**Design for the Animation Framework**

**Intro**

This document defines the goals for the Animation Framework, and how they should be achieved.

**Definitions**

EDITOR – The program used to create the SPRITES.

FRAMEWORK – A set of procedures that allow the GAME to read the SPRITES from a file, display them at the correct resolution, and play them as desired.

GAME – A program using the FRAMEWORK.

SPRITE – A set of ANIMATIONS.

ANIMATION – A set of FRAMES which can be played in sequence.

FRAME – A collection of JOINTS.

JOINT – A JOINT contains a shape, a color, and a list of FILTERS which are used to generate an image. In the EDITOR, each JOINT is linked to a BONE.

BONE – BONES are used by the EDITOR to control the position of JOINTS in a FRAME. They are not used by the FRAMEWORK.

SKELETON – A collection of BONES.

PROJECT – A collection of global values that are shared between SPRITES.

**Skeleton editor**

The editor’s skeleton editor allows the user to define the motion of the animations. Each bone is simply represented as a dot, which can be dragged around by the user.

Bones have the following properties:

* Name – The name of the bone. When renaming a bone, if the desired name does not exist, the bone will be added as a new bone. If the name exists in the animation but not the frame, it will be linked with the other ones that share that name. If the name exists in the same frame, the user will be required to chose a different name.
* Frame – The frame in the skeleton that the bone belongs to. Visual indicator only, cannot be changed by the user.
* Parent – The bone that is this bone’s parent. When changed, will also change the parent for all bones linked to this one. All bones must have a parent, with the exception of a single “control” bone.
* Children – All bones with this bone as their parent. Visual indicator only, cannot be changed.
* Links – Number of bones in other frames that have the same name as this bone. Visual indicator only.
* Position – x and y location of the bone in the frame.

The skeleton allows a base skeleton to be defined. Bones in the base:

* Cannot have their parent changed outside the base editor.
* Cannot have their name changed outside the base editor.
* Must exist in all frames in all animations.

The skeleton editor allows for two types of animations: standard and contextual. Standard animations control the entire skeleton at once. Only a single standard animation can play at once. Contextual animations control only part of the skeleton, and can have multiple playing at once.

When playing the animation, the framework sets the positions of the bones in a descending order, starting with the parent bone, and the its children, then their children, ect. Contextual animations have a parent bone(which the contextual animation controls the rotation of, but not the position), and have control of all its child bones. An example of a use for a contextual animation would be to have a sprite with a running animation, and a contextual animation to control the head. The contextual animation could be played to have the character look back in response to a trigger, while the rest of the character continues to run as the player desires.

**Sprite Editor**

The sprite editor allows the user to create the sprite. Each sprite is based off a skeleton (multiple sprites can be based off a single skeleton).

The sprite editor includes a base editor, which allows the user to define a Joint for each bone in the base skeleton. For all animations, the joints will be added to there paired bones. No offsets can be done in the base editor.

Joints can be moved on a frame to give them an offset from the bone, and their shape/color/filters can be edited on a per-frame basis. Optionally, changes to a joint can “cascade” to all of the same joint in later frames of the same animation.

Joints have the following properties:

* Name – The name of the joint. For joints in the base, this cannot be changed outside the base. For “free” joints, this can be changed at anytime, but will break links to other joints.
* Frame – The frame that the joint belongs to. Visual indicator only.
* Bone – the bone that this joint belongs to. Visual indicator only.
* Render Order – defined by the bone’s position in the frame’s list of bones. Displayed here for clarity.
* Shape – A list of points that define the shape of the joint.
* Color – The base color of the shape. Can select a color from the project palette, or give it a custom color. Can be edited in any frame.
* Offset – x and y amounts that the joint is offset from its bone.
* Filters – filters that will be run over the joint when it is rendered.

**Sprite Rendering**

There are three times that the framework does calculations for the sprite: Loading, scaling and playing.

Loading takes place when the game loads the sprites from their files. This will likely happen at startup or during a loading screen. Here, the game reads the sprite file, and creates all the images in memory.

Scaling happens anytime the window is rescaled. This may take a few seconds, so the game should be paused to allow rescaling. Here, the framework recreates the images at the new desired size.

Playing happens anytime the animation to be displayed changes. This can happen when the frame is changed, or a contextual animation is started/stopped/changes frames. The steps here are:

1. Calculate the joints that need to be rendered. This is done by flowing from the control joint down through the child structure, and checking whether any joints have contextual animations active.
2. Order the joints by render order? (not sure if this will need to be done, still working out how render order will work)
3. Draw the joints.