Change Log for DWL Code

# 11/7/12

Adam- Added DWLAgent. This class is everything needed for an ‘out of the box’ DWL implementation.

Adam- Added Policy. This encapsulates a way of selecting actions. Not necessarily containing a DWL process. This should allow for agent type heterogeneity.

Adam- Added WLearningProcess. This has everything that is needed for one policy in a DWL agent without any of the collaboration stuff.

Adam- Added Action class. This was done as I was getting annoyed with not being able to instantiate AbstractAction.

# 12/7/12

Adam- Added NeighbourReward. This allows a single value to be used as a reward function.

Adam- Added ColaberationDOP. This is a general dop that can select collaboration coefficients.

# 27/7/12

Fixed few bugs and lost the notes for it. One was in reward; it is just a change of order to fix a lost update. The rest where in the collaboration and distribution system.

# High level view

Agent

Select action from all policies and pick winner

Execute this though a simulator and wait for results

----------------Barrier: Leave on action results

Write what agent did and the reward.

Read the neighbours updates

----------------Barrier: Leave on received all update

Learn from them

----------------Barrier: Leave on learning complete\*

\*optional as probably will not impact on the next state, could happen anytime until state is actually used. Batching of updates may be needed if comms are bad.

# To use

## Set up

Create a DWL Agent

Overload a DOP and Reward as required for application.

Call addLocalPolicy on the agent, passing in the reward and DOP. The name passed should represent Agnet name and policy name in multi policy environments.

Call addRemotePolicy for any policies implemented by neighbours. Naming should represent the policy’s implementer and its name on that agent.

Call createColaberationPolicy the arguments represent how many times to break the range of c and the maximum of this range. For example 4,1 would give 0, .25, .5, .75, 1.

## Running

Call nominate on an agent. Give this action to Gridlab-D.

Call updateLocal on the agent, passing in the Gridlab-D results. This will learn based on the local policies’ reward functions.

Call addMessage on an agent, passing in the designation agent’s message (this can be accessed by writeMessage). In effect this will tell the first agent what the second agent did how good it was.

Call updateRemote, this will use the information exchanged in communications to learn and set correct states for remote policies.

## Optional

Call the purge functions (can be done through the finish run function). These clear vectors and manage memory. UPDATE, mostly done elsewhere, now just comms need cleaning

## Notes:

Naming should be checked and improved upon.

On Mac the file writing needs to be changed to its assways method. Mostly in DWL->writePolicies

Neighbour sets are not yet handled by the DWL agent so it is user’s responsibility.

~~Messages are based on KeyValuePairs, this is probably too specific. This may need to be changed in the future.~~

Most documentation is in pseudo-java doc as my IDE processes this well.

~~Make collaboration policy learn this requires total reward to be maintained.~~

~~Need to comprehensively check that the right vales are being updated at the right times and state are correct. This seems ok provisionally.~~