Generic IK

Simple and easy to use IK solution for both Humanoid and Generic rigs

Generics Studio 2016

Index:

- 1.0 **Introduction**.:
  - 1.1 Getting Started.
  - 1.2 Demos.

- 2.0 **Components and utilities**.:
  - 2.2 Kinematic Solver.
  - 2.3 Analytical Solver.
  - 2.4 CCD Solver.
  - 2.5 FABRIK Solver.
  - 2.6 IK Chain Builder.

- 3.0 **Finally,**.

1.0 Introduction.

Hi, my name is Mohamed Hassan, the founder of Generics Studio, a studio dedicated to create fun games and useful tools for developers.

Thanks for your interest of our package and I hope you find it as you expected it to be.

Generics IK offers an easy and simple to use IK solution for both Humanoid and non-humanoid models.

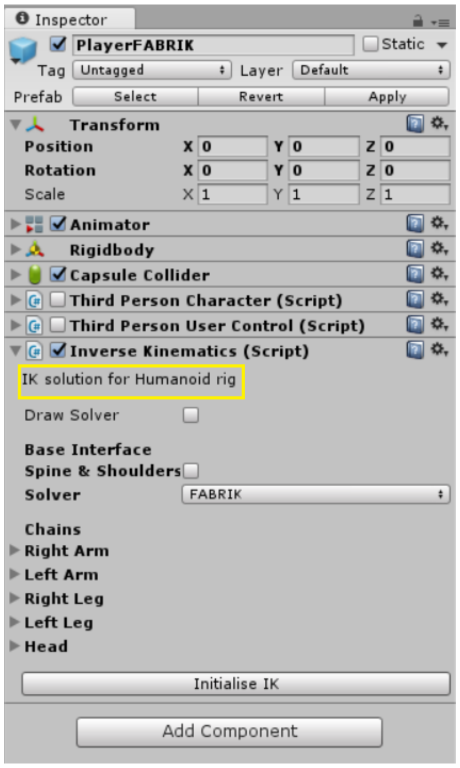
This can be used to create fun-interactive game characters, and that could be achieved as what the designer of the game sees, or could be even to correct some of animation that might seem to have a little offset from an object or a target for example.

With that being said, let's take a look on how to actually use the tools!

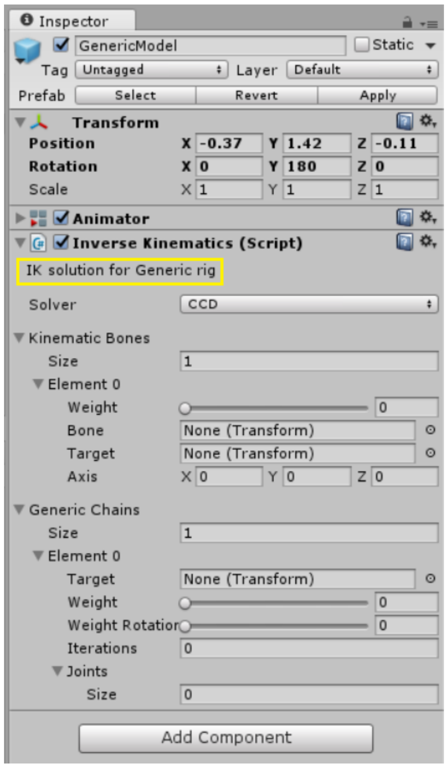
1.1 Getting Started.

There are 2 ways of using the IK system. It’s either visually from the inspector or through code, and we will take a look at them both.

