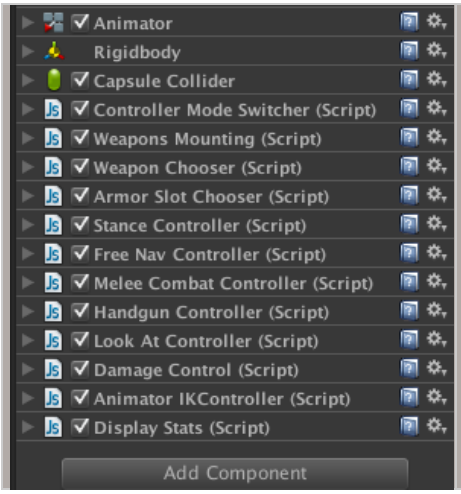
ka·ta [**kah**-*tuh*] - ***noun*** - an exercise consisting of several of the specific movements of a martial art, especially a pattern prescribed for defending oneself against several attackers.

-gon
a combining form meaning “angled,” “angular,” used in the formation of compound words: *polygon; octagon*.

ka·ta-gon [**kah**-*tuh-gon*] - ***noun*** - A system for navigation and stance control based on martial arts patterns and rules for use in moving polygon based game characters around and engaging in combat and defending against attackers.

THE KATAGON SYSTEM



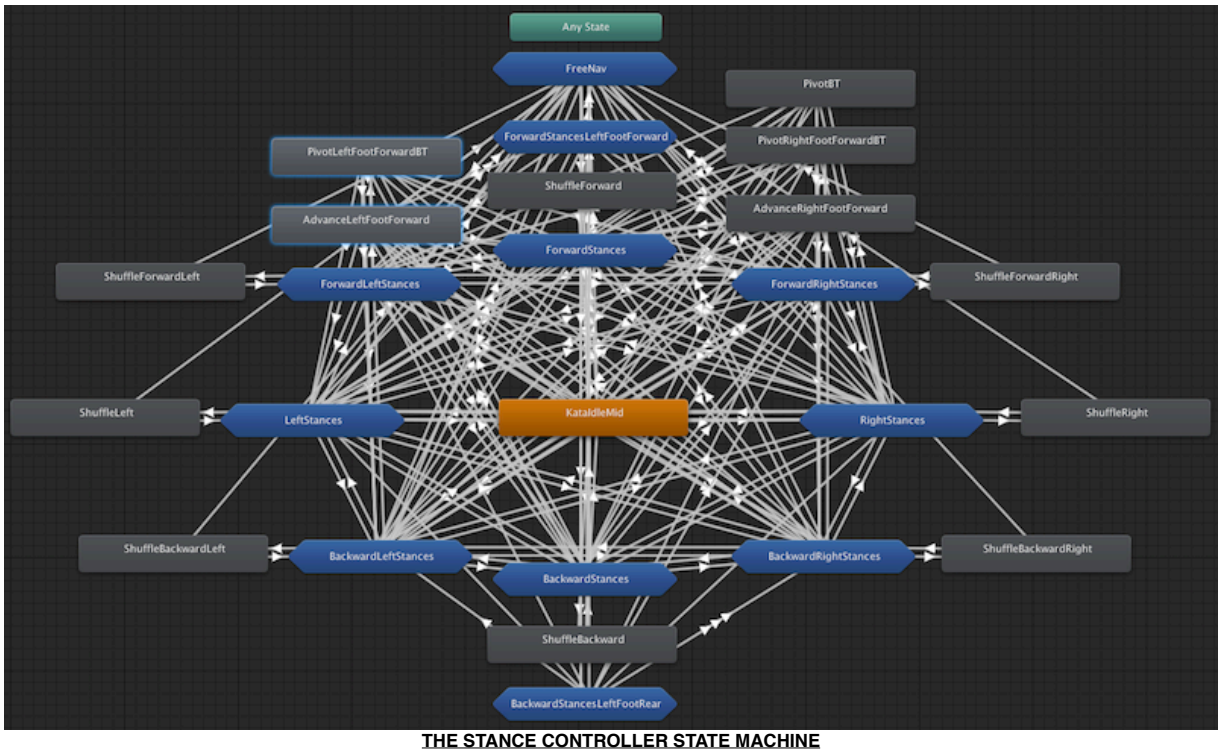
The Katagon Procedural IK Combat & Navigation System

The Katagon System was developed from research done into creating procedural martial arts based rigs using constraining splines that IK goals ride along and martial arts combat and defense theory and rules. With the introduction of the Mecanim module in Unity 4.0 the ability to port this over to Unity for use with Avatars became possible.

