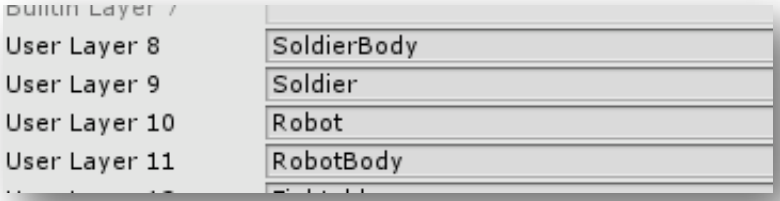
DOCUMENTATION

Importing project and using shooter prefabs

Tags and layers are used for various purposes by shooter agents. After importing project, tags should be automatically imported by unity. If not, try restarting unity or you can add them yourself.

Make sure shown tags are available in the project.

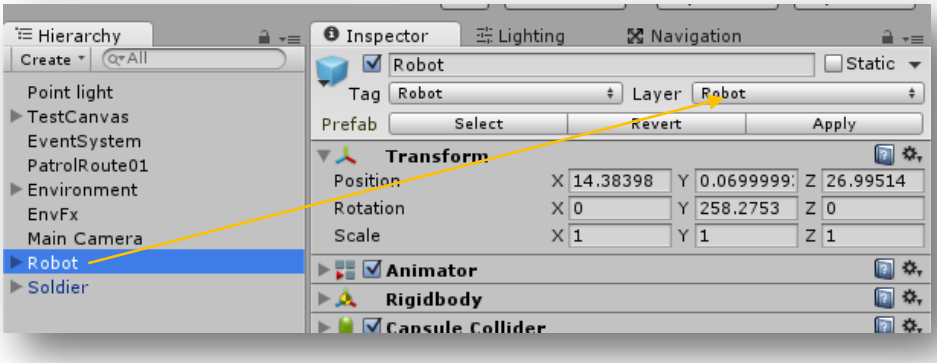
添加必须要的tag 与 Layer



You will need to add these layers.

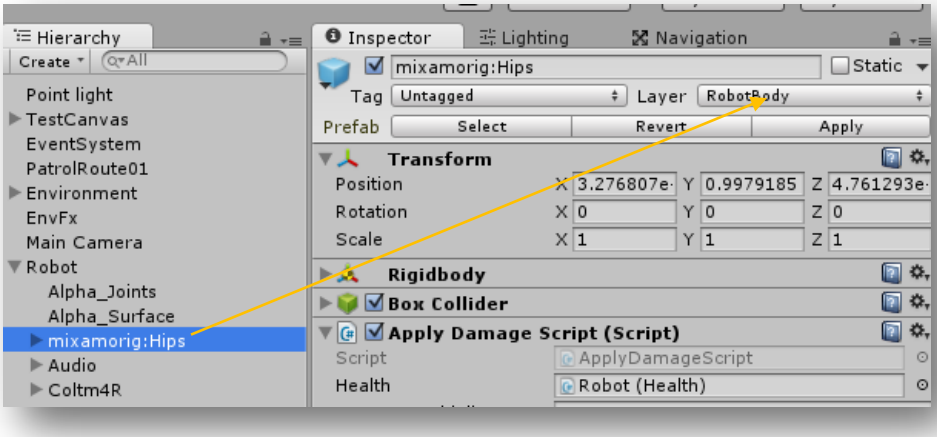
After adding these layers open example scene and set Robot shooter's layers as shown below.

Select main Robot transform and set its layer as Robot, it will ask you if you want to change children layers, hit No.



设置Robot Layer

Select hips like shown, set its layer to RobotBody,this time hit yes to change children layers.

