# Action tree construction

## TreeStage

Manages **TreeWorkers**. When a goal condition is met, it freezes the workers for future use by other stages.

Implementations:

**TreeStage_FixedGames**
Builds until a certain number of games have been played.

**TreeStage_MaxDepth**
Builds until any branch reaches a specified depth.

**TreeStage_MinDepth**
Builds until all unfailed branches reach at least a specified depth.

**TreeStage_SearchForever**
Build forever. No termination condition.

## WorkerFactory

Makes new **TreeWorker**s within the Pool system. Workers can be returned to the pool and reassigned to different **TreeStages**.

## ActionQueue

Stores **Action**s so they can be polled to get a set of booleans for each timestep of simulation by **GameLoader**

## GameLoader

QWOP game implementation using the JBox2D library. Each instance runs on its own class loader so many copies can be used without collisions.Can send commands to the game, and receive state information back.

## TreeWorker

**TreeWorker**s tell the game simulation what to do, the ask the **ISampler** goal nodes to reach, and report information to an **IDataSaver**. Many **TreeWorker** instances can operate on the same tree simulatneously

## State

72-number runner state. 3-DOF for 12 links, plus velocities.Usually obtained from a **GameLoader.** The state is divided up into **StateVariable**s for each body (6 numbers each).

## Node

Base unit of the tree. Each node represents a **State** and the **Action** it took to get from the parent node's state to this node's state.

Most users of Node keep track of the root node, and traverse the tree from there.

## Action

Keys and durations which take the runner from the **State** at the parent node to the state at thiss node.

## ISampler

Rules that each **TreeWorker** follows for traversing the existing game tree and adding nodes. It is split into *tree policy, expansion policy,* and *rollout policy.* Not every sampler uses all of these.

Implementations:

**Sampler_Deterministic**
Depth-first search with no random selection.

**Sampler_Distribution**
Sample purely according to the distribution rules in the **ActionSet** assigned to the nodes.

**Sampler_FixedDepth**
Sampler only makes workers build to a certain depth before instructing them to start over.

**Sampler_Greedy**
Singles out promising areas of the tree and only builds those. Similar to the old sampler from the early days.

**Sampler_Random**
All choices are random.

**Sampler_UCB**
Upper confidence bound for trees. Best overall sampler with adjustable weighting on exploration/exploitation.

## IDataSaver

Receives tree/game information on a per timestep, per game, and per stage basis. Not all of these are used by every **IDataSaver** implementation.

Implementations:

**DataSaver_Dense**
Full state and action information saved at every timestep...
- **DataSaver_DenseJava**
  ... as serialized Java objects.
- **DataSaver_DenseTFRecord**
  ... as binary Protobuf files in TFRecord form.

**DataSaver_Null**
A placeholder which does nothing.

**DataSaver_Sparse**
Saves per-run information, enough to re-create games, but not info at every timestep.

**DataSaver_StageSelected**
Saves only information sent by the tree stage upon its completion.

## ActionSet

Sets of actions and a **Distribution** on which to choose them.

## Distribution

Rules for choosing from an **ActionSet**.

Implementations:

**Distribution_Equal**
Even chance of picking any **Action** from the set.

**Distribution_Normal**
Normal distribution set around some duration mean and standard deviation.

## IActionGenerator

Picks "potential actions," which could be used from a node, These are **ActionSets**. when it is created. These are used when expanding from a node, or determining if all options have been exhausted at a node.

Implementations:

**ActionGenerator_FixedSequence**
This can provide sets of actions which rotate with depth in the tree. There may be exceptions specified at whatever depths.

## IEvaluationFunction

Evaluate a value or score for **Node**s.

Implementations:

**EvaluationFunction_Constant**
Constant score regardless. Mostly as a placeholder.

**EvaluationFunction_Distance**
Evaluation based solely on torso x position.

**EvaluationFunction_HandTunedOnState**
Combinations of factors hand-picked.

**EvaluationFunction_Random**
Assigns a random score. Mostly for testing.

**EvaluationFunction_SqDistFromOther**
Distance in state space (squared) of the Node's state from another specified state.

## TFRecordWriter

Handles turning the Protobuf objects into binaries that TensorFlow will like.

## SavableFileIO

Serializes and writes Java objects to file. Can also load and manipulate these files.

# Graphical user interface

## IUserInterface

All graphing, tree visualization, etc. managed here.

Implementations:

**UI_Full**
All plot panels, tree visualization, etc.

**UI_Headless**
Placeholder for when UI is not needed (e.g. running from command line).

## TabbedPaneActivator

UI elements which can be turned on/off/updated.

## PanelTree

OpenGL rendering of the tree. Usually the main panel displayed in a **IUserInterface**.

## PanelTimeSeries

Scrolling time-series plots. Nice for real-time monitoring of search status, etc. Uses JFreeChart.

**PanelTimeSeries_WorkerLoad**
Monitors the productivity of **TreeWorker**s. Plots timesteps simulated per second.

## PanelPlot

Scatterplot abstract class. Uses JFreeChart.

**PanelPlot_Controls**
Various plots of controls.

**PanelPlot_SingleRun**
States along a single run.

**PanelPlot_States**
Plots of states vs. any other.

**PanelPlot_Transformed**
Plotting states which have been transformed.

## PanelRunner

Stick-figure runner posed to display states.

**PanelRunner_SimpleState**
Display the runner posed in a specific state, selected from a **Node**.

**PanelRunner_Snapshot**
Display the runner, and its past/future during a run sequence. Possibly all potential futures.

**PanelRunner_Comparison**
Find and display states similar to the selected one.

## INodeFilter

Trims down a list of **Nodes** based on some criteria.

**NodeFilter_Downsample**
Evenly reduces the number of nodes. Good for when plotting 100k points is too slow.

**NodeFilter_Identity**
Placeholder which does not alter the list of nodes..

**NodeFilter_MissingInfo**
Filter out nodes which are missing state or action information.

**NodeFilter_SurvivalHorizon**
Keep only nodes which have at least a certain depth of nodes

## ITransform

Transform states to a set of numbers and potentially back again. Mostly state reduction.

**Transform_Autoencoder**
TensorFlow-trained autoencoder.

**Transform_Identity**
Placeholder which does not alter the state.

**Transform_PCA**
Principal component analysis of a bunch of states, turned into a transform for others.

## PanelRunner_Animated

Stick-figure runner animated. This base version can re-simulate from a sequence of actions in order to animate the runner.

**PanelRunner_AnimatedFromStates**
Animate from an existing set of states. Don't need to resimulate.

**PanelRunner_AnimatedTransformed**
Animate the runner based on states which have been compressed and then decompressed from an **ITransform**.

# Feedback Control

## IController

Feedback controller. **State** in, **Action** out.

Implementations:

**Controller_NearestNeighborApprox**
Uses a large trajectory library.

**Controller_Null**
Always says to do nothing!

**Controller_Tensorflow_ClassifyActionsPerTimestep**
Classifies which combination of keys should be pressed given a **State** input.