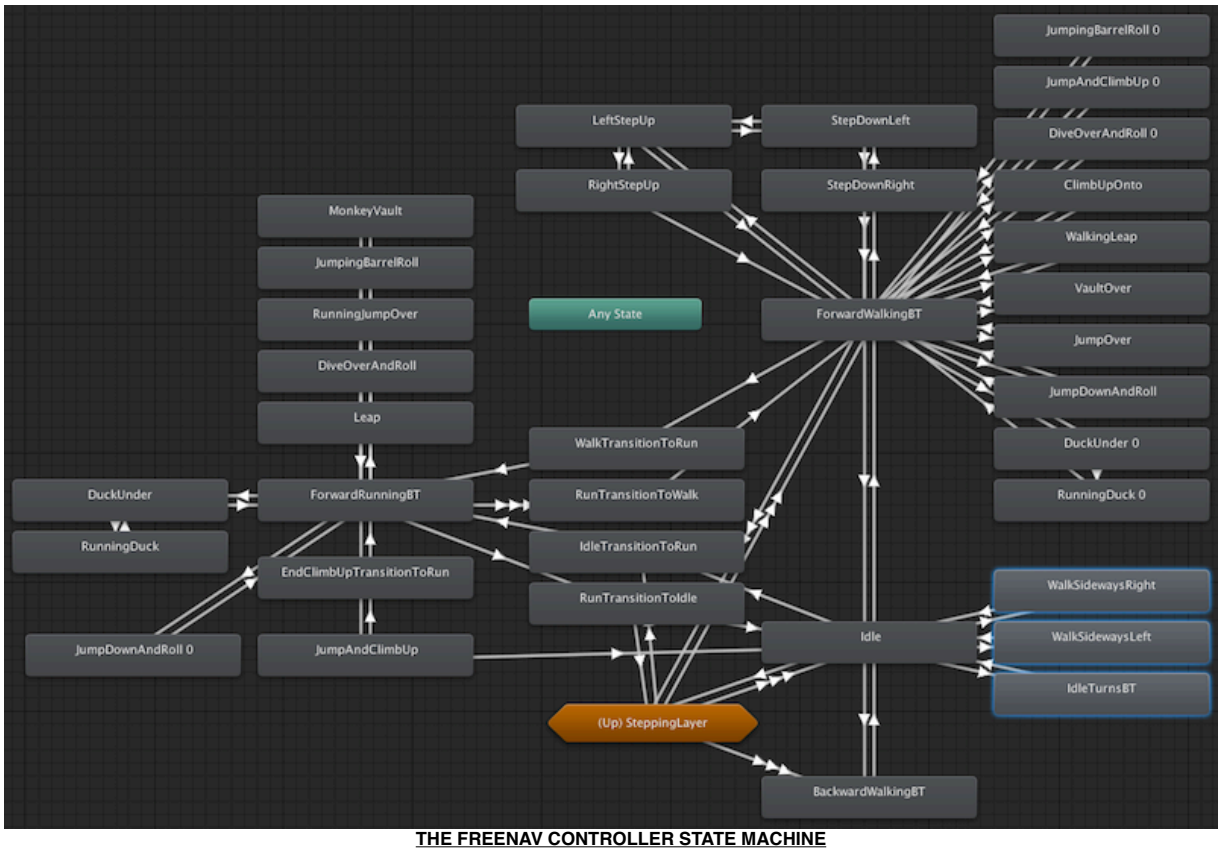
With a broad style of game mechanics in view a modular system was set up where keyboard or axis input determined the navigation and stance movements of the lower body and mouse left and right buttons controlled upper body movement via a procedural IK system of constraints. This allows the system to be extended to accommodate any weapon or style to be used with very few or no further animations needing to be created. By balancing of variables or adjusting spline shapes in each of the modes a wide variety of gameplay styles can be tweaked to accommodate the needs of the game mechanic attempting to be achieved.

The Katagon Procedural IK System was developed modularly and with extensibility foremost in mind without the need for creating numerous new animations to handle a new weapons or spellcasting ability for example. All parameters are exposed to allow scripting and storing series of parameters suitable to your game mechanics and character expression as presets. The first release of the Katagon System includes a sword and shield Melee Mode and a right hand Handgun Mode for the Player Avatar. The Melee Mode is based on constraining the IK Goals to a Katagon Arc Spline with controls for Radius and From and To, allowing you control of swing arc and elbow bend during Swing and Block execution. (Chris West of the amazing MegaFiers suite of plugins for Unity was kind enough to rewrite his spline system specifically for inclusion in the Katagon System). The Handgun Mode is based on the use of an aiming constraints system controlling positioning and rotation of IK Goals. Using either the constraint to spline or constraint to an aiming objects system, or a combination of both, a spectrum of weapons handling including kung fu staff, magical staff, two handed sword, shoulder or hip fired two handed rifle, spear, axe and others as well as handing complex hand gestures such as drawing runes in the air for spellcasting, martial arts strike and block bare handed techniques. Being procedural they can be adjusted, tweaked and scripted to allow variation as necessary to achieve the style you are seeking for your game character.

The Mecanim Controller is at the heart of the Stance and FreeNav system for moving your character through your game world environment. For further defining of the style you wish to achieve you may want to adjust some of the default transitions.



THE STANCE CONTROLLER STATE MACHINE



THE FREENAV CONTROLLER STATE MACHINE

The FreeNav Controller is entered from the Stance Controller by setting the FreeNav boolean to true.

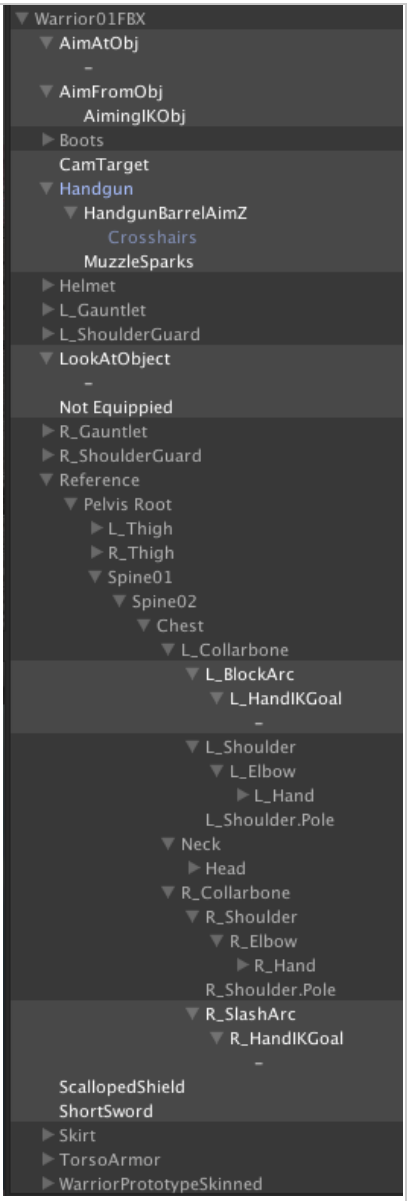
```
animator.SetBool("FreeNav", true);
```

Conversely one exits the FreeNav Controller by setting FreeNav to false.

```
animator.SetBool("FreeNav", false);
```

In the KatagonPrefabs->AddToYourCharacter folder are several prefabs that assist the procedural IK system in reacting to changes in input. You can of course replace the weapons with your own after some minimal setup for initial axis positioning and rotation and mounting offsets.