- **The inspector way**.
  All you have to do is simply drag and drop the InverseKinematics.cs file from the project onto the object, here you get one of two possibilities, either your model’s rig is Generic or Humanoid and the IK script will automatically detect that.
  If the rig is generic then you will get 3 parameters. Solver, Kinematic Bones and Generic chains and you can set as many as you want.
  but in Humanoid, everything is automatically set up for you after you click Initialise IK.



```
Inspector
PlayerFABRIK                         Static
Tag Untagged          Layer Default
Prefab  Select   Revert   Apply
Transform
Position   X 0   Y 0   Z 0
Rotation   X 0   Y 0   Z 0
Scale      X 1   Y 1   Z 1
Animator
Rigidbody
Capsule Collider
Third Person Character (Script)
Third Person User Control (Script)
Inverse Kinematics (Script)
IK solution for Humanoid rig
Draw Solver
Base Interface
Spine & Shoulders
Solver            FABRIK
Chains
Right Arm
Left Arm
Right Leg
Left Leg
Head
Initialise IK
Add Component
```



```
Inspector
GenericModel                         Static
Tag Untagged          Layer Default
Prefab  Select   Revert   Apply
Transform
Position   X -0.37   Y 1.42   Z -0.11
Rotation   X 0       Y 180    Z 0
Scale      X 1       Y 1      Z 1
Animator
Inverse Kinematics (Script)
IK solution for Generic rig
Solver            CCD
Kinematic Bones
  Size            1
  Element 0
    Weight                    0
    Bone          None (Transform)
    Target        None (Transform)
    Axis          X 0   Y 0   Z 0
Generic Chains
  Size            1
  Element 0
    Target        None (Transform)
    Weight                    0
    Weight Rotation           0
    Iterations    0
    Joints
      Size        0
Add Component
```

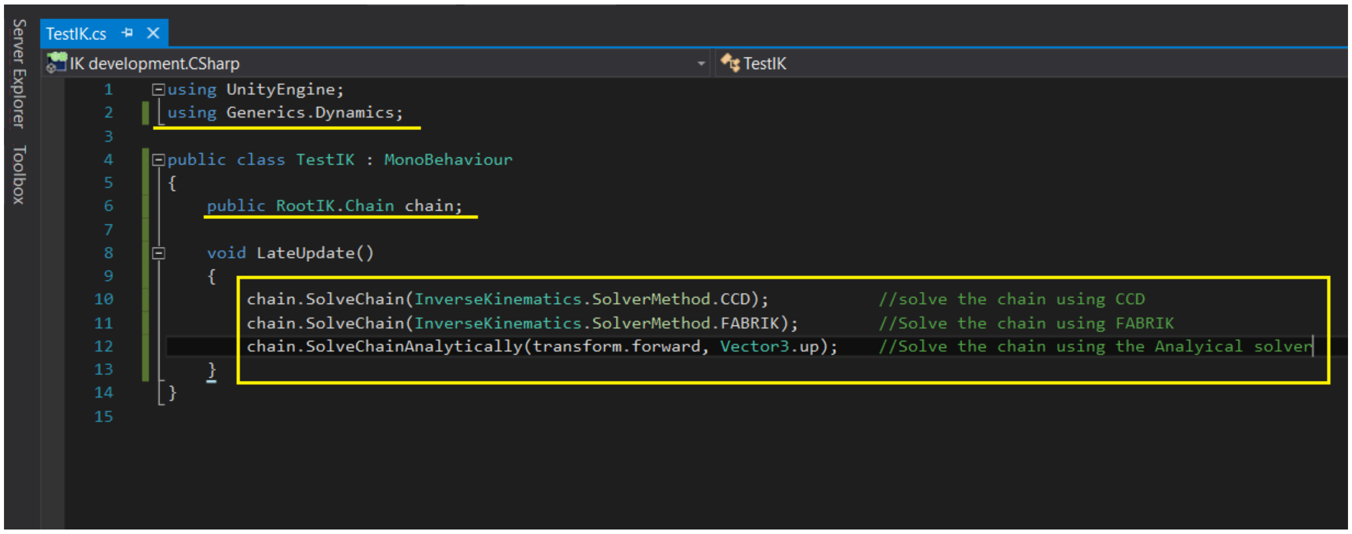
Now set the target Transform in each chain, the number or iterations, weight and you are good to go !

