EleganTps User Guide

October 2015

Version 1.04

Empty Page

Document Revisions

Date	Version Number	Document Changes
28.10.2015	1.04	

Table of Contents

1 Introduction

1.1 Scope and Purpose

This project's purpose is covering main aspects of a third person game.

1.2 Workflow

This project uses unity 5's state machine behaviour system mostly. Almost every single animation state has its own behaviour. This behaviour has methods to be overritten. By overwriting these, we decide what to do when entering, during, exiting a single animation(or substate machine).
Some of these functions are shown below.

```
30 references
public override void Init(Animator anim)...

28 references
public override void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)...

6 references
public override void OnStateIK(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)...
```

Figure 1.1

Some of these functions are shown in figure 1.1, Init function is created by inheritance to get some public variables and object references on scene. For other methods u can refer to unity documentation.

Another important workflow part would be classes that hold data / data&functions. For example, there is a class that holds every parameter of a weapon. These parameters are modifiable in play mode.

Animator states are lined up well and every animation and blend trees are named correctly to speed up the understanding process. Almost every public inspector parameters are explained in classes.

1.3 Prefabs overview

There is a folder named prefabs in the project. Every prefab are modified for general use. But can be changed to work very different according to your game.

Some of these prefabs are:
Bullet hit sounds: Gameobjects with audiosource components with played on awake tick.
Bullet holes: Gameobject to be appear on walls if a bullet hits.
Camera rig: Everything about camera and some other features that needs to be in touch with camera like crosshair-target system.
Player: Main character with all parameters modified for general third person game
Environment Effects: This gameobject has a script to hold any bullet hit effect/sound/bullet holes according to a tag.
Char_Ethan Curve checker: This prefab can be used when modifying an animation to work with foot placement, or you can change this to work with your own character.(Part 4.)

Weapons: Every weapon with different paremeters.
Mobile Controls: A good example of how to use mobile control system with this project.
Canvas 2d: Crosshair related objects that has 2 dimensional canvas.
Canvas 3d: A 3D game menu hud example like the one used in modern games.

Some of these prefabs 'must be' in the scene to start and play the game without any error. These are : CameraRig, Canvas2d, Canvas3d, Environment Effects, Player.

2 Player

2.1 Player attributes script

2.1.1 General Information

This script holds main character's public parameters and other functions which are related to character. *State machine behaviours gets or sets data of main character using this script.*

2.1.2 Public variables

Cover vars: This class's parameters used when covering(substate machine named Cover)
- Crouch up peak limit: Up limit to be able to up peek
- Wall offset: Wall distance to align the player when entering cover
- Moving normal check offset: Offset to check cover wall if it is ended or not
- Stand accelerate: Stand cover walk acceleration force
- Crouch accelerate: Crouch cover walk acceleration force

Control mode: Third person games may use one of these modes. With free mode player can rotate-move look at camera... An example for this third person game : Mad max. If you select static mode player always rotates when camera does. Player can't look at camera or rotate if camera does not. Game named splinter cell can be given as an example to this type of control.
Use reach to grab: Animations are played when grabbing a weapon
Use sudden stop animation: Animations to be played when player stops suddenly
Guns: This list holds the weapons character have.
Current magz: This list holds the current ammo count for any weapon type(every index defines bullet type)
Max magz: Max ammo count that player can have
Know gun parts: All the gun parts to be attched on weapons, that player knows.
Walk speed locomotion: A blent tree parameter to be set for walk animations.
Debug Fire 2 Press: Can be used for debugging aiming and everything related to it.
Debug Weapon Positions: Can be used to debud weapon positions when aiming or firing without aiming.
Gun Collider check mask: A layer mask to decide if there is a wall in front of a player and if he can or can't aim and fire.
Vector angle w targ epsilon: Min angle to decide if character turned to target enough that he can fire.

Min Vector Angle targ to raise weapon/Look IK/ Hand IK: Angle to decide if player must start to lookIk or not.

For other variable definitions of Player Atts script please go to editor.

2.2 Character Controls

2.2.1 General Information

SetupAndUserInput script is designed to define every aspects of controls. By seperating control script, converting these controls to other platforms will be easier.

2.2.2 Supported Control platforms

Currently Pc/mac and multitouch screen mobile device "controls" are supported, but It can also be converted to playstation or xbox easily.

This control script covers almost every type of control logic that can be used in third person game. From button down to button press for both touchscreen and keyboard. For touch screen swiping examples are also given. Some automating examples are also available in this control script for mobile devices.

2.2.3 Control Schemes

Keyboard/Mouse

```
W/A/S/D: Movement
Shift : Sprint
Tab : Open-Close Menu
E: pick up weapon
Space: Jump-Climb
F: Draw last weapon - holster weapon
G: Drop weapon
Q: Cover
Left Ctrl: Crouch - Slide
Mouse 0(Right Click): Fire
Mouse 1(Left Click): Aim
R: Reload weapon
1: Select first weapon
2: Select second weapon
3: Select third weapon
V: Enter-Exit weapon modification
Right-Left-Up-Down Arrow: Change parts
Left Alt: Walk-Run toggle
T: Weapon flashlight On-Off
Mouse 2(Mid Click): Secondary Fire/choose weapon from menu
Mouse Wheel: Choose weapon from menu
P: Restart
Esc: Exit game
```

Mobile:

2.3 Change Main Character

Character must have a skeleton that is comptible with humanoid animation type. You can check if your character is compatible by clicking configure button in inspector avatar definition. If all the bones are green it means that it is compatible to use with humanoid animations.