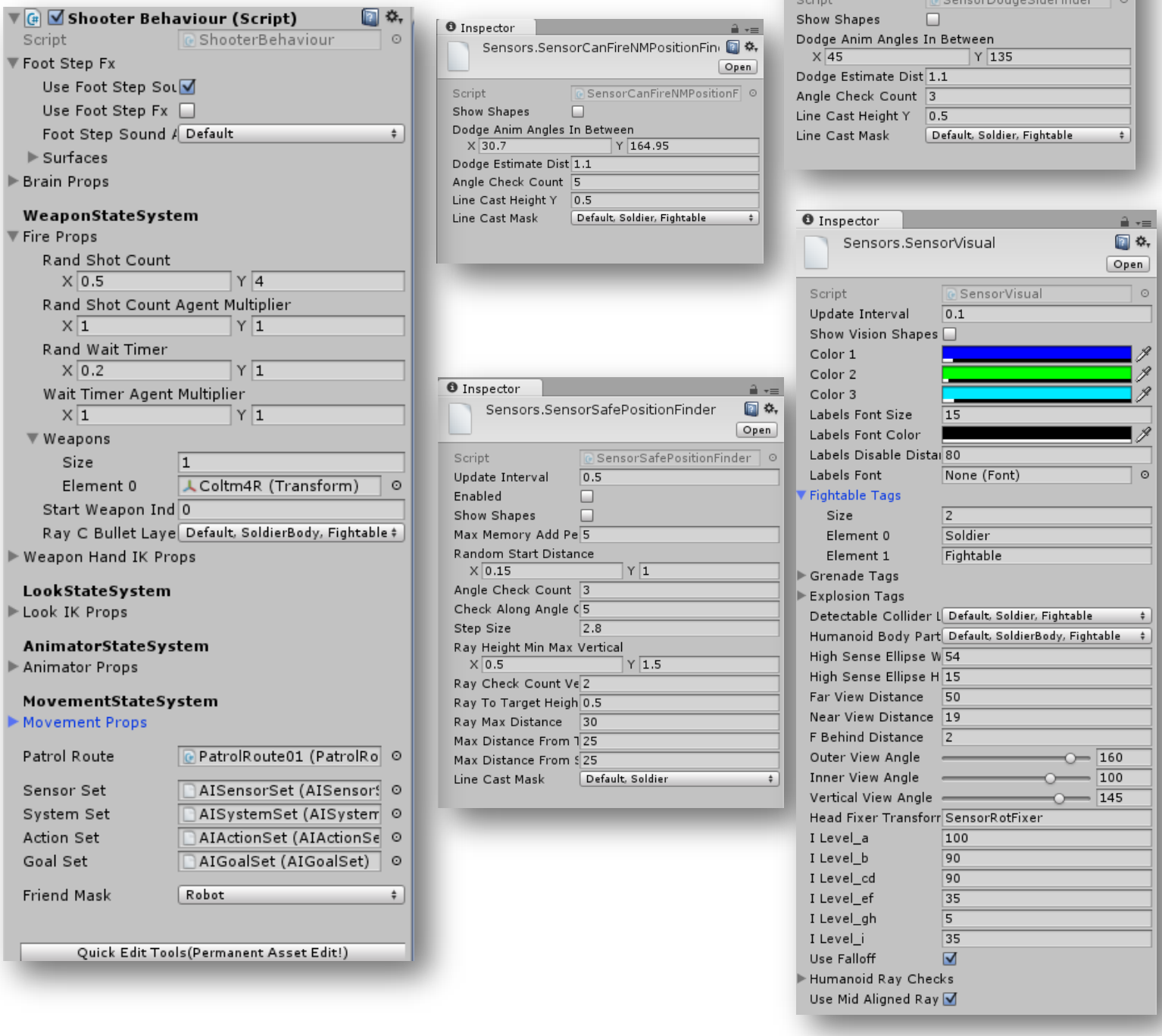


**** After that select soldier prefab from hierarchy and change its layers like you did with robot, this time set them as Soldier and SoldierBody.***

