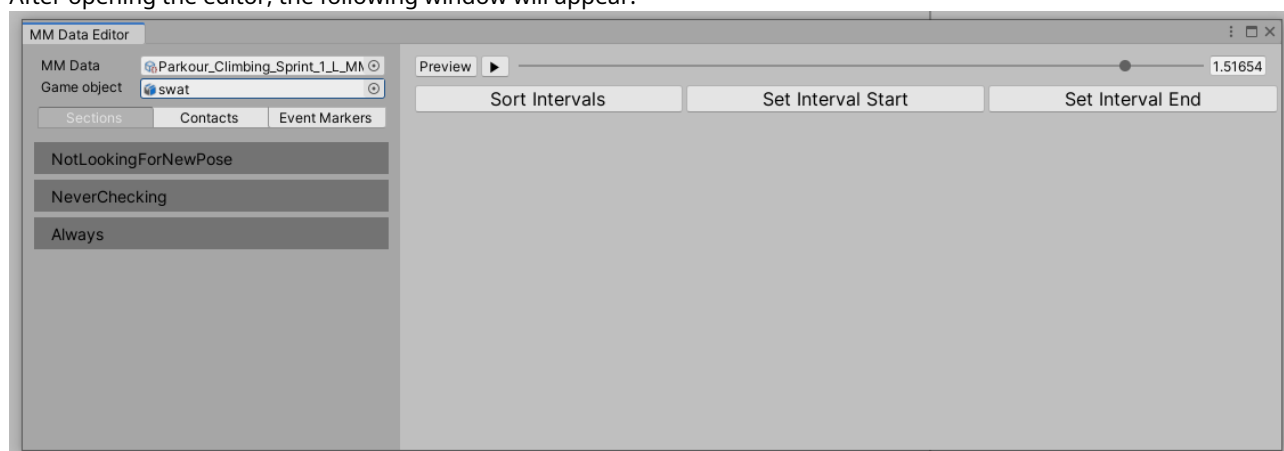
In this section we set the conditions that must be met to start the transition to the next state. All conditions must be met.

6. Motion Matching Data Editor

Motion Matching Data Editor is an editor designed to edit selected AnimationData options created by Motion Matching Data Creator. In it we can edit AnimationData sections, create Contact and Impact points and add so-called Event Markers. Motion Matching Data Editor is opened by double-clicking on the AnimationData asset or from the Unity top menu:



After opening the editor, the following window will appear:



It is divided into two parts. On the left side I have general options, or a list of elements of the currently edited settings. On the right side I have the ability to play animations (on the stage) and the ability to set the options that we have selected on the left side.

Left side of editor:

MM Data-currently editing AnimationData.

Game object-the object on which the animations will be played and on the basis of which the calculations will be performed (Contact/Impact points).

Then we can choose what we want to edit (**Sections, Contact, Event Markers**).

Right side of editor:

Button **Preview**-using it we create an animation graph with which the animation will be played. Additionally, the animated object is set to position (0,0,0) (I recommend creating a separate scene for editing Animation Data). If the animation is currently playing, it will be stopped and its time will be set to 0.

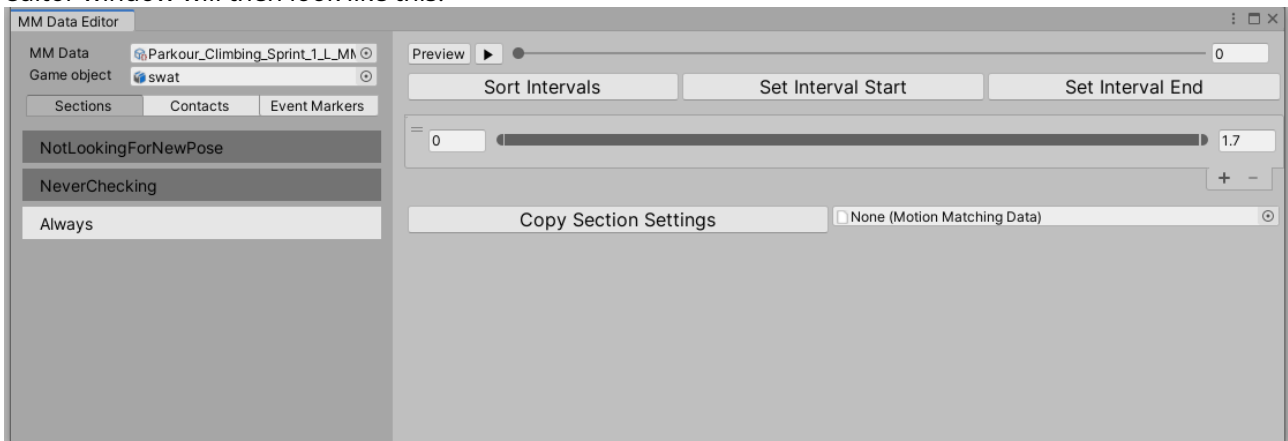
Button **Play**-is used to start playing an animation. When pressed, it changes to a pause animation button.