Follow these steps to change your character:
1. Import your character and set its animation type to humanoid using inspector (figure 2.1)
2. Drag and drop your new character model inside prefab named Player in hierarchy(figure 2.2)
3. Reset your new character model's position and rotation in Transform component.(figure 2.3)

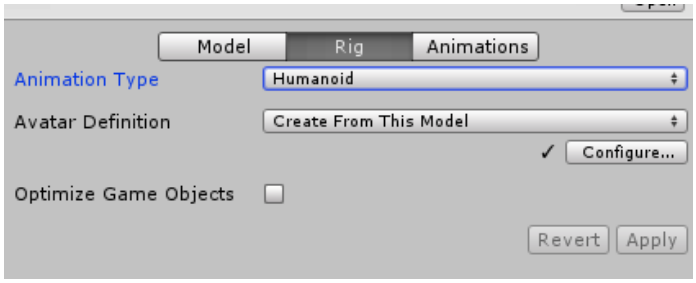


Figure 2.3.1

Figure 2.3.2

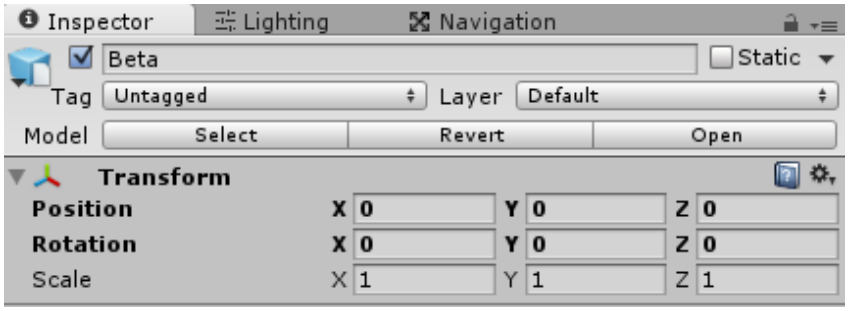


Figure 2.3.3

4. Find object named WeaponIK in char_ethan ‘s hierarchy
5. Duplicate it.

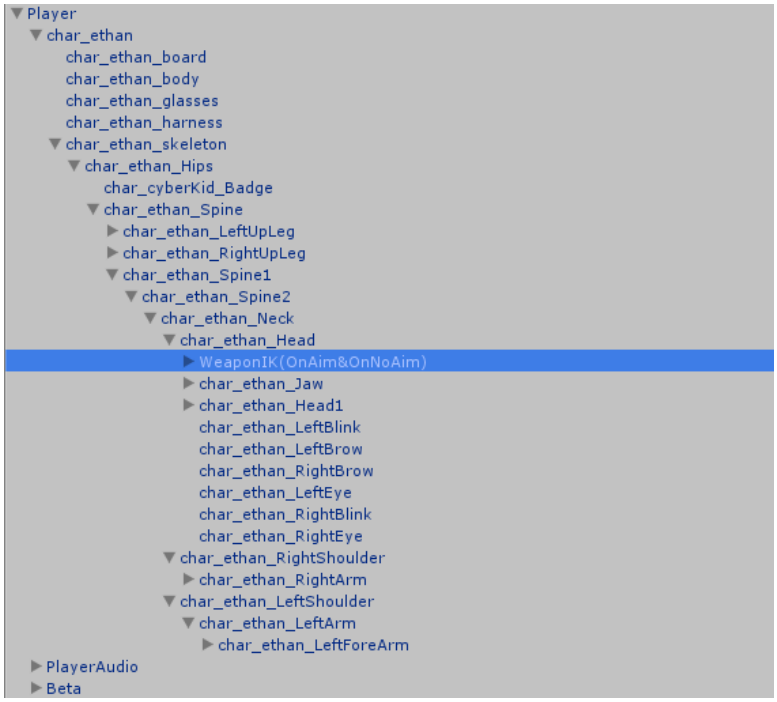


Figure 2.3.4

6. Find your new humanoid model's head and drag and drop your new object named WeaponIK you have just created by duplication

Figure 2.3.5

Figure 2.3.6

7. Now you need to follow the same steps for RightHandHold(RightHandPosRot) and LeftHandHold. Duplicate and drop them as child of new model's hands.

8. Disable char_ethan (old model)
9. Drag and drop your new model's avatar from project panel to Animator component of player.

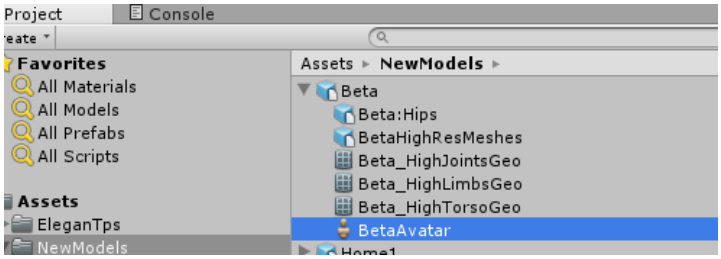
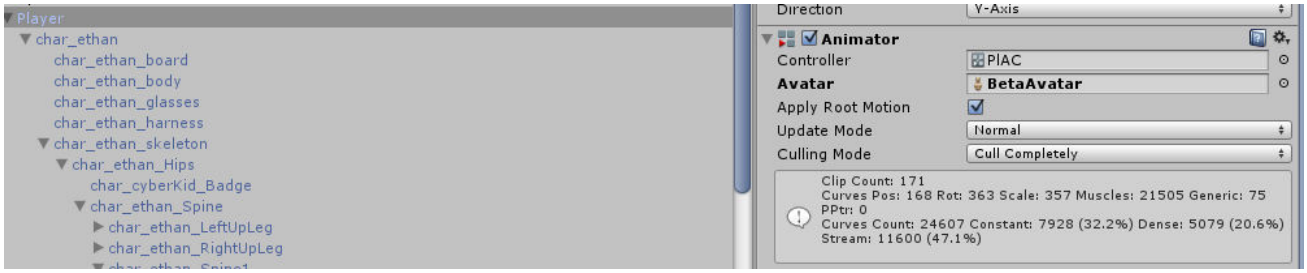


Figure 2.3.7



That's all for changing character. To fix weapon positions for your new character refer to Modify Make your own weapon subject(4.3).

