/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Ramin Edjlal Copyrights 2015.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Learning Autamata.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* The every sum of probability is one.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*\_)

\* four formula .tow for Regard regime and tow for penalty regime.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(-)

\* Derived Quantum Automata Penalty All Objects of Derived Automata\*\*\*\*\*\*\*\*\*\*\*\*(-)

\* Malfunction Reward and Penalty Detection\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\_\*)

\* Penalty Reward Action Failure\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*\_)

\* Mistuning of Penalty and Regard Data in IsRegard and IsPenalty Values\*\*\*\*\*\*\*(+)

\* No Reason For Malfunction of Reward and Penalty Mechanism\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\_)

\* 1395/1/2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(\*:Sum(3)) (\_:Sum(4)) (-:Sum(2)) (All Errors:(7))

\* Penalty Regard Action is Useful For One Time Per AllDraw Object.\*\*\*\*\*\*\*\*\*\*\*\*

\* No Solution to Overcome to static Behavior Of Quantum Variables Inhererete.\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace LearningMachine

{

[Serializable]

public class LearningKrinskyAtamata

{

int r = 100;

int m = 100;

int k = 100;

public double[] alpha;

bool beta = true;

double[] fi;

public bool IsReward = false;

public bool IsPenalty = false;

protected double Reward;

protected double Penalty;

protected int