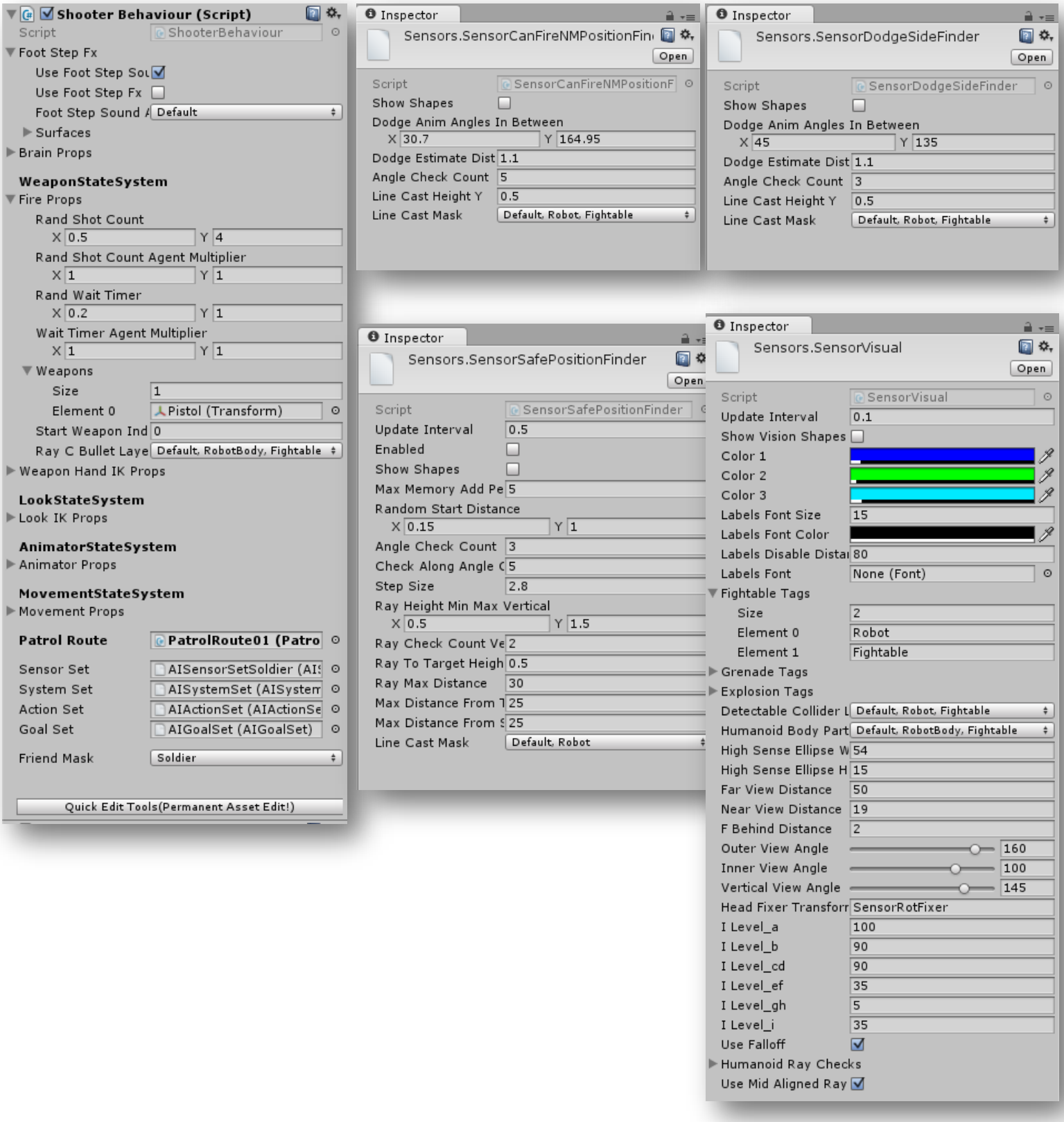
You may also need to set sensor detection layers for shooters. If they are not set automatically (which most probably they won't), set them by looking at pics below.

Robot Sensor Layers / Other Layers

Navigate to ***Assets/IntenseAI/Prefabs/Aıs/Robot/SensorScRobot*** them modify corresponding sensors using inspector. Then modify ***ShooterBehaviour*** component of ***Robot***.



Navigate to ***Assets/IntenseAI/Prefabs/AIs/Soldier/SensorScSoldier*** them modify corresponding sensors using inspector. Then modify ***ShooterBehaviour*** component of ***Soldier***.



Scriptable Objects

Scriptable objects are basically serialized scripts that can be customizable by inspector and works like an asset. In this system shared tools are achieved using this feature of unity.

Tool Maker Window

This window is used to easily find and create a scriptable object to customize and use with agents.

While ***'Create Single'*** tab is selected, you can search entire project for Sensor-System-Action-Goal. Then when you hit create button, that tool will be Instantiated as a prefab in the project panel under selected folder.

-To create your own tools (scriptable objects), take a look at one of already written scripts and when you are done get back to tool maker window to create that tool for your ai.

While ***'Create Set'*** tab is selected, you can Instantiate a holder for your scriptable objects so that *ShooterBehaviour* script will be able to use them.

Tip- All sets are copied on start and brain works using copied temporary set of tools to prevent unwanted modification of serialized properties.

Scriptables

Tip-Goap system may be hard to understand at first. This documentation is not enough to explain whole goap logic. I recommend checking references section which is in the end of this document.

Actions

Any class that is derived from ***AIAction*** class 'except abstract classes' is considered as action scriptable object / tool that can be instantiated using tool maker window.

Basically an action is a step to reach an ***AIGoal***.
You can find detailed information about ***overridable*** functions and useful ***properties*** in ***AIAction*** script.

Goals

Any class that is derived from ***AIGoal*** class is considered as a goal tool that can be instantiated using tool maker window.

You can find detailed information about ***overridable*** functions and useful ***properties*** in ***AIGoal*** script.

Sensors

Classes that are derived from:

-***AISensorPolling***, ***AISensorRequest***, ***AISensorTrigger***

is considered as sensor tool that can be instantiated using tool maker window.

Sensors used to fetch information from the world and add those informations to memory.