2.4 Player Audio

I have modified and integrated the footstep sound system published by unity. It is easy to define and change foot step sounds for any collider that character steps on top of.

1. To use foot step sound system, first change floor model's (or collider if they are seperated) tag to anything you like. I've already created a tag named Stone and MetalLight. In this case I've set my object's(floor collider object) tag to stone.

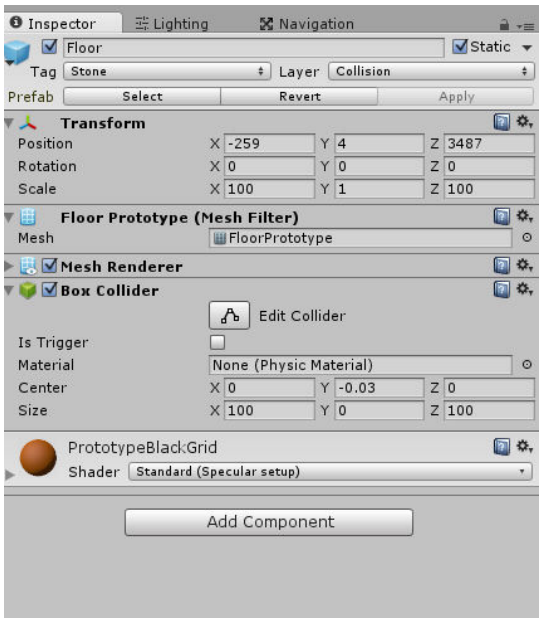


Figure 2.4.1

2. Create an empty gameobject as a child of PlayerAudio gameobject, and name it FootStep<yourtag> like figure 2.4.2 below. (or just duplicate one of available items and rename)

Figure 2.4.2

3. Add an audiosource component to this object if it does not have.

4. Click player prefab in hierarchy and find footstepsounds in player attributes script.

5. Increase surfaces size and enter your new tag to be played if player is on top of any ground with this tag.

6. First drag and drop your Gameobject(in my case FootStepStone) to Source of surface. (figure 2.4.3)

7. Set clips size and just drag and drop your audio files to these elements (figure 2.4.3)

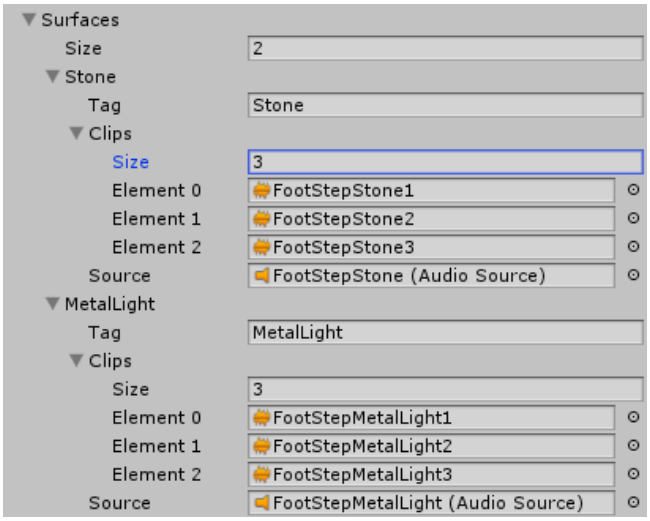


Figure 2.4.3

3 Foot Placing and Slope

3.1 General Information

EleganTps has its own foot placing system. There is a script named FootAlign, player prefab must have this script attached in order foot placing system to work.
This script has two main jobs:
1. Placing foot and leg according to ground changes.
2. Changing character's capsule collider for ground changes(Slope system).

3.2 WorkFlow

3.2.1 Slope system workflow

Slope system changes player's collider according to player's forward. Raycast is used in front of player collider to learn ground changes before player moves there.
Sloping is good if you want character to climbing steps, or sharp collider low height objects. Along with this, sloping is good with foot placing to fix all the player model's hips aligning.

To make slope system work you don't have to do anything special, just enable slope enabled variable which is already enabled by default. But you can change these variables according to your character.

Slope collider smooth: Capsule collider change speed according to ground changes
Slope distance: Slope distance in front of character
Slope down ray dist: You can change this to modify ray distance to detect ground
Slope distance: Change the max height player can slope(you may need to change ray distances)
Slope up ray dist: Slope ray starting position height upwards

3.2.2 Foot placing system workflow

Foot placing has two main aspects covered: Placing the foot and legs to ground properly. Unity's Inverse Kinematics is used to tell our character where its foot should be.
Two main things is used to find foot position:
1. Raycasting
2. Animation curves
Animation curves are used to find foot height according to an animation.
Raycasting is used to find ground and align feet according to curve we get from animation curves.
Using these with each other makes foot placing system to work with many different surfaces.

3.3 How to use foot placing

In order to make foot placing work with an animation, you must create curves for that animation. Some animations like the ones used in Locomotion are already rigged with curves. You can look at them as an example if you are stuck.
Step by step:

1. Make sure foot align script is attached to player prefab.
2. Select your animation from project panel and open curves.
3. Create two curves for foot to be used as a reference.