PLAYER AVATAR HIERARCHY SETUP



The highlighted objects above are additions to the original hierarchy and can be found in the Katagon Prefabs folder

Setting Up Your Avatar Hierarchy

To set up a new Player avatar you need to drop a few game objects into the Hierarchy of the avatar. The objects are highlighted in the screen capture to the left.

Assisting Objects

AimAtObj - This is used in Handgun Mode for aiming. The '-' child object is a visual aid. It can be turned off or removed from the Hierarchy.

AimFromObj - The AimFromObj gets constrained to the Avatars right shoulder in Handgun Mode. The ***AimingIKObj*** as a child of the ***AimFromObj*** sits .45 units forward on the local Z axis and controls the aiming hand position and rotation.

CamTarget - This is the object the Main Camera looks at. Currently it sits 0.65 units forward of the local Z axis of the avatar root at approximate eye level 1.6 units on the Y axis and is centered at 0.0 on the X axis.

LookAtObject - This is used to control the upper body LookAt and rotates the torso and head. The '-' child object is a visual aid. It can be turned off or removed from the Hierarchy. It's initial position should be 2 units forward of the avatar on the local Z axis and at approximate eye level, about 1.8 on the local Y axis and centered at 0 on the local X axis

Not Equipped - This is an empty GameObject placeholder for dropping in slots that have no armor or weapon for that slot selection.

L_BlockArc - This arc is for IK Goal constraining the blocking arm IK in Melee Combat Mode. Place at L_Shoulder position as a child of the L_Collarbone.

L_HandIKGoal - The IK Goal constrained to the L_BlockArc placed as a child of the L_BlockArc.

R_SlashArc - This arc is for IK Goal constraining the swinging arm IK in Melee Combat Mode. Place at R_Shoulder position as a child of the R_Collarbone.

R_HandIKGoal - The IK Goal constrained to the L_SlashArc placed as a child of the L_SlashArc.

Weapons Objects

Handgun - Handgun Mode - To set up a Handgun for use with Katagon, sit the transforms axis at the top of the handgrip and rotate it's axis so that Z points down the barrel and Y points upward. The ***HandgunBarrelAimZ*** obj sits in the forward part of the handgun barrel and is used to point the Bullet in the direction the handgun barrel is pointed at. The ***MuzzleSparks*** object is a particle emitter triggered at each shot. Also included is an in-scene ***Crosshairs*** target which can be toggled on if you wish a visual aid in game.

Sword & Shield - Melee Mode - Place the sword and shield in the Avatar Hierarchy and use the WeaponsMounting script to set the offsets for position and rotation so it sits properly and is constrained to the hand or arm appropriately.

Most objects are simply empty game objects that use the Transform to get and set input and output. The L_BlockArc and R_SlashArc are KatagonShape splines. (written by Chris West and based on his MegaFiers/MegaShapes spline system). They are placed in the rig hierarchy at the respective shoulder joint positions and use the collarbone rotation to set it's localRotation or localEulerAngles from as it's immediate parent.

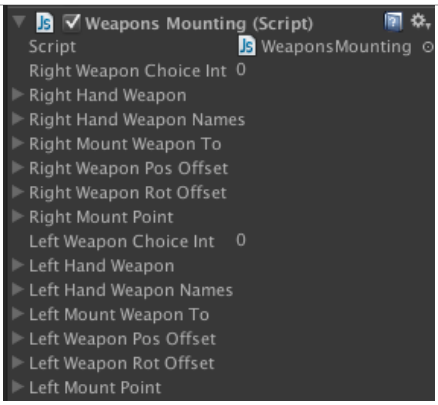
Most empty gameObject prefabs have a visual aid for use during development for tracking visually. They can be turned off or removed from the Hierarchy.

Included in the Katagon Procedural IK System are some helper scripts for mode switching, weapons equipping and mounting and armor outfitting. You may have a more advanced or already developed inventory system, in which case you can deem which scripts are not necessary. They were added to get developers off to a flying start and as examples of scripting the system.

PLAYER AVATAR SCRIPTS

Controller Mode Switcher

The ControllerModeSwitcher.js handles switching between modes by turning scripts off and on appropriately. The switching is handled via keyDown input. Currently Melee Combat mode is switched on using the 'C' key. Free Navigation mode is turned on using the 'F' key. Handgun mode is entered using the 'G' key. As more modules are added to handle other combat modes such as archery or two handed sword or sword and dagger for example, these can then be added to this script and switching controlled via any input the user wishes.



You can also set the positional and rotational offsets in Game mode by manipulating the Vector3 variables in the Inspector, recording the numbers and applying them in Scene mode.

Weapons Mounting

Weapon mounting is controlled by WeaponsMounting.js. BuiltinArrays of weapons can be set up for either hand with mounting points on the rig and rotational and positional offsets per weapon. The weapon currently used is controlled by the int variables 'rightWeaponChoiceInt' and 'leftWeaponChoiceInt'. This chooses the corresponding weapons and their offsets and mounting points from the matching arrays.

- Drop 'rightHandWeapon' or 'leftHandWeapon' into array slot.

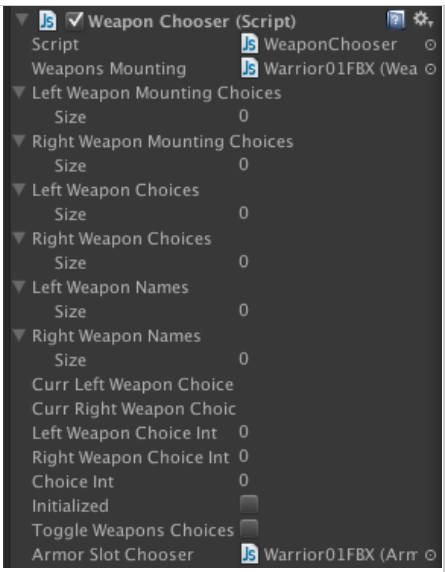
- Set the mounting point by dropping the appropriate rig joint into the 'rightMountWeaponTo' or 'leftMountWeaponTo' corresponding array slot. *(For example a sword or gun gets mounted to the rightHand joint where a shield would more properly mount to the elbow joint of the rig.)*