The slider is used to manually scroll through the animation.

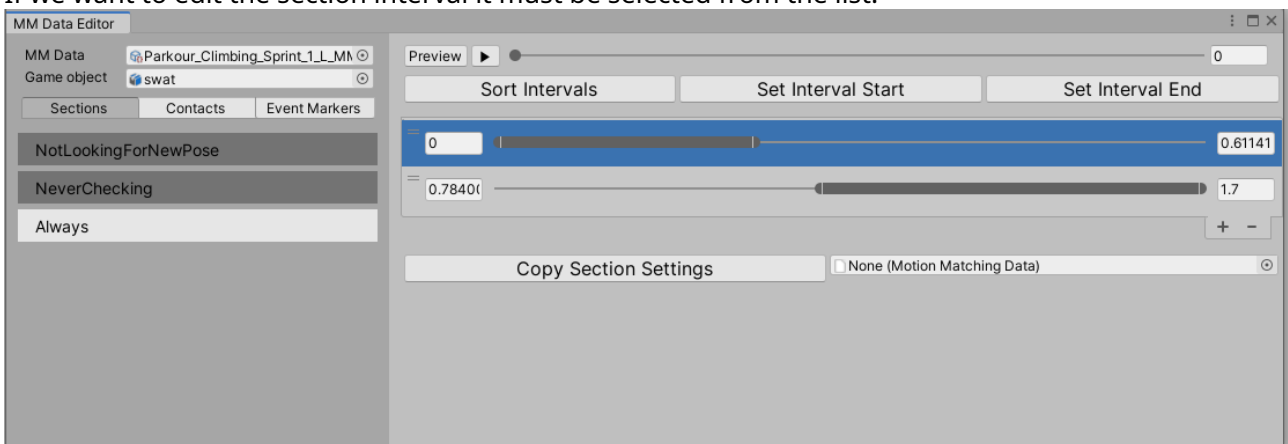
If we want to play or count contacts in the selected AnimationData we need to set **Game Object**. Otherwise it will not be possible to calculate or play the animation.

Description of editing Section:

After selecting from the ToolBar**Sections** on the left side, the existing sections in AnimationData will be displayed. After selecting a section, the ranges of that section will be displayed on the right side. The editor window will then look like this:



If we want to edit the section interval it must be selected from the list:



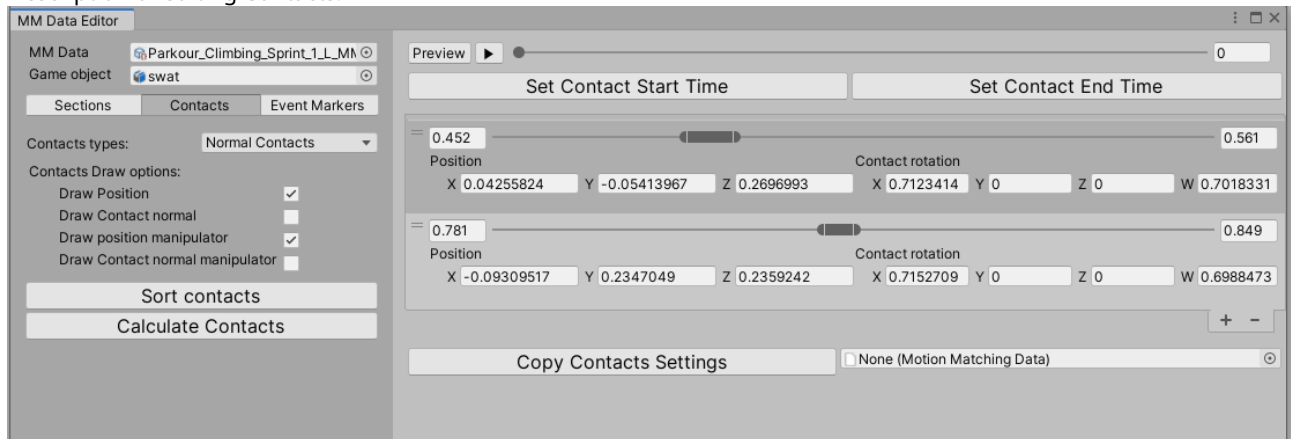
The buttons above the intervals are used to: Sort intervals – sorts the section intervals.