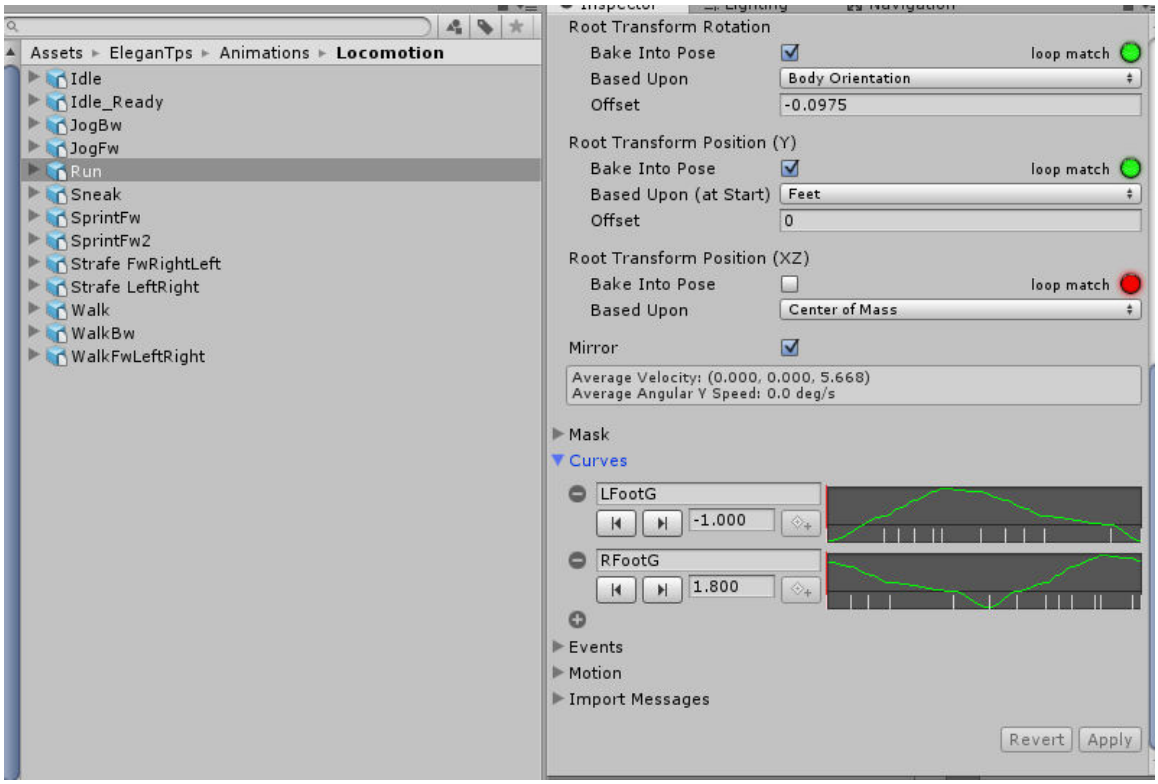


Figure 3.3.1

4. Change curves names like above.
5. You can change the preview model to char_ethan Curve Checker to be able to find foot height during an animation correctly. (by clicking avatar shaped button on left bottom of preview panel). You can always modify this curve checker for your own model.

6. As you can see in figure 3.3.2, left foot is in front of second curve checker pattern texture. You can assume this as 0.5 like the one used in this project. And set the curve value like in this figure. And if foot is on third square pattern it should be 1. (and 1.5, 2, 2.5...)

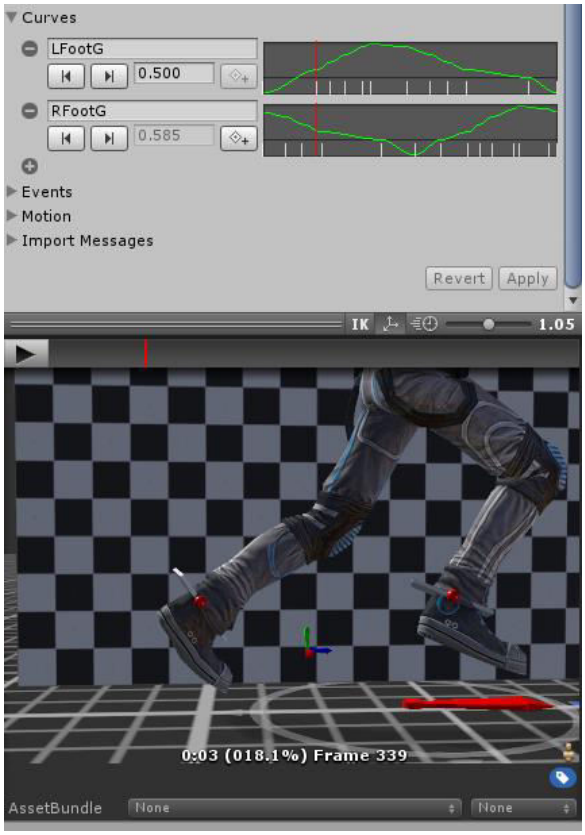


Figure 3.3.2

7. The important thing is that you should define the moment foot leaves the ground as zero and meets the ground as zero.
8. Once you finished defining both feet curves hit apply and system will do the rest.
9. Your new model's foot rotation may be different. You can use Left foot rotation fix, and Right foot rotation fix inspector variables of foot align script to fix this.

You can also modify foot height curves by changing curve effector value of foot align script. Animation curves just needs to be proportional and 0 or less if foot is on ground.

4 Weapons

4.1 General Information

Weapon system is a big part of this project. Currently 4 type of animations are available for different types of weapons. But system is developed in a flexible way so that it can define all types of weapons easily.

4.2 Supported weapon types

Many weapon prefabs are included in this project but many other different weapons can also be defined.
Supported types:
Shotguns, Automatic assault rifles, all types of machine weapons, pistols, single shot rocket launchers
Not supported currently:
Automatic rocket launchers. This logic is not defined yet due to lack of model for this weapon.