- Set the rotational and positional offsets set so that the weapon sets properly in relation to the joint axis it is mounted to. *(To get the offsets position the weapon and rotate it appropriately. Since offsets are derived locally, drop the weapon as a child of the rig joint it will mount to and then record and apply the offsets to the weapon positional and rotational offsets. After offsets are recorded the weapon can return to it's former place in the hierarchy.)*

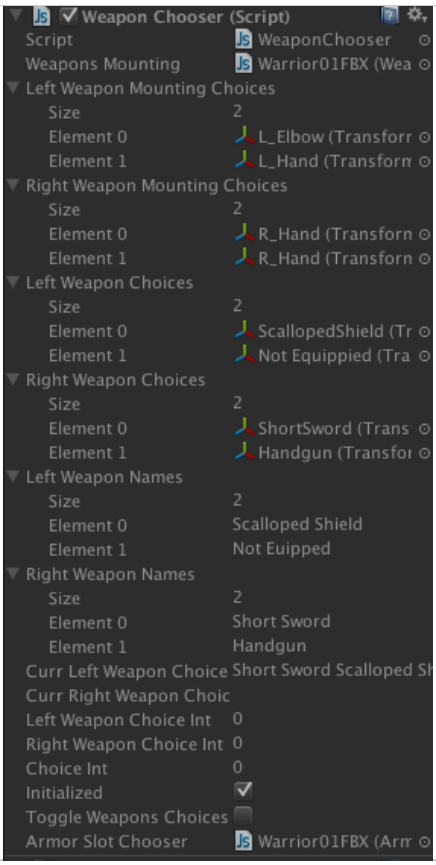
- The weapon will remain constrained to the mounting point. *(The weapon does not have to be in the rig or avatar hierarchy to be mounted nor do you need to skin it to a bone.)*

You can set up the keys to control mode switching. In the screenshot above I am using the 'c' key to switch the Animator to Stance and the procedural IK to Melee mode. I could use a different key to switch either Stance or Melee and gain individual control over how and when they switched navigation and combat mode. In Handgun Mode in the above screenshot you can enter Handgun Mode by tapping the 'g' key and remain in whatever navigation mode you are currently using. Use the Mode dropdown menu in the Inspector to choose the Mode your game starts in.

Due to the way WeaponsMounting is handled it is not necessary to skin the weapon and keep it in the rig hierarchy. This allows easy pickup and/or dropping of the weapon and no setup in a 3D app to skin the weapon to the avatar rig. Some prior setup to place the axis at the proper pivoting point may be necessary. For example, you would want the axis of a sword in the handgrip or the axis of a handgun in the pistol grip..basically in the position the weapon would naturally pivot about when in use. A sword would obviously not pivot about it's blade tip for example.



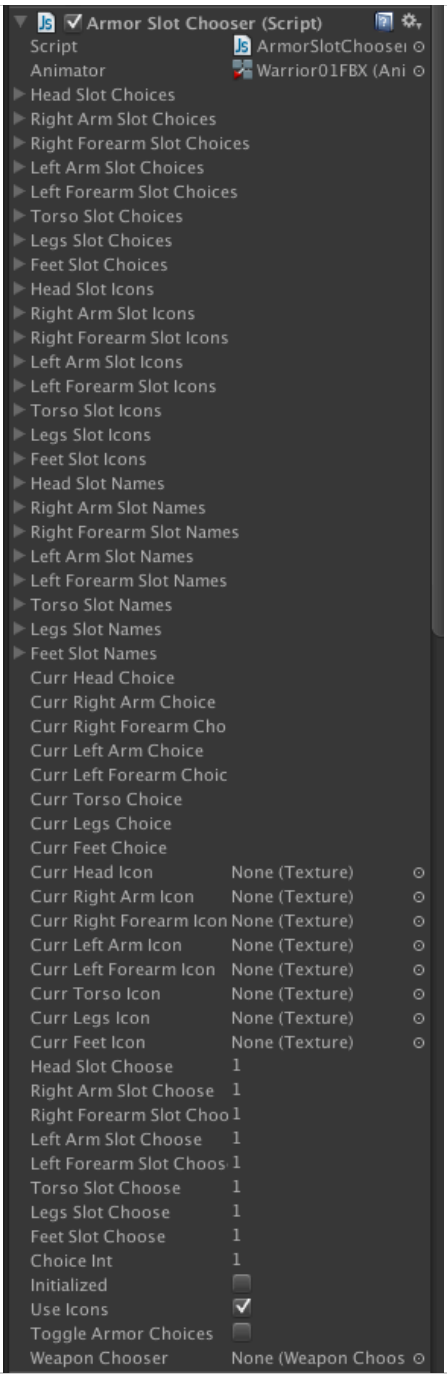
Weapons Chooser GUI Interface *- Above is the WeaponsChooser in Scene mode. It automatically fills it's arrays from the WeaponsMounting arrays as in the figure to the right which was taken in Game mode.*



Weapons Chooser

WeaponChooser.js controls a GUI interface for weapons switching in game. When toggled on it will display a button that will cycle through the pairs of objects used as weapons. For example a sword and shield for right and left hand pair versus a Handgun and Not Equipped for a right and left pair.

You can set up as many weapons pairs as you need. If a hand is not using a combat object then use the 'Not Equipped' prefab. This is done in the WeaponsMounting Inspector while the WeaponsChooser supplies a GUI interface by copying those arrays from the WeaponsMounting Inspector. The current GUI simply toggles through the array cycling back around when at the end of the array.



Armor Slot Chooser

The ArmorSlotChooser.js script presents a GUI interface for turning off and on armor or clothing for various parts of the avatar. You can use Icons or textual buttons. Icons are currently set for a 128x128 pixel size. To set it up simply drop all matching components into their corresponding slots. For slots without any equipment use the empty 'Not Equipped' GameObject.