Polling sensors are good if you need to check things regularly with an interval time.

Trigger sensors are event driven sensors that are fired when something happens. Like when a bullet damage is taken.

Request sensors are good for fetching information as needed by actions.

You can find examples for each of these types (to use as a reference) in the project.

Systems

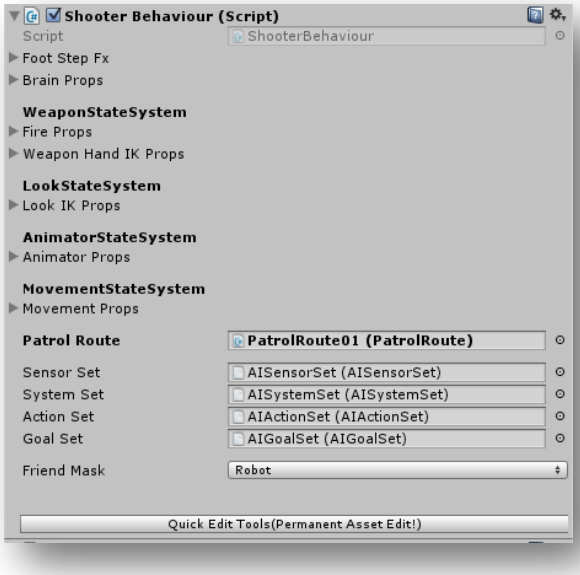
Classes that are derived from ***AISystem*** is a scriptable object that can be instantiated using tool maker window.

Systems are mostly used to examine informations (of memory) that are added by sensors.

You can find detailed information about ***AISystem*** functions and useful ***properties*** in ***AISystem*** script.

Tip: You can watch this tutorial to learn basics of scriptable objects in this project:

https://youtu.be/3V1mQ-DaZKs



StateSystems

You can see in the picture there are couple of statesystems, for each of these systems there is a serialized class to let you customize shooter state systems.

Tip-A state system is a controller for an agent. Actions use these systems to give specific orders to control agents. You can examine further by checking state system scripts.

Tip- Currently there are 4 statesystems that controls the shooters.

You can use these statesystems to create many other actions by giving specific commands to shooter.

Weapon StateSystem

The weapon shooting properties. Like weapon hand properties and gun firing properties. This can be used by actions to make the shooter agent start firing/stop firing etc...

Look StateSystem

This system uses Unity IK to modify head/body look at position. There is a sublevel system (in this statesystem) to make this beautiful look IK even better and flexible. One flexibility is, an humanoid agent can switch look positions smoothly without having to worry about body weights. Currently, only horizontal looking is defined.

Animator StateSystem

As you can understand by its name, this system is used to animate a character with Animator and get the results if needed, by actions. This is used in many of the actions. Take a look at how this works in action scripts. Basically you give an animate command and a finish state, you ask this system if finish state is reached. If you don't want your agents to stuck in an action, you need to use this system properly, or you need to add an interrupt transition to get back to your last state or you can set your action as an uninterruptable action.

Movement StateSystem

This can be used to send shooter to a specific position, or even better you can tell him to go to/turn to current enemy position, you can even use *navmeshpaths* but this is not used by default for it is expensive. Not much to say, system is easy to use. This system is using *Unity navmesh* to calculate paths.

Tip -Shared props script is used to connect the memories between agents. So that if different types of agents need to communicate later (like between a shooter and a dog), this MonoBehaviour will be used as a connector.

Tip-Don't set your character's main gameobject's layer similar to the layer name you use for character bones. You can look at an already made Shooter Agent's layer naming as a reference. For ex. I have set Robot's main gameobject's layer as 'Robot' and all of the robot's child bones' layers are set to RobotBody.

Tip- Using ShooterBehaviour component's Friends field You can show ai, which layers will be used to detect as friends, so that ai will listen messages from them.

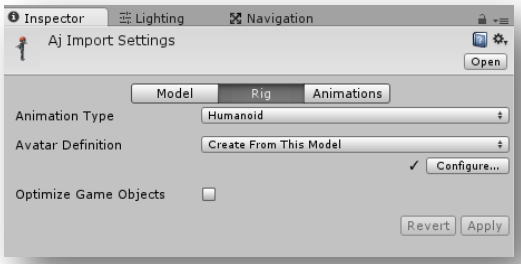
Shooter Specific

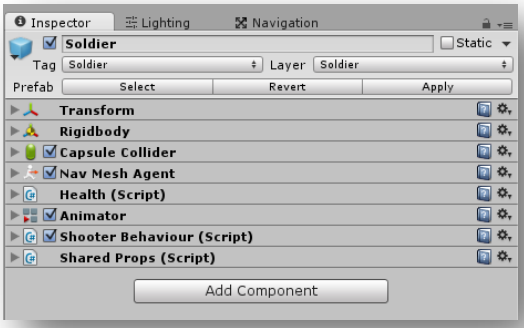
Importing a new character

While importing a new character model as a shooter you need to set up specific things properly so that the state systems will work as expected.