4.3 Make your own weapon

Defining a new weapon can be really detailed according to your needs. There will be an example to show you how to define your unique weapon. In this example we are going to define a automatic machine by using prefab named Ak.
Step by step:

1. Find a weapon model.
2. Drag and drop your weapon model as a child of ak prefab into the hierarchy panel.
3. Fix position rotation and scale like figure 4.3.1. Basically match size and positions.

Figure 4.3.1

4. If you have your clip object seperated like figure 4.3.2 you can create a clip prefab for this weapon to be used when reloading.

Figure 4.3.2

5. Create a new slot in player Atts script for our new weapon to test. Rename Ak to anything you like in hierarchy and drop it to our slot in player Atts.

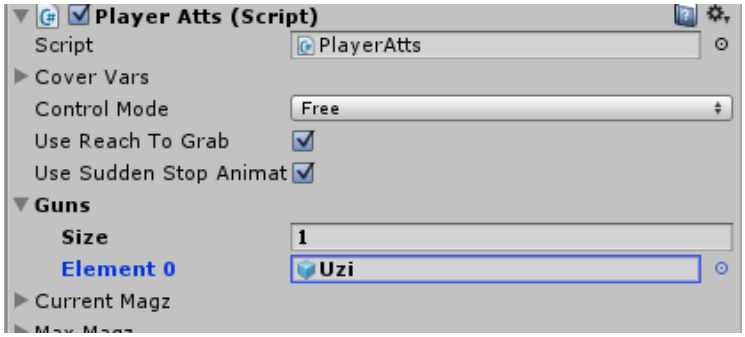


Figure 4.3.3

6. Delete clip of weapon and disable old model. We will define a new clip prefab for our weapon.

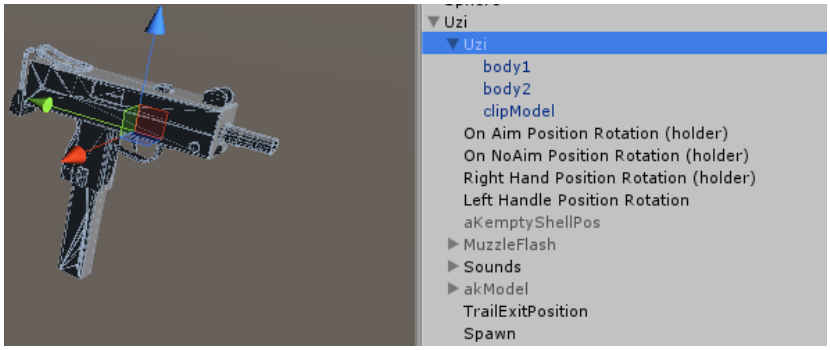


Figure 4.3.4

7. I suggest creating a new folder for our new weapon. Now drag and drop clip model of your new weapon to project panel to create a new prefab. In my case it is clipModel in figure 4.3.4
8. We need to add two components to our clip. Rigidbody and box collider. Set rigidbodys is Kinematic variable to true, adjust your box collider if you need to and disable box collider. Everything should look like this... (we will use this clip later)

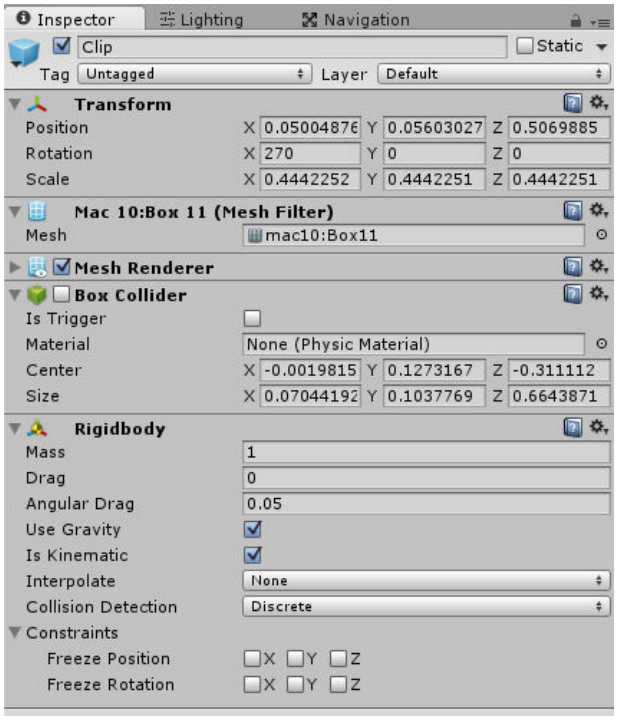


Figure 4.3.5

9. Now, fix sphere and box collider to match our new weapon model.

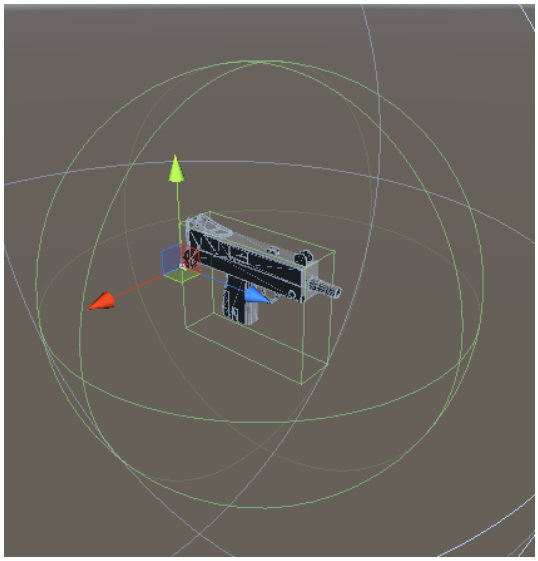


Figure 4.3.6

10. Now hit play. If you have changed your character like I did and did not fix old weapon that we are using to create one, you will see that he does not know how to hold this weapon. Now we will show him how to hold it correctly.