You can also use icons at 128w x 128h for each armor slot icon. Set the useIcons boolean to true and drop the texture in the corresponding array slot.

If you want to change equipment per armor slot via scripting, each armorSlot has a Switch function that you send an integer corresponding to that piece of equipments array position. For switching the armor equipment set for example from set [0] to set [1].

```
//set the script
var armorSlots : ArmorSlotChooser;

//define the array position
var choice : int = 1;

//call the switching functions
armorSlots.SwitchHeadChoice(choice);
armorSlots.SwitchRightArmChoice(choice);
armorSlots.SwitchRightForearmChoice(choice);
armorSlots.SwitchLeftArmChoice(choice);
armorSlots.SwitchLeftForearmChoice(choice);
armorSlots.SwitchTorsoChoice(choice);
armorSlots.SwitchLegsChoice(choice);
armorSlots.SwitchFeetChoice(choice);
```

You could also send a different integer to each switching function to mix and match your equipment based on gameplay or pickups, in game purchases and the like.

Stance Controller

There are eight basic directions that can be taken from any stance being the current center There are nine stances that can be stepped into. These are, in clockwise order, "MoveForward", "MoveForwardRight", "MoveRight", "MoveBackwardRight", "MoveBackward", "MoveBackwardLeft", "MoveLeft", "MoveForwardLeft" with "IdleMid" being the center position. These are controlled currently by tapping the WASD keys.

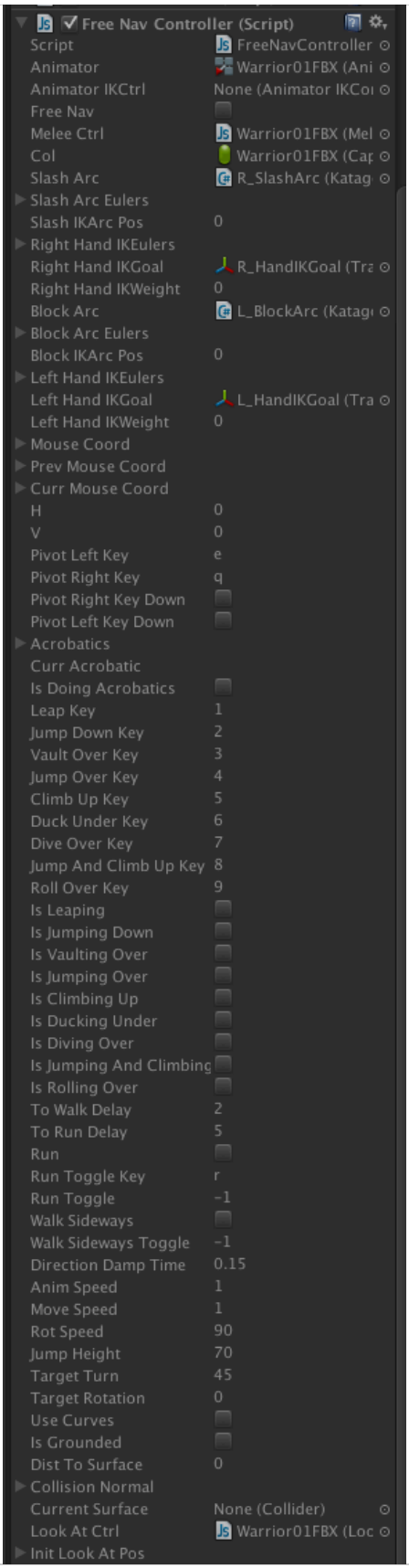
W = MoveForward
A = MoveLeft
S = MoveBackward
D = MoveRight
WA = MoveForwardLeft
WD = MoveForwardRight
SA = MoveBackwardLeft
SD = MoveBackwardRight
AD = IdleMid

If the key controlling the current stance remains down or is pressed again whilst in the stance associated with it it will shuffle in that direction. Pivoting is controlled by the 'Q' and 'E' keys while in Stance Mode.

Stance Height is controlled per stance from the stanceHeights array. 0 is the lowest, deepest form of the stance and 1 is the highest form of the stance. stanceHeights[0] controls stance height for controllerDirectional[0] etc. through each array respectively using float values between 0.0 and 1.0.They can be set up in the array and will be used when the corresponding Stance is current. Conversely they can be scripted to be set randomly or varied based on any other input normalized from 0 -> 1.

Stance to stance transition speed is controlled by the stanceSpeed variable. The advantage of using an upper body procedural IK system is that the lower body speed can be controlled independently without generating new animations. By balancing the upper body speed variables with the stanceSpeed you can fine tune the performance of your character.

In Stance mode the 'lookAtPos' stays where you last had the right mouse key down. This allows turning the upper body to face an opponent and remaining turned to combat with the left mouse down or use them in tandem to combat in an arc forward of the avatar. In FreeNav mode the 'lookAtPos' will drift back to center once the right mouse button is up.



FreeNav Controller

FreeNav or free navigation mode is accessed using 'F' key down input. Standard Horizontal and Vertical input axis are used. The lookAround is controlled by the right mouse down button.

W = forward
WA = forward turning left
WD = forward turning right
S = backward
SA = backward turning left
SD = backward turning right
A = turn left on the spot
D = turn right on the spot

Use the 'R' key to toggle between Walk and Run modes. Included are several acrobatic movements. "Leap", "JumpDown", "VaultOver", "JumpOver", "ClimbUp", "DuckUnder", "DiveOver", "JumpAndClimbUp", "RollOver". These are controlled by the number keys 1 thru 9 respectively. They will execute from either the Walk or Run mode during FreeNav.

In FreeNav mode the Horizontal and Vertical InputAxis control Direction and Speed as opposed to the stance to stance WASD tapping controls in Stance Mode.