Tip-First, your character should be humanoid and it must be Unity IK compatible. After making sure that your character is capable of them, you can follow these steps to import your character.

1-) Make sure you imported character as humanoid and import options are matches this picture.





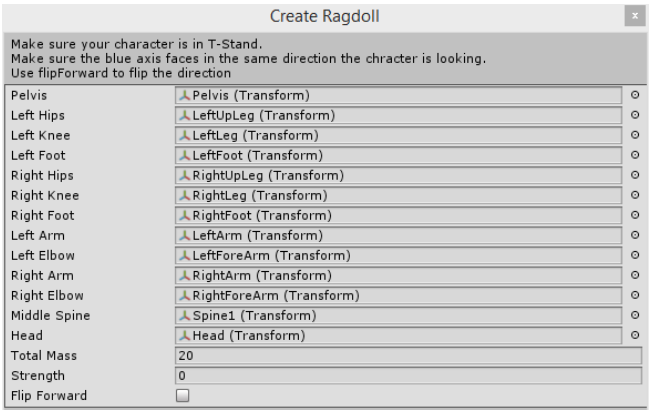
2-) Drag and drop your character to hierarchy window.

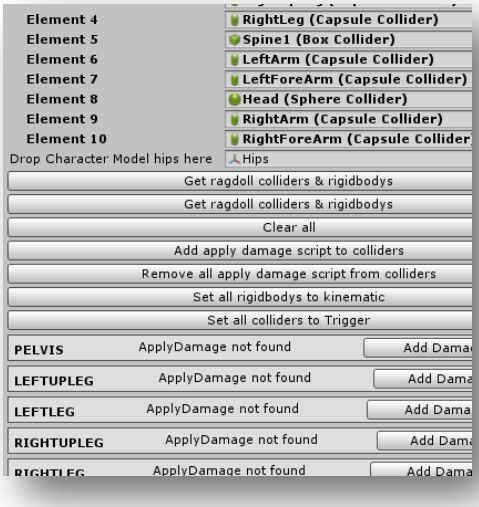
Make sure all of these components are added. I recommend copying these components from any of the available shooters and paste those components as new.

So that you won't have to fix everything. (like serialized public variables).

Tip -Don't forget to import Shooter Animator Controller to your animator.

3-) Now you will need to create a ragdoll for your new model. Open *GameObject/3D Object/Ragdoll* window and show need bones of your model to create a ragdoll. Hit create then move on to next step.





4-) Open Health component of your character now, and drop your model's hips bone to *'Drop Character Model Hips Here'* field. When you drop the hips it should show an option to find all colliders. Hit that button to access options.

Hit these buttons:

a-Add apply damage script to colliders.

b-Set all rigidbody's to kinematic.

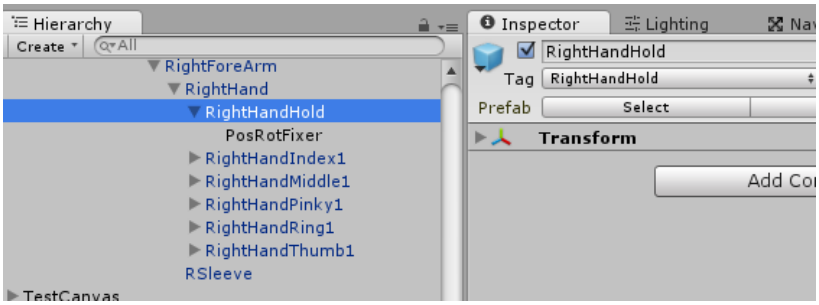
c-Set all colliders to trigger.

You can now modify damage take multipliers for specific bones if you like.

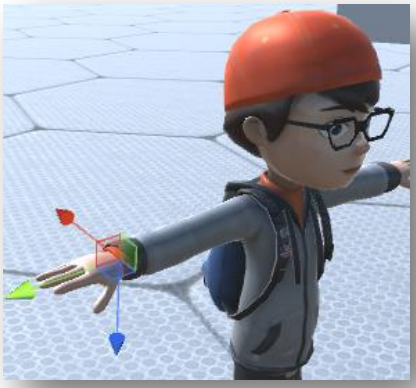
5-) Drag and drop a weapon prefab to scene for new shooter and drop this weapon to weapon field.