Figure 4.3.7

11. Still in play mode, find your character's gameobject named RightHandHold(RightHandPosRot). And start to play with its position and rotation in scene panel and make him hold the weapon correctly like I did. Dont mind left hand position for now.

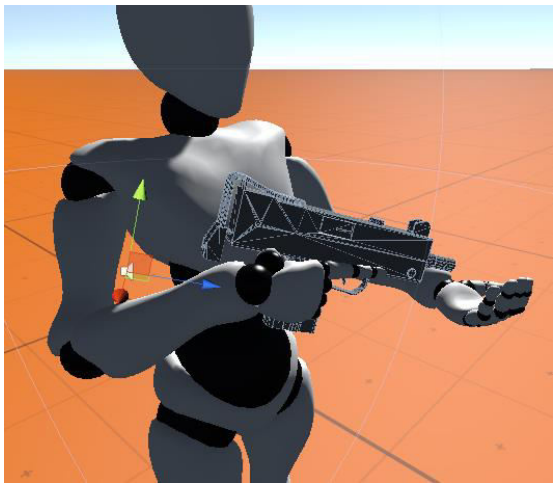


Figure 4.3.8

12. Now copy right click to transform component of RightHandHold(RightHandPosRot). And hit copy component.

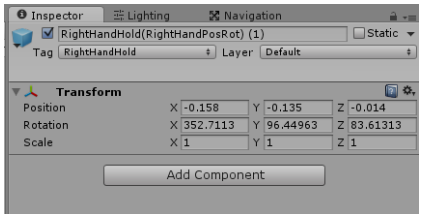


Figure 4.3.9

13. Hit play button again to exit from play mode. And paste your component to object which is child of weapon, named Right Hand Position Rotation (holder)'s transform component (right click to transform and past component values). When you hit play again you will see that character remembers this holding position.

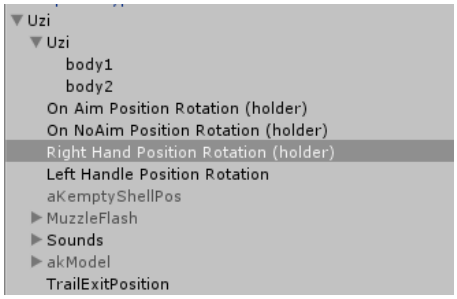


Figure 4.3.10

14. By following the similar way fix your left hand holding position however you want. But this time modify directly object named Left Handle Position Rotation (holder) , which is child of your new weapon. I want him to hold the weapon like in figure 4.3.11.

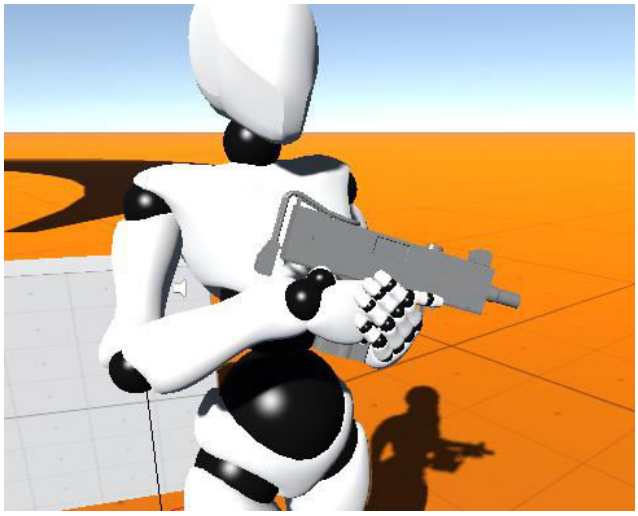


Figure 4.3.11

15. Now we will tell our character how to aim with his new weapon. First set both of these variables to true from Player Atts script :

Figure 4.3.12

16. Hit play and find object named WeaponIK from hierarchy. Again, fix position and rotation from scene panel of this object.