In FreeNav navigation mode the overall speed of the animations is controlled with 'freeNavSpeed'. It can be set in the Inspector to use during Game mode or can be scripted. For example you may want to accelerate during running so you would write...

```
var runSpeed : float = 1.5;
var walkSpeed : float = 1.0;
var acceleration : float = 0.1;
if(run && freeNavSpeed < runSpeed){
      freeNavSpeed = freeNavSpeed + acceleration;
} else if (!run && freeNavSpeed > walkSpeed){
      freeNavSpeed = freeNavSpeed - acceleration;
} else {
      freeNavSpeed = 1.0;
}
```

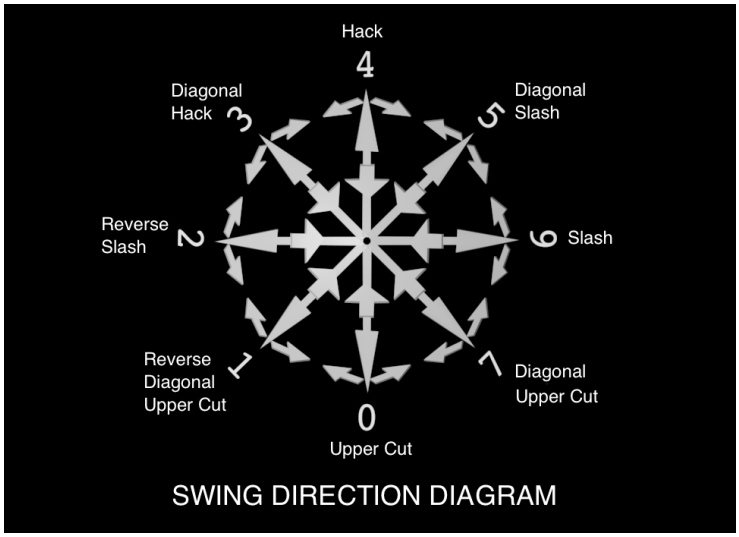
Booleans are set during acrobatics and the isDoingAcrobatics boolean is set to true and set back to false upon completion of that acrobatic.

If 'run' is not toggled or set to true then in FreeNav mode the Controller walks if 'V' (Speed) is not equal to zero. The Run Toggle Key can be set in the Inspector.

JumpHeight controls an upward force on the character rigidbody. A 'JumpForce' Curve is applied to each animation for the acrobatics that can be altered to suit the heights of obstacles or gameplay style. Set 'useCurves' to true and the JumpForce' Curve is sampled during that animation and multiplied by the JumpHeight. This applies a variable upward force on the avatar's rigidbody.

Melee Combat Mode

Melee Mode is for sword & shield combat and is controlled from the MeleeCombatController.js script.. Swinging is controlled by leftMouse movement, lookAround is controlled by right mouse movement and blocking is toggled by the 'z' key. There are 8 swings one can use derived from the cross and diagonals. From the avatar point of view and going clockwise from the bottom starting point are "upperCut", "reverseDiagonalUpperCut", "reverseSlash", "diagonalHack", "hack", "diagonalSlash", "slash", "diagonalUpperCut".



These are set up so that if you have just completed a swing then the next swing is either the swing it is currently at, which is opposite to the previous swing, or the next swing is to either side of the of the swing it is currently at. The swing chosen is derived from the mouse movement with left mouse down. If allowed it then uses that swing.

The right mouse down controls the torso movement left and right and up and down. The right mouse controls based on screen position of the mouse pointer. Upper left and the avatar looks to the upper left and lower right the avatar looks to the lower right. Since the mouse move coordinate for the left mouse controlling the swing is local and based on previous mouse coordinates one can control swinging from any screen position. In combination the left and right mouse can turn the torso so that swinging can be made to happen in a semi-circle in front of the avatar.

Blocking is toggled using the 'Z' key down currently (*all inputs can be changed in the Inspector so you can set up key controls in any manner you are comfortable with*). The 9 block positions are "highLeft", "highCenter", "highRight", "midLeft", "midCenter", "midRight", "lowLeft", "lowCenter", "lowRight". These block positions are based on mouse coordinates and the sector of the screen it is in. Basically it can be pictured as dividing the screen into a 3x3 grid of equal sized rectangles.

Right Hand Start and End Euler Angles per swing can be set up if you do not want to use the defaults.

You can control blockSpeed and swingSpeed for example to be slower to react to fatigue or damage and quicker for speed or power gains and to adjust to set up a style you find fluid.

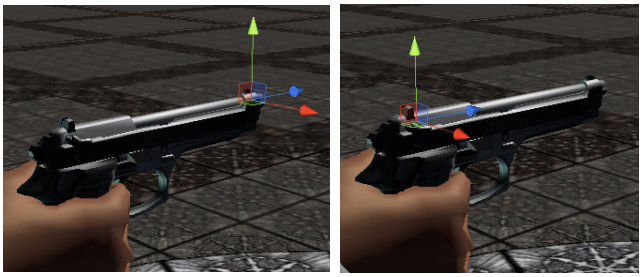
Handgun Mode

Handgun Mode is for use with a right hand handgun. You aim with the left mouse down and fire with the space bar. The aimAtSpeed can be adjusted to suit the gameplay preferences. In the CombatFollowCam script on the Main Camera are two booleans - 'shooterCamMode' and 'useShoulderCamMode' - that control the camera behavior in Handgun Mode. If 'useShoulderCamMode' is true then when the left mouse button is down the camera will be looking over the right shoulder and aiming at the ray cast from the mouse position targeting crosshairs into the scene.

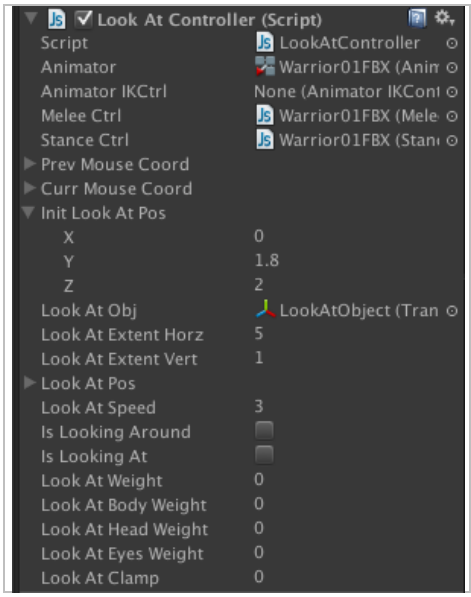
The Firing Button can be set here *(it is currently using the space bar)* as well as the Firing Interval *(how long before the next shot can occur)*, FirePower *(the initial force the projectile is ejected with)* and the HandgunRange *(the range at which it is no longer effective)*. The MuzzleSparks object, a child of the Handgun object placed at the gun barrel exit, is also the AudioSource for the bullet sound effects. The MuzzleSparks Particle Emitter can be adjusted beyond its basic defaults.