6a-) Find right hand bone of your new model and create these empty gameobjects as child of your model's right hand bone. (You can also copy them from another agent)

(RightHandHold and PosRotFixer)



Rename them like shown in this picture and set RightHandHold's tag to RightHandHold.



6b-) Reset RightHandHold transform's position and rotation.

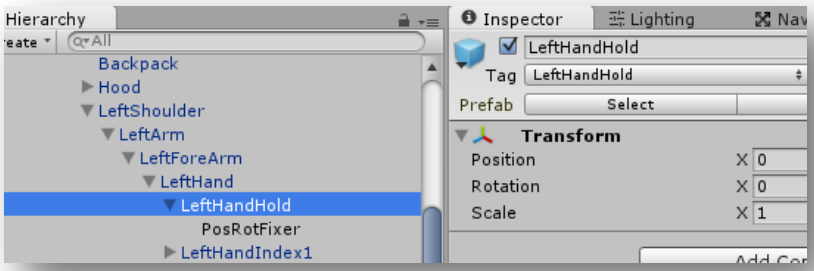
Make sure RightHandHold named gameobject you have just added has axis directions like shown in this picture.

Tip-This step is needed to unify all characters for weapon positioning.

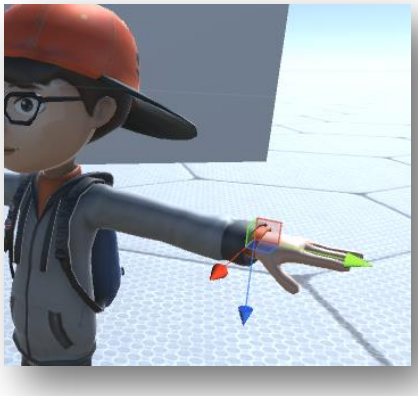
7a-) Find left hand bone of your new model and create these empty gameobjects as child of your model's left hand bone.

(You can also copy them from another agent)

(LeftHandHold and PosRotFixer)



Rename them like shown in this picture and set LeftHandHold's tag to LeftHandHold.

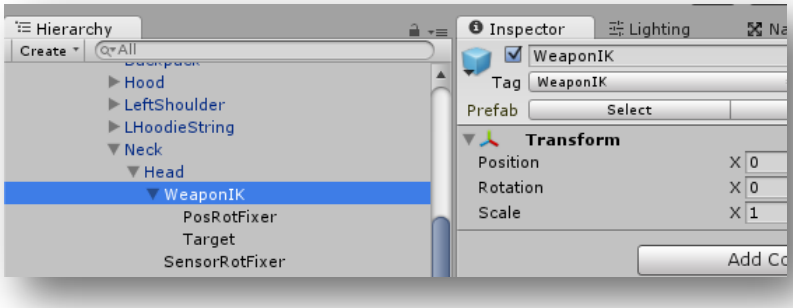


7b-) Reset LeftHandHold transform's position and rotation.

Make sure LeftHandHold named gameobject you have just added has axis directions like shown in this picture.

Tip-This step is needed to unify all characters for weapon positioning.

8a-) Find head bone of your new model and create these empty gameobjects as child of your model's head bone (You can copy paste all of them from another shooter '**Recommended**').



(WeaponIK, PosRotFixer, Target, SensorRotFixer)

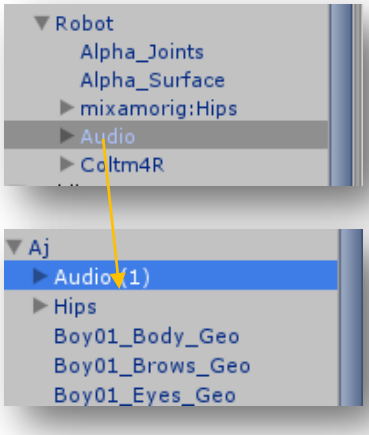
Tip-SensorRotFixer is needed for SensorVisual to align rotation properly.Make sure you rename the corresponding field in the sensorvisual tool if it is different.