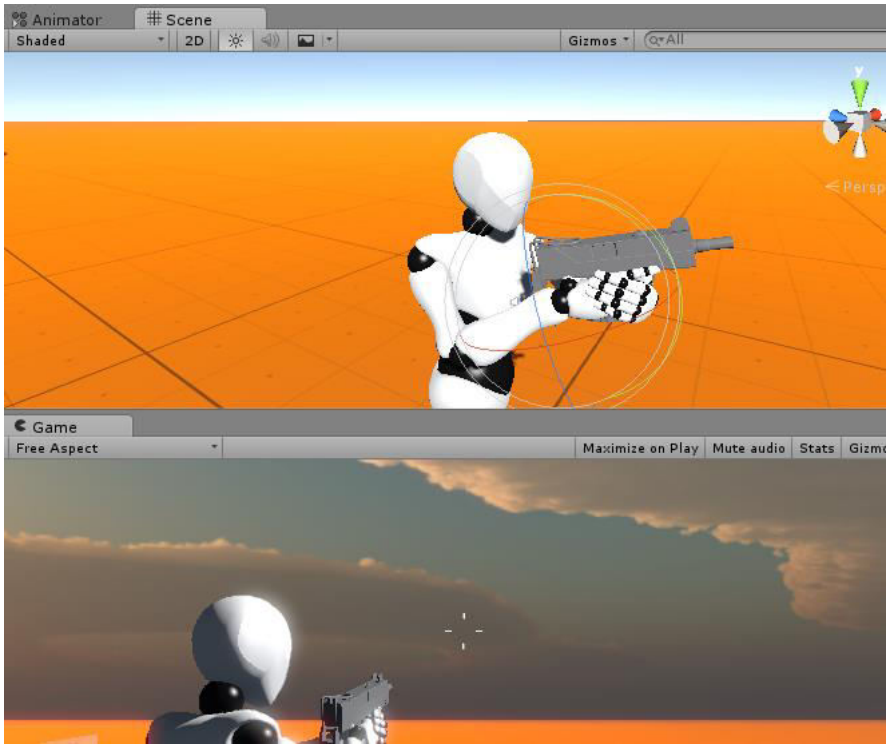


Figure 4.3.13

17. Now copy WeaponIK's transform component and exit from play mode.
18. Like we did before paste component values to object named On Aim Position Rotation (holder) (child of weapon)' s transfrom component.
19. To speed up fixing position for shooting without aiming position paste the same values to On NoAim Position Rotation (holder)'s transform component.

20. Hit play again and this time modify WeaponIK gameobject however you want player to hold weapon when shooting without aiming.

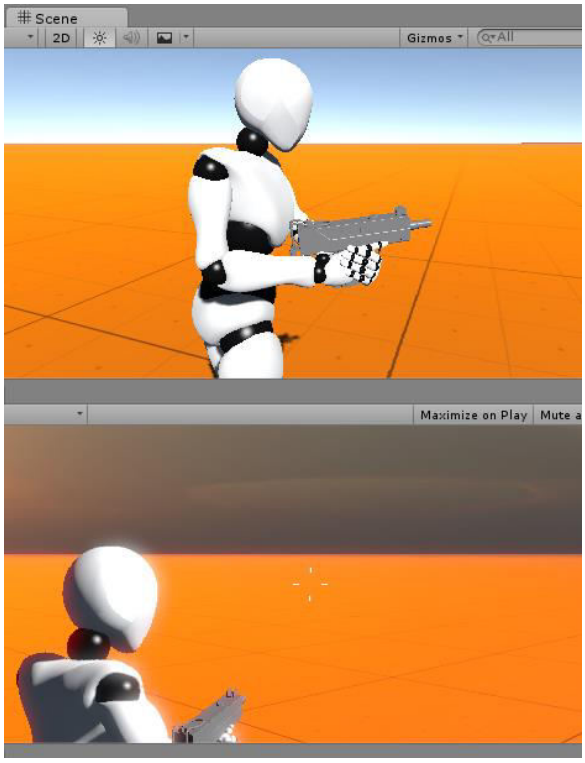


Figure 4.3.14

21. Again copy WeaponIK gameobject transform component, stop play mode and paste values to On NoAim Position Rotation (holder) gameobject's transform component.
22. Now drag and drop our clip that we've created before as a child of new weapon. Then fix position, rotation for this clip.
23. Click your new weapon in hierarchy and find variables named Cur Clip Object and Cur Clip Prefab. Drop your clip object from hierarchy to cur clip object. And drop your clip prefab from project panel to cur clip prefab variable.
24. You can modify the weapon as you like from now on. But let's continue to finish main aspects of our weapon. Set gun style to 1. And go to animator panel layer named arms, and select pistol reload animation. From inspector set debug to true. Now if you hit play mode you can tell your character how to hold clip by changing L Hand Clip Pos/Rot Fix variables. (you must be on reload animation)

4.4 Taking it one step further

Weapon system in eleganTps allows you to define your own animations for a particular weapon. If you open animator controller and go to layer named Arms and open substate machine named Weapon you can see there are three lines of animation set with their reload animations(one of them figure 4.4.1). If you think of rigging a particular weapon with its own animations you can also do it easily.

Figure 4.4.1

If you add 4 animations like you see in figure 4.41 + 1 reload animation for your weapon and create transitions like these along with behaviour scripts, only thing you need to change will be weapon style variable in gunAtt script and your new weapon will use these animations.

4.5 Melee weapons

EleganTps has also basic melee weapon system to show that project is capable of defining it. Melee weapons are still in development and may be improved in next updates by suggestions coming from you.

4.6 Weapon Modification system

Project has a unique weapon modification system, by using this system weapons can be modified by developer in scene mode or in play mode by player. Many weapon parts logic is added to project. There is also a modifiable weapon prefab available in project. Note that in game(play mode)modifiable weapons are not tested yet with more than one modifiable weapon and there may be tiny bugs, and modifying rocket launcher type of weapons not available.

5 Cover System

5.1 General Information

There is also cover system available in eleganTps. This system is good enough to define many different cover positions for player.

5.2 Cover system workflow

Cover system had three different main aspects.

1. Finding cover position and entering cover mode.
2. Moving in cover mode.
3. Aiming in cover mode.

Finding cover position is defined by an empty gameobject that has two components: Bezier spline and a box collider. And of course you will need a wall.

