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**.

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).

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

TreeWorkers 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

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.

MORE TODO

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.

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.

Action

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

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.

ActionSet

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

IActionGenerator

Picks "potential actions," which could be used from a node, These are **ActionSet**s. 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.

Text