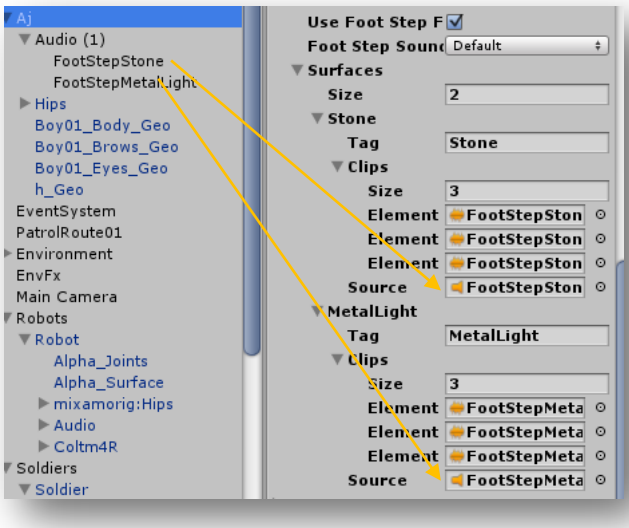
Tip-Target gameobject's local Z position modifies weapon aiming. I recommend copy pasting gameobjects from another shooter to prevent mistakes for this step.

8b-) Reset WeaponIK 's rotation and position. Set Its tag as WeaponIK. Rotate the WeaponIK to match axis directions like shown in picture.

9-) To setup foot step audio, duplicate gameobject named Audio (from one of the shooter prefabs) and set its parent as your new Model. Reset rotation and position of Audio gameobject.

Show audio sources as your new gameobjects like shown below.





-That is all you need to do, to correctly setup your character.

Tip-After these steps, create your sets and fill them with tools and drop those sets to ShooterBehaviour component.

Tip-Uniquely set your character's main gameobject layer. Then set all his bones to same layer. Like; Aj for main transform layer and AjBody for all child bones layer.

Tip -When using SensorVisual make sure to create an empty gameobject as a child of your model's head (shown in Step '8.a'), and write this transform's name to SensorVisual's corresponding public field. 'Default name is 'SensorRotFixer' , if you name it like this don't worry, it will be found. You can find the correct rotation for this fixer gameobject by using Quick edit tools option of ShooterBehaviour script.

Weapon System

There are three types of weapon animations in the animator, for pistols, for rifles and for big guns like rocket launchers. However, these animations do not limit the weapon styles that can be imported into project. For Ex. You can define a shotgun that uses pistol animations for weapon system.

Weapon system is capable of using almost all types of guns including shotguns, pistols, machine rifles and projectile firing. Currently, projectile firing is not shown with a prefab example, but it should work in theory.

Adding a new weapon

You can import new weapon models and make them actually fire. You can even add unique character animations for these weapons.

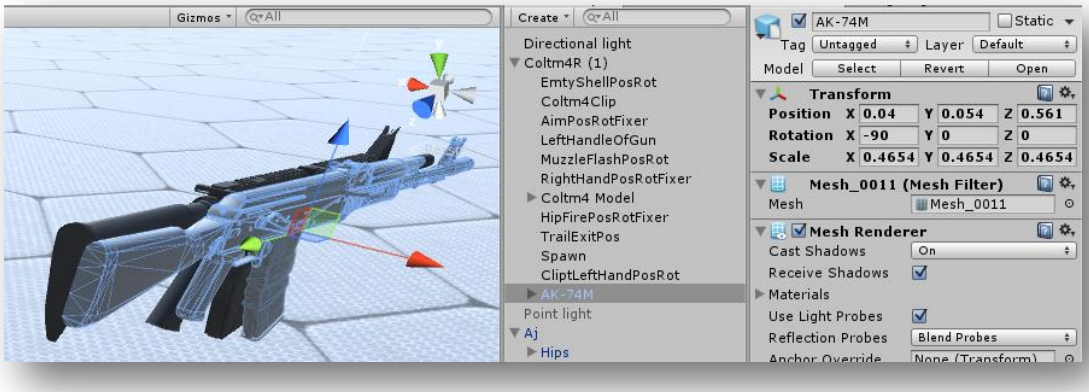
If you want to add new weapons to agents, I recommend using one of already made weapon prefabs as a reference so that process will be a lot faster *(like shown in step by step weapon importing).*

You will need to find the correct positions and rotations for all of these pos/rot holders to make your weapon look good in ai's hands.

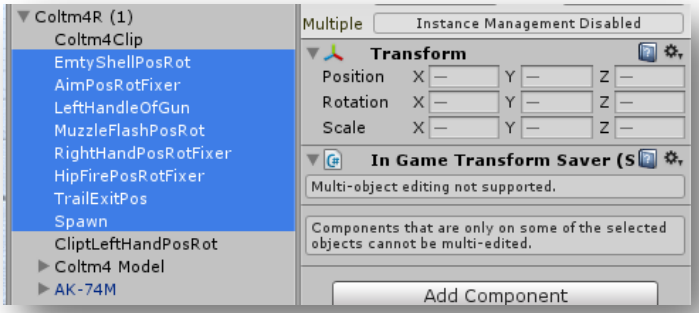
1-) Drag and drop an already made weapon prefab to the scene.

