2D IK

This package contains tools which help to determine the translated positions of a group of transforms given a desired target position using inverse kinematics (IK). This will make it easier for you to pose characters for animation or aid you in manipulating characters in real-time.

Workflow

- Setup your hierarchy of Transforms for IK.
- Create a GameObject for IK.
- Add IKManager2D Component to the GameObject.
- In the IK Solvers menu, click on + to add your IKSolvers. This will create a GameObject for each new IKSolver.
- Setup your individual IK solvers. The parameters specific to each solver are listed below.
- If desired, setup effectors for each individual IK solver. This will be useful for having IK targets in runtime.
- After all IK solvers are created, arrange the IK solvers in priority in the IKManager2D. IKSolvers further down in the list have greater priority over previous IKSolvers.
- Pose your hierarchy of Transforms using effectors or pose gizmos in the SceneView.

...
Profit!

IK Manager

This component controls and drives all of the IK solvers for this package. Add this to a GameObject which will be used to control the IKs for a given hierarchy of Transforms. An IK solver without a IK manager will not work.

Properties:

- Weight
  This is the master weight which affects all solvers controlled by the manager. This weight will be applied to each solver's weight and will be used to blend between the results of the IK solver positions and the original positions of the Transforms affected.
- Solvers

This is a list of all solvers which are controlled by this IK manager. Arrange all IK solvers in this to run them in order of priority. IKSolvers further down in the list have greater priority over previous IKSolvers as they are iterated one by one in the order of the list.

Solvers

These are the default solvers that come with the package. Choose the right solver for your particular IK problem.

Solver2D

The following are properties which are common to all solvers:

- Constrain Rotation - This constrains the rotation of the target to the rotation of the effector.
- Restore Default Pose - This restores the original positions of the Transforms to its initial state before the IK solution is applied.
- Weight - The weight is used to blend between the results of the IK solver positions and the original positions of the Transforms affected. A value of 0 means that the IK solution is ignored while a value of 1 means that the IK solution is fully applied. This is multiplied with the manager's weight.

Limb

This is a standard two bone solver.

- Target - The transform to perform IK on to reach a desired position.
- Effector - The transform which is used as the desired position for the target.
- Flip - Use this to flip the result of the Limb solver.

CCD (Cyclic Coordinate Descent)

This solver uses the Cyclic Coordinate Descent algorithm to solve the IKs. This is an iterative solver which generates an approximate solution for the IKs and slowly converges to a better solution the more times the algorithm is run. Due to this an exact solution may not be reached and this solver will run to a predefined tolerance or number of runs.

- Target - The transform to perform IK on to reach a desired position.
- Effector - The transform which is used as the desired position for the target.
- Chain Length - The number of transforms involved in the IK solution starting from the target.
- Iterations - How many times the algorithm is run while it has not reached the desired position.

- Tolerance - How close the target solution must be to the desired position before the algorithm terminates.
- Velocity - How fast transforms in the chain iterate to the desired position. Having a high value means that the initial portions of the chain rotate quickly to the desired position, resulting in the later portions of the chain not rotating if the desired position has been reached. Having a low value means that the later portions of the chain are more involved in the IK solution.

FABRIK (Forward And Backward Reaching Inverse Kinematics)

This solver uses the Forward And Backward Reaching Inverse Kinematics algorithm to solve the IKs. This is also an iterative solver which generates an approximate solution for the IKs and slowly converges to a better solution the more times the algorithm is run. Due to this an exact solution may not be reached and this solver will run to a predefined tolerance or number of runs.

- Target - The transform to perform IK on to reach a desired position.
- Effector - The transform which is used as the desired position for the target.
- Chain Length - The number of transforms involved in the IK solution starting from the target.
- Iterations - How many times the algorithm is run while it has not reached the desired position.
- Tolerance - How close the target solution must be to the desired position before the algorithm terminates.

Composite FABRIK

This solver also uses the FABRIK algorithm to solve the IKs. The Composite FABRIK solver allows for more than one chain/s which must be linked togther with a common root Transform.

Root - The common root Transform which all chains must be linked to.
Chains - A list of targets and effectors which will be used to build the chains.

Adding New Solvers

You can add your own solver by extending from the class Solver2D. Your extended class will then show up as a new solver under the solver menu in the IKManager2D component.

Solver2D

This is the base class for all IK solvers in this package. IKManager2D will detect all classes extending this and accept it as a solver it can control. Implement or override the following methods to create your own IK solver:

- protected abstract int GetChainCount()
  This function returns the number of IK chains the solver owns. Use this to return the number of IK chains your solver owns.
- public abstract IKChain2D GetChain(int index)
  This function returns the IKChain2D at the given index. Use this to return the IKChain2D your solver owns at the given index.
- protected virtual bool DoValidate()
  This function does validation for all parameters passed into the solver. Use this to check if your solver is set up correctly with all inputs.
- protected virtual void DoInitialize()
  This function initializes the solver and builds the IK chains owned by the solver. This is called whenever the solver is invalid after changing the target of the solver or other parameters of the solver. Use this to do initialize all the data from the parameters given to the solver, such as the IK chains owned by the solver.
- protected virtual void DoPrepare()
  This function prepares and converts the information of the Transforms (position, rotation, IK parameters etc) to structures which can be used by the IK algorithms. Use this to do any work to gather data used by your solver when updating the IK positions.
- protected abstract void DoUpdateIK(List effectorPositions)
  This function calculates and sets the desired IK positions for the Transforms controlled by the solver given a list of effector positions for each chain owned by the solver. The effector positions may be overridden by user positions if manipulated from the SceneView.
- protected virtual Transform GetPlaneRootTransform()
  This function returns the transform whose localspace XY plane is used to perform IK calculations. Use this to define the Transform used.

IKChain2D

This is the class which is used to store the transforms involved in an IK chain. When a chain is set up with a target and a transform count, the solver's Initialize will populate the chain with the right transforms if valid.

- Target - The transform to perform IK on to reach a desired position.
- Effector - The transform which is used as the desired position for the target.
- TransformCount - The number of transforms involved in the IK solution starting from the target. This is generally equivalent to ChainLength in solvers.
- Transforms - All transforms involved in the chain. In general, the last transform in this is the target transform and the first transform is considered the root transform for the chain.

- Lengths - The lengths between each transform in the chain.

Solver2DMenu

This attribute allows you to tag your Solver2D with a different name under the IKManager2D. Use this if you do not want to use the name of the class of the Solver2D.

Example for LimbSolver2D:[Solver2DMenuAttribute("Limb")]