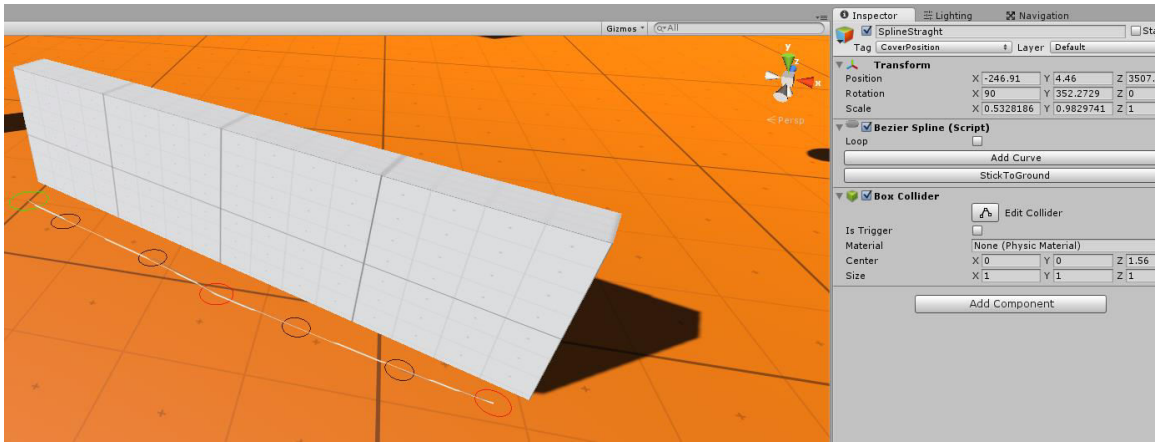
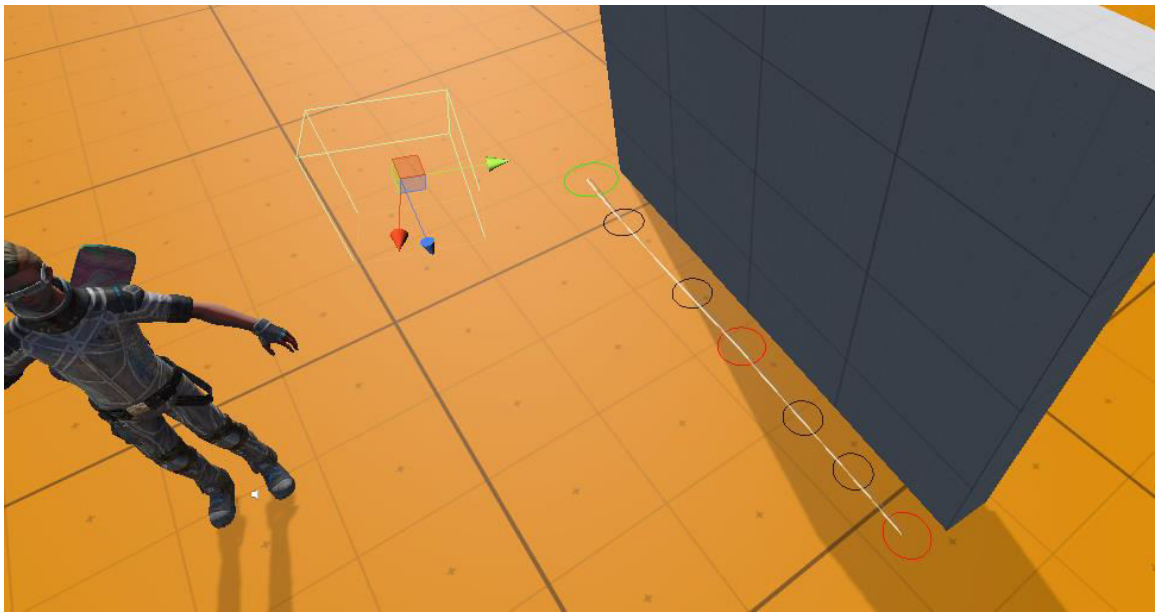


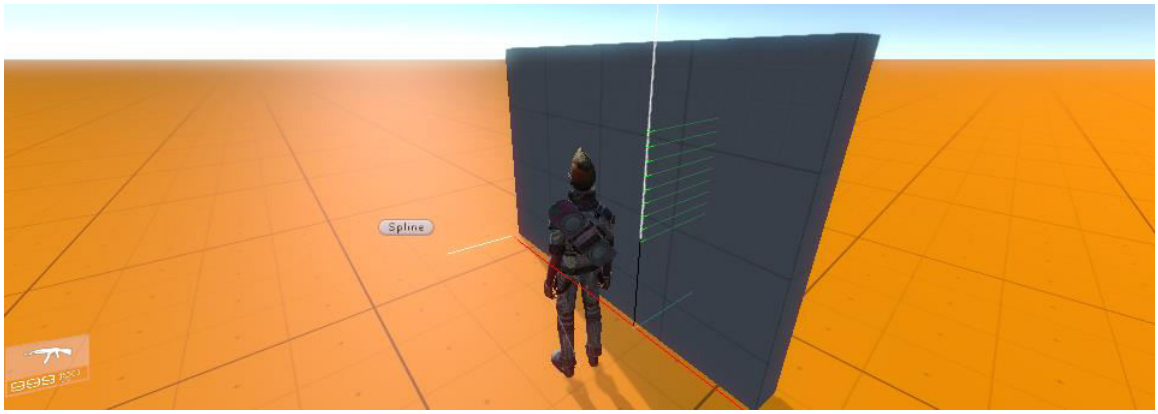
Figure 5.2.1



In the figure above a complete cover position is defined. That bezier spline is used to determine the close cover position in this particular cover position. And box collider is used by sphere cast to find all cover positions around. So make sure you bury that box collider below ground or move it somewhere that any other gameobject won't collide with it. This empty gameobject needs to have these two components and it's tag must be CoverPosition like you see above. Don't forget bezier spline's rotation is important also. You can take a look at example cover positions available.

By using this spline and wall character will decide which position he should cover. He must be close enough to somewhere in this spline to enter cover position.

After entering cover position character will use raycasts to check cover height, cover edge, by looking at cover height and cover edge he will decide if he should crouch or stand, if he can edge aim or not.



These kind of raycasts are done to determine cover height, position etc. If player dont find any cover wall he will get out of cover mode.
Different cover system is supported as you can see. You can define cover that player automatically crouches/stands as he walks or you can define covers that can be destroyed by enemy, or circular cover positions can also be defined.

Don’t forget that colliders must be shaped smooth in order cover system to work correctly. Be careful when working with mesh colliders.