2-) Now drag and drop new weapon model to the hierarchy as a child of the weapon prefab you have in your hierarchy after completing step 1.

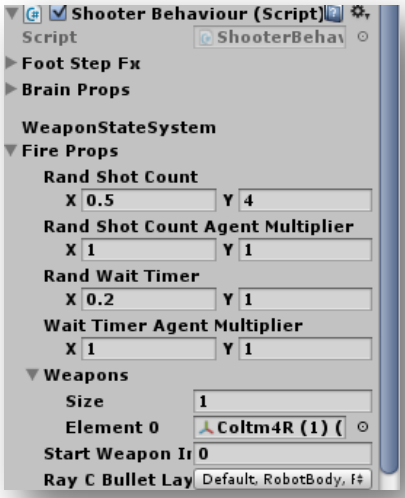
3-) Reset position and rotation of this new model's transform and set positions and rotations to match old model roughly by just looking at it.



Tip-There is an editor script (InGameTransformSaver) that lets you copy in game transform modifications after stopping game. You can add that script to shown gameobjects to load corrected and saved positions/rotations after stopping game.



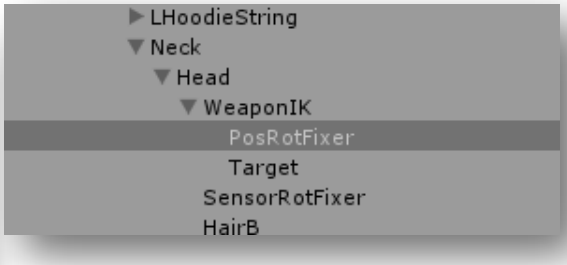
4-) You will need something to use as a target. I will use a robot as a target practice (needs to be an enemy) while I am correcting weapon positions and I will also remove unnecessary actions from my character's *actionset*. Then I will set target's health to a higher value to prevent him from dying and then I will set weapon's current capacity to a high value, lastly I will stop enemy robot's planning from *BrainProps' StopPlanning* option. (this steps are optional; I just want to make sure I will correct everything in one step).



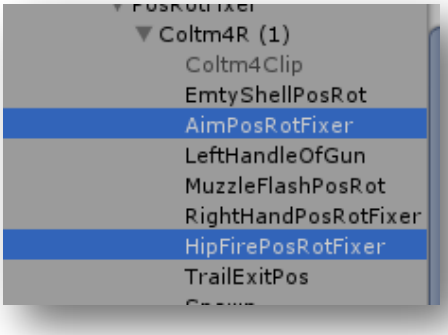
5-) Set this weapon as your character's weapon (drag and drop to *ShooterBehaviour's FireProps' Weapons' Element 0* field). Hit play and while your ai is shooting or staying idle you can follow these steps to correct each piece by pausing game.

-You can now disable old weapon model which is child of weapon main transform.

a-) To correct weapon right hand position to match the character animation (no need to be aiming to correct). Pause game and position/rotate transform named *PosRotFixer* which is child of *RightHandHold* like shown above. Then copy paste this transform component to transform named *RightHandPosRotFixer*(which is child of weapon). Hit save to temporary button in ***In game transform saver*** component of *RightHandPosRotFixer* (You will need to hit ***'load from temporary'*** after stopping play mode to load these saved values).



b-) This time make sure character is aiming/firing and don't pause the game. Similar to previous step, modify *PosRotFixer* of *WeaponIK* and copy paste transform component of *PosRotFixer* to *AimPosRotFixer* or *HipFirePosRotFixer's* transform component. Hit save to temporary button.



c-) Modify further attributes of weapon as you prefer, like trail exit position, left handle of gun, use in game transform saver as needed. Stop play mode and load all your temporary saved transform component values. After that whenever that weapon is used it will remember those values and everything will be just like you have set.

6-) You can also set a new clip for your weapon to character to use while reloading. Weapon model's clip is need to be extracted from weapon model to use as a usable-in-left-hand-clip while reloading. Look at one of available weapon clips to rig your new clip model with needed components. Set it as your weapon's child. Position it or rotate it as needed. Create a clip prefab from this in the project folder. Select new weapon's main transform in hierarchy and drag and drop to clip fields of *GunAtt* script (current clip and clip prefab). After that you can hit play mode and while character is reloading (new clip model should be in his left hand) pause game and fix this clip using *PosRotFixer* of *LeftHandHold*. Copy *PosRotFixer's* transform component and paste these values to *CliptLeftHandPosRot* (child of weapon).

References

(This project would not be possible without the work of Jeff Orkin.)

http://alumni.media.mit.edu/~jorkin/aiide05OrkinJ.pdf