The Bullet Object to be used is also dropped into its Inspector slot. *(make sure the bullet has a BulletBrain script attached)*. The bullet has it's own raycasting system so it will not pass through objects but register a hit if they are in the bullet trajectory path.

When the bullet is Instantiated as a 'shot' it will Destroy itself upon hitting an obstacle or when the firingInterval has passed. The shot distance is also tracked here and queried by the BulletBrain.



Axis setup for the Handgun for the MuzzleSparks and HandgunAimingObjZ objects respectively.



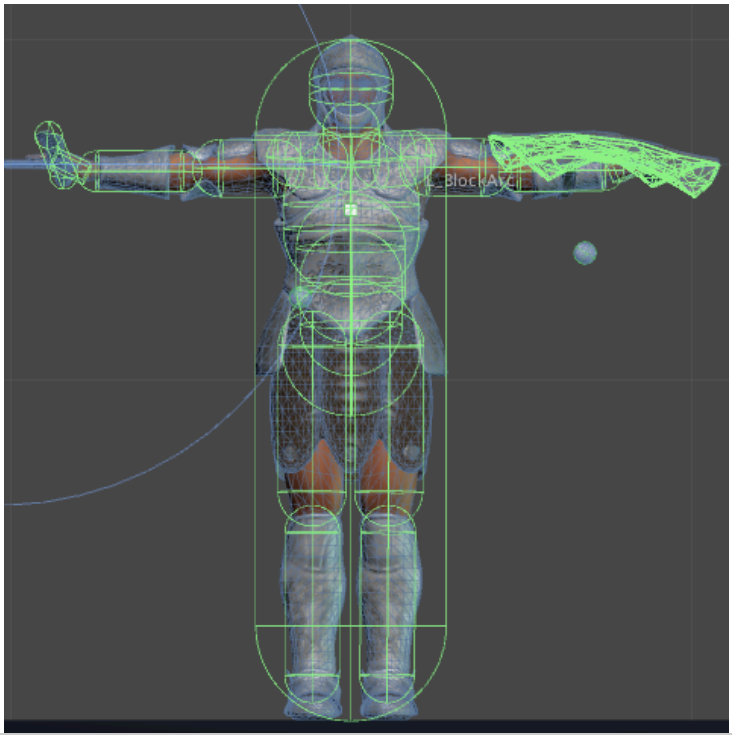
LookAt Controller

The LookAtController.js script handles positioning of the LookAtObj which controls the upper body twist and bend. The right mouse down controls lookingAround by moving the LookAtObj left and right and up and down. Mecanim uses this position to rotate the torso and head of the rig to follow the LookAtObj. The lookAtSpeed variable is exponential (lookAtSpeed* lookAtSpeed*Time.deltaTime). This control is essential for setting up upper body motion appropriate to the gameplay mechanic or style you are attempting to achieve.

All the objects you want to receive damage from weapon strikes are dropped into the hitZones array in the Inspector. When in Game mode an array for zoneDamage is created that tracks damage sent from the NPCWeaponsStrike script on the opponents weapon. Note I include the avatar CapsuleCollider and weapons in hitZones.

Damage Control

The DamageControl.js script records triggered collisions from weapons and bullets. Each of the Rig Joint transforms are dropped in the slots. The Rig Joints themselves have a Kinematic Rigidbody and a Collider as components for registering a hit. The actual hit is recorded from scripts on the weapon or projectile and sent to the DamageControl script of the avatar root transform of the object Collider in the rig. The 'damage' can then be monitored and extracted to use to affect other variables such as health or causing a death animation to occur or applying a force to a ragdolled joint chain for simulating a hit. The Figure below illustrates the setup of colliders on the Avatar Rig for recording triggered collisions.



Animator IK Controller

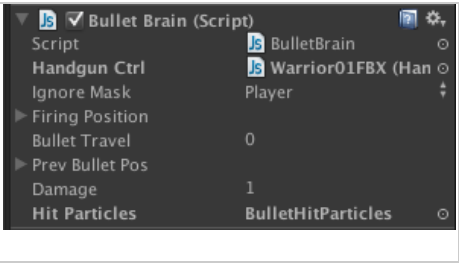
The AnimatorIKController.js script is the hub of the IK controller subsystems. Once calculations are made for IK Goal positioning and rotation and lookAtObj position they are sent to this script for handling by Mecanim. Also included here are weighting for other layers, in this case the Finger Pose layerWeights.

MAIN CAMERA SCRIPT

Combat Follow Cam

The CombatFollowCam.js script handles the navigation camera movement. The damping variables control the speed of translations and rotations. During navigation it generally follows the Avatar. For 'shooterCamMode' the camera can be set so that when the left mouse button is down, which is used for aiming with screen crosshairs, it will sit behind the right shoulder of the Avatar if shooterCamMode and useShoulderCamMode is true.