NOTE ! : all chains that you want to solve for MUST have an iteration higher than zero (it can be any positive number) to work otherwise the solver will ignore the chain.

One thing to note is that if your rig is generic, then you have to set the joints manually, while in humanoid its automatically set and you just need to worry about the targets and weights for each chain.

- **Code way**:
  - Create a new script
  - Include the namespace Generics.Dynamics;
  - make a public field as the picture;
  - in LateUpdate() call the SolveChain(SolverMethod) on that chain
  - You're done ! but of course don't forget to setup things in the inspector.

More on the API and code method in the Components and Utilities chapter.



```
using UnityEngine;
using Generics.Dynamics;

public class TestIK : MonoBehaviour
{
    public RootIK.Chain chain;

    void LateUpdate()
    {
        chain.SolveChain(InverseKinematics.SolverMethod.CCD);           //solve the chain using CCD
        chain.SolveChain(InverseKinematics.SolverMethod.FABRIK);        //Solve the chain using FABRIK
        chain.SolveChainAnalytically(transform.forward, Vector3.up);    //Solve the chain using the Analyical solver
    }
}
```

1.2 Demos.

Demo scenes are included in the package so feel free to play with to help you better understand how the flow goes.

- **Feet Placement demo**:

  In this demo, we used the Analytical solver to better reposition the root (hips) and the both feet to suit the type of ground we are standing on, which is everything set on the layer “Water”.

- **Interaction IK demo**:

  In this demo, we use raycasts to find a hit point and the hit normal on surfaces set on the layer “Transparent FX”, then feed these into to the right arm chain and solve. The result is the player touches walls and other object when he gets near them.

- **Inverse Kinematics demo**:

  In this scene, we showcase all the different/simple set ups that could be required in a typical game production, of course there’s more but these are the essentials.

- **Prototype demo**:
  Here we test the IK on non-animated Generic rigs.

2.0 Components and utilities.

RootIK.cs:

All you need to know about that class is that it contains the building blocks for any IK chain. It contains the Chain class which includes parameters like:

- **Target** = the target transform
- **Weight** = the IK weight for the whole Chain
- **WeightRotation** = the weight of the last bone's rotation (the end effector)
- **Iterations** = the number of times the solver will solve each frame.
- **Joints** = the joints of that chain (the chain's links)

Now the chain class has some helpful methods and they are:

- **SetIKPosition(Vector3)** = set the IK goal
- **SetIKRotation(Quaternion)** = set the end effector's rotation
- **GetIKPosition()** = returns the current IK goal's coordinates
- **GetIKRotation()** = returns the current IK goal's rotation
- **GetEndEffector()** = returns the transform of the end effector
- **SetIKPositionWeight**(**float**) = set's the weight of the chain
- **SetIKRotationWeight**(**float**) set's the WeightRotation
- **SolveChain**(**SolveMethod**) = solve the chain using either CCD or FABRIK.
- **SolveChainAnalytically(Vector3 direction, Vector3 rotAxis)**

  = solve the chain analytically, direction is the direction of the character (usually we supply transform.forward unless you have other plans), and rotAxis is the axis of rotation for the 2nd bone (the hinge) in the chain.

2.2 Kinematic Solver

The Kinematic Solver solves Kinematic Bones, Kinematic Bones are essentially just single bones which should look at something (i.e. Head looking at a target, a finger on a trigger maybe and so on).

It contains parameters like:

- **Weight** = the weight of the bone
- **Bone** = the bone transform
- **Target** = the target transform
- **Axis** = the direction of look at in local space

2.3 Analytical Solver

The Analytical Solver solves the IK chain with essentially 1 Mathematical equation (excluding rotation mapping), hence computational power is reduced, the Analytical Solver suffers from 1 big limitation that is, it cannot handle IK chains with more than 2 links (3 joints) because of the visual and trigonometric schemes it follows.