Set Interval Start – sets the start of the selected interval to the current animation time, identical to the animation time on the slider above.

Set Interval End – sets the end of the selected interval to the current animation time, identical to the animation time on the slider above.

Below there is a button with which we can copy section settings from another AnimationData, this is useful when we have AnimationData whose animations are "mirrored".

Description of editing Contacts:



The following options will appear on the left:

Contact types-Here we can choose whether AnimationData will have Contacts or Impacts.

Draw Position-if selected the positions of the currently calculated Impac/Contact AnimationData points will be displayed in the scene view.

Draw Contact normal-if checked in the scene view the normal directions of the currently calculated Impac/Contact AnimationData points will be displayed.

Draw position manipulator-if selected, a manipulator will be displayed in the scene view, positioning the point currently selected on the right side of the Contact/Impact.

Draw contact normal manipulator-if selected, a rotation manipulator for the currently selected point on the right side of the Contact/Impact will be displayed in the scene view.

Sort Contacts Button-sorts touchpoints according to their time.

Calculate contacts button-calculates contact points in the selected AnimationData according to their settings.

On the right side we have the option to set contact/impact points:

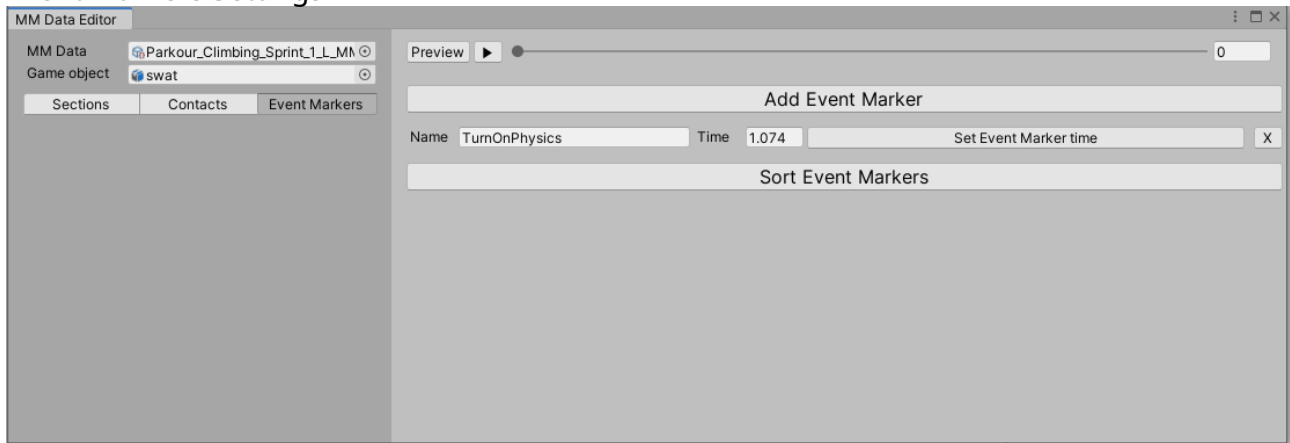
Button**Set Contact Start Time**-is used to set the starting contact time to match the current animation time.

Button**Set Contact End Time**-is used to set the final contact time to match the current animation time.

Button**Copy Contact Settings**-copies contact settings from the selected AnimationData.

In the list of current contacts we can see the current settings of a particular contact. If any of them has been selected in the scene view, the appropriate manipulators will be displayed (if they have been selected on the left side of the editor). Here we can see the start and end of the contact time, the contact position (local to the animated object) and the orientation (which is used to calculate the normal vector of the contact). We can change the time settings using the buttons as well as the slider. We change the orientation and position settings using the manipulators in the scene view. If we want to edit a specific point, it must be selected from the list of all contacts.

Event Markers Settings



There are no additional options for Event Markers on the left side.

On the right side we have:

Button **Add Event Marker** used to add a new marker.

Next we have a list of markers, with the marker name, its time, buttons to set the time to match the current animation time, and a button to remove the marker.

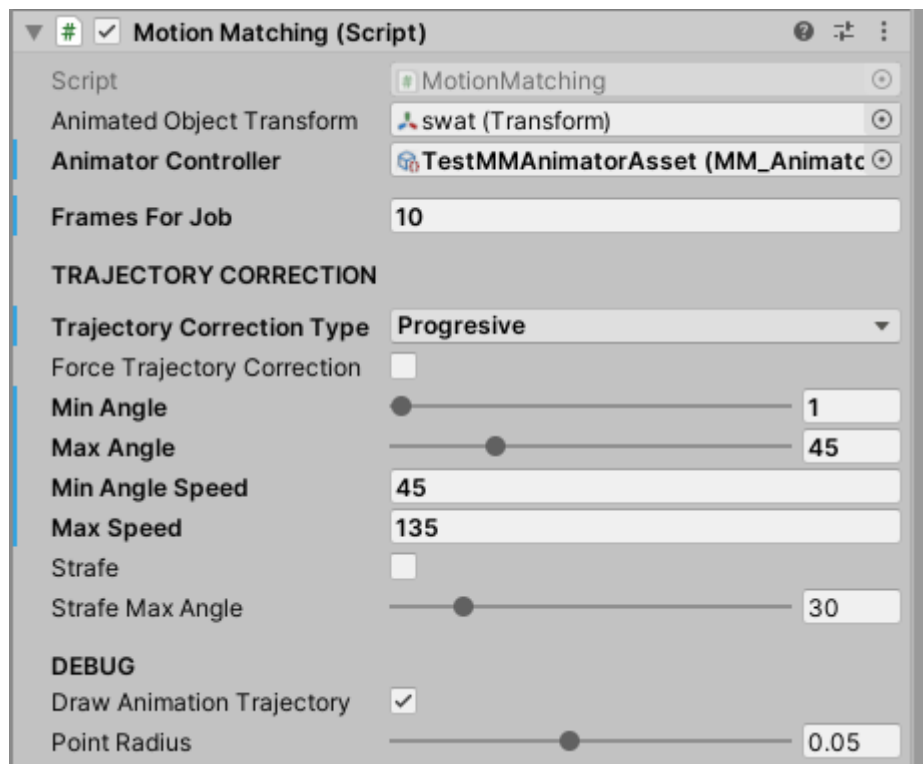
The Sort Event Marker button is used to sort the markers based on their time.

Failure to sort the markers may cause errors (this will be changed in the future).

We use event markers using MotionMatchingStateBehavior, more about this when discussing the Motion Matching component.

7. Component Overview **Motion Matching**

This is the main component that implements the Motion Matching Animator Controller logic.



Animated Object Transform – this is the transform of an animated object, if not set it will assume the transform of the object to which the component belongs.

Aniamtor Controller – our own created Motion Matching Animator Controller.

Frames for job – this option sets how many animation frames should be counted in one thread. This number shouldn't be too small, because calculating 100 animation frames doesn't take much time and we don't want to create 10 jobs for this, where each of them will deal with 10 frames.

TRAJECTORY CORRECTION – these are options that we set so that the system corrects the trajectory of the animated object in accordance with the trajectory provided by the player.

Trajectory correction type–is the way the trajectory is corrected, currently there are 4 methods of trajectory correction:

- 1.Constant – the animated object is rotated at a constant speed so that the first point of the trajectory of the currently playing animation coincides with the first point of the trajectory requested by the player.
- 2.Progressive – works identically to constant with the difference that the rotation speed of the animated object depends on the angle between the direction to the first point of the animation trajectory and the trajectory transmitted by the player.
- 3.Match Orientation Constant – the animated object is rotated at a constant speed so that the orientation of the first point of the animation trajectory matches the orientation of the first point of the trajectory transmitted by the player.
- 4.Match Orientation Progressive – the rotation direction is also selected based on the orientation of the points, but the rotation speed depends on the angle between these orientations.

* The first point of the trajectory, i.e. the first point whose time is greater than 0.

Force trajectory correction-if checked, trajectory correction will occur in states where it is disabled (only in Single Animation and Motion Matching type states).

Min Angle and Max Angle- the range within which the angle between the trajectories (calculated depending on the type of trajectory correction) must be in order for trajectory correction to occur.

Min Angle Speed-this is the speed in degrees per second at which the animated object is to be rotated when the angle between the trajectories is equal to Min Angle.

Max Angle Speed-this is the speed in degrees per second at which the animated object is to be rotated when the angle between trajectories is equal to Max Angle. It is also used as a constant speed when the trajectory refinement type is set to Constant or Match Orientation Constant.

Strafe-if we want the character to "strafe" we check this option. Additionally we need to pass the strafe direction from the code to the component.

Strafe Max Angle-if the angle between the strafe direction passed to the component and the forward direction of the animated object is greater than this value, the trajectory is improved.

Using the Motion Matching component we can add MotionMatchingStateBehavior to individual states.

MotionMatchingStateBehavior is an abstract class thanks to which we can implement logic that will be executed only in the state to which the object of the class inheriting from it will be added. This class has 7 abstract methods that we can implement, they are:

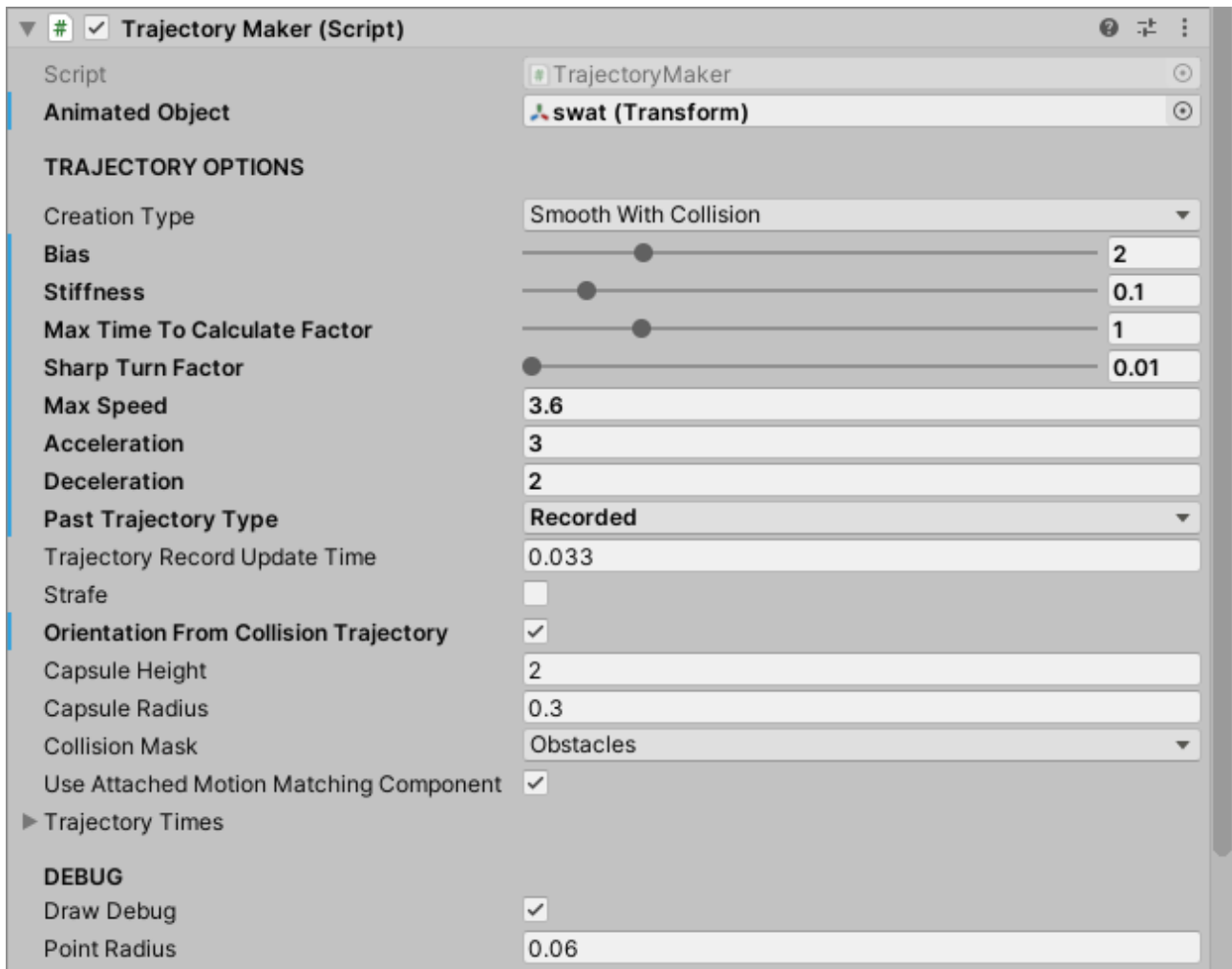
- 1.Enter() – is called after entering a state.
- 2.Update() – The corresponding Update from Mono Behavior.
- 3.LateUpdate() – The Late Update equivalent of Mono Behavior.
- 4.OnStartOutTransition() – this method is called when the conditions for transitioning to the next state are met, before we exit the state.
- 5.Exit() – the method is called after exiting the state, i.e. when the state weight in the animation graph is zero (state blending ends). Exit is executed after OnStartOutTransition()
6. OnContactPointChange(int fromContactIndex, int toContactIndex) – this method is executed only in the Contact type state when the index of the current contact to which the character will be moved changes.
- 7.CatchEventMarkers(string eventName) – in this method we can implement behaviors that should occur when the Event Marker we set (we add them in the Motion Matching Data Editor) has been caught.

This class also contains 3 fields:

1. logicState – reference to the current state we are in
2. mmAnimator – reference from the MotionMatching component
- 3.transform - reference to Transform animated object

8. Component Overview **Trajectory Maker**

Trajectory Maker is a component that creates a trajectory based on which the next animation will be selected. To use Trajectory Maker, you need to set its options according to the animations you have and pass it a value **Input(Vector3)** based on which the trajectory will be created.



Animated Object – this is an animated object from which a trajectory of points with negative time will be created. If it is not set, it takes the value of the object on which the component is located.

Creation type – the type of trajectory that will be created. There are currently 4 types of trajectories:

1. Constant
2. Constant with Collision
3. Smooth
4. Smooth with Collision

Bias–the higher the value, the future trajectory points will reach the target position faster, and the trajectory will have sharper arcs.

Stiffness–if it is 0 it has no effect on the trajectory, if it is one the points will reach their target positions at the same time and the trajectory will not form smooth arcs.

MaxTimeToCalculateFactor-this is the time based on which the coefficient causing the creation of arc-shaped trajectories will be calculated, the shorter the time, the sharper the arcs will be and the points will reach their position faster, the higher the time, the slower the trajectory will be and the arcs will be gentler.

SharpTurnFactor-if it is 0 it has no effect on the trajectory, if it is 1 with fast changes in direction of movement the trajectory points will move faster to the expected position.

MaxSpeed-maximum speed for which the trajectory is to be created.

Acceleration-acceleration with which the trajectory is to be created.

Deceleration-the acceleration by which the trajectory should decelerate when the Input vector is zero.

PastTrajectoryType-future trajectory type for points with negative time. Currently there are 2 types of past trajectory:

1. Recorded – the trajectory is recorded from the Animated Object parameter.
2. Copy From Current Data – the past trajectory is copied from the currently playing animation from the Motion Matching component.

Trajectory Record Update Time-This is the time at which a trajectory point is to be recorded. **Strafe**-if the option is checked the orientation of the trajectory points will be equal to the direction passed to the component (**StrafeDirection**).

Orientation From Collision Trajectory-if this option is checked the orientation of the trajectory points will be equal to the directions to the individual points after the collision. It only works with the trajectory type **Smooth With Collision** and **Constant With Collision**.

Capsule Height-height of the capsule for which the trajectory with collisions will be calculated. **Capsule**

Radius-radius of the capsule for which the trajectory with collisions will be calculated. **Collision Mask**-

mask for which collisions are to be checked. **UseAttachedMotionMatchingComponent**-if this option is checked, the times of the trajectory points will be taken from the MotionMatching component located on the same object as the Trajectory Maker. If the Motion Matching component is not located on the same object, we can manually enter the trajectory times into the list below or reset the MotionMatching component to the Trajectory Maker in the Awake method of a separate MonoBehaviour script.