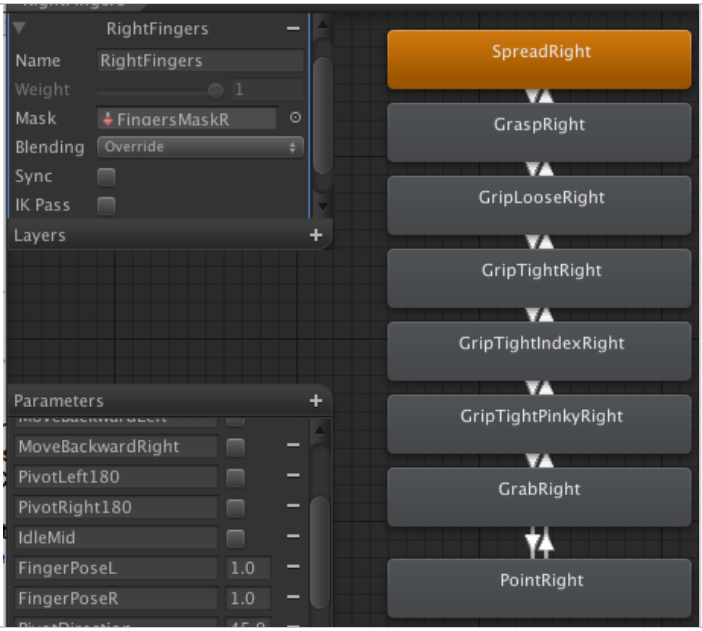
BULLET BRAIN SCRIPT



Bullet Brain

The BulletBrain.js script is attached to the "Bullet' GameObject, which can be placed anywhere in the Hierarchy. It is Instantiated by the Handgun Mode in the HandgunController script. When fired it sends out a Raycast along it's forward Z axis to determine if any obstacles are hit. This will stop the bullet from 'going through' an object due to the amount of bullet travel from frame to frame not registering an object in its path due to it not colliding. Upon colliding it Instantiates a particles prefab at the hit.point using the hit.normal and Vector3.Reflect to position and rotate the particle system.

FINGER POSES



Finger Poses

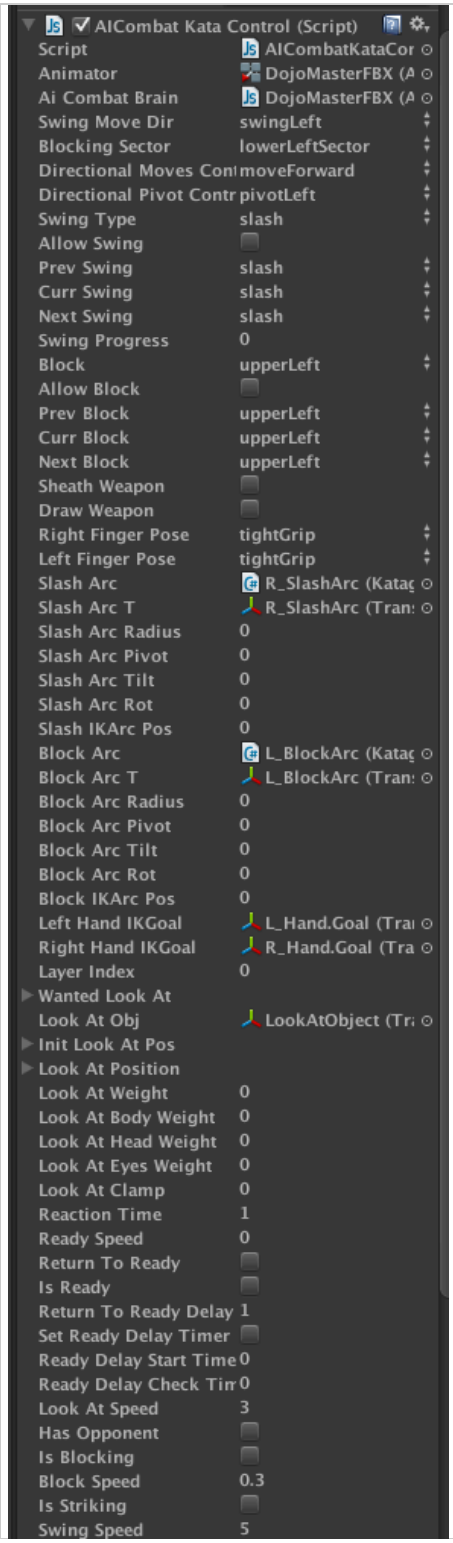
Finger poses are set up in separate State machine Layers and finger poses set using an integer for both left and right finger poses..

0.0 - 0.99 = Spread
1.0 - 1.99 = Grasp
2.0 - 2.99 = Grip Loose
3.0 - 3.99 = Grip Tight
4.0 - 4.99 = Grip Tight Index
5.0 - 5.99 = Grip Tight Pinky
6.0 - 6.99 = Grab Right
7.0 - 7.99 = Point

These give the hand pose a range of positions that should accommodate most setups.

The FingerPoses system can be extended by adding further finger poses to the FBX file, setting the range for the added animation and then setting up the transitions as per the default set.

AI - DOJOMASTER



AI Combat Kata Control

The AICombatBrain.js script tracks the opponent, attempting to turn and pivot and move from stance to stance to keep the Player Avatar at a default Distance and within the Cone Of Attack. It sends it's decisions forward the the AICombatKataControl.js script to handle the Mecanim transitions.

The DojoMaster was included to provide an opponent for the Player character during development. Whereas the Player script setup is modular, the AI scripts consist of a 'Brain' and a 'Body'. The AICombatKataControl acts as the body and moves the DojoMaster about using signals/inputs from the brain, AICombatBrain. It makes use of the same Animator Stance and FreeNav Controller as the Player avatar, though the Hierarchy setup is slightly different.

AI Combat Brain

The AICombatBrain.js script tracks the opponent, attempting to turn and pivot and move from stance to stance to keep the Player Avatar at a default Distance and within the Cone Of Attack. It sends it's decisions forward the the AICombatKataControl.js script to handle the Mecanim transitions.

The AICombatBrain tracks the opponents position and whether their weapon is moving and if that proximity is deemed to be an attack. If so it goes into Retreat mode till beyond the defaultDist and then reverts back to Attack mode.

The checkInterval variable is the time interval between distance checks and angle to opponent calculations for determining the inputs passed to the AICombatKataControl component.

The coneOfAttack determines the cone projected forwards within which the opponent can be attacked. 1.0 is 90 degrees left and right and 0 is exactly forward.