2.4 CCD Solver

CCD stands for Cyclic Coordinate Descent.

It's an iterative search Method that produces good results with low computer power.

One drawback is its tough to use CCD for multi-end-effectors system and get good result, but we work around that by breaking the system into smaller chains.

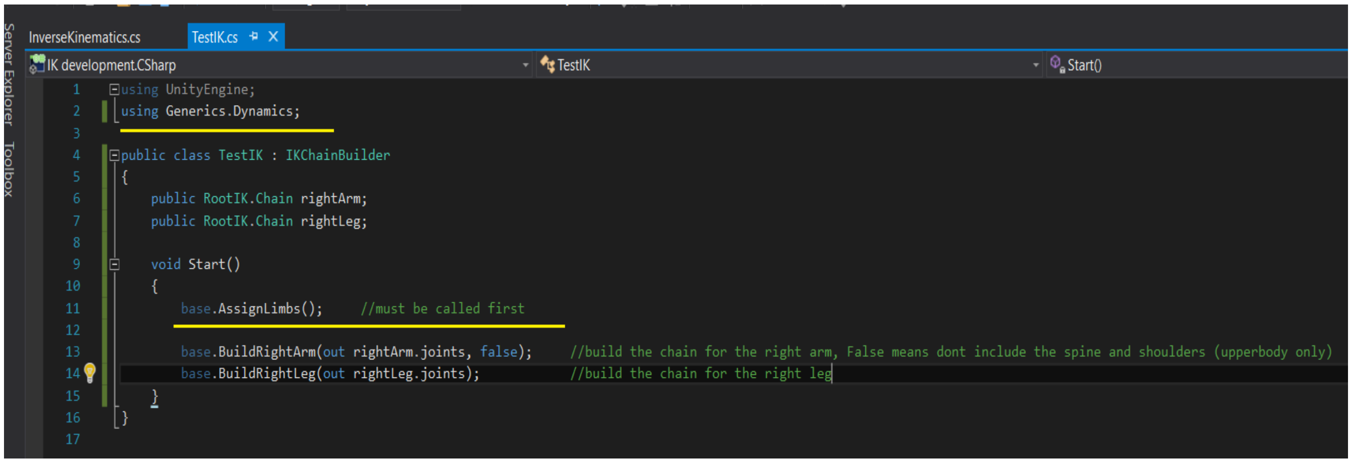
2.5 FABRIK Solver

FABRIK stands for Forward And Backward Reaching Inverse Kinematics, its also an iterative search method with an algorithm that simplifies the IK problem to some basic Vectors Math making it so fast even faster than the CCD, but as always not everything is peaches and cream. In FABRIK, we cannot specify how much influence 1 single joint in the chain has on the entire chain which gives you an IK chain with each joint fully influenced by the solver not by a mixture between the FK (the animations) and the IK like the CCD.

2.6 IK Chain Builder

Lets assume you have created a script for you humanoid character and still wants to use IK with code but you also don’t want to supply the joints transforms in the newly created field, don’t worry….IK Chain Builder got your back

- **first you need to inherit from IKChainBuilder.cs which is a monobehaviour so you still get all the mono juice.**
- **Then Call AssinLimbs()**
- **Then call the Build method base on the limb you want to build so for example.**



```
using UnityEngine;
using Generics.Dynamics;

public class TestIK : IKChainBuilder
{
    public RootIK.Chain rightArm;
    public RootIK.Chain rightLeg;

    void Start()
    {
        base.AssignLimbs();     //must be called first

        base.BuildRightArm(out rightArm.joints, false);     //build the chain for the right arm, False means dont include the spine and shoulders (upperbody only)
        base.BuildRightLeg(out rightLeg.joints);            //build the chain for the right leg
    }
}
```

3.0 Finally,

Note that mixing animations with IK produces the best results, so be aware that a raw IK solution “can” result in non-balanced representation of a complex game character’s rig.

for support and feedback contact me on my email moatyiv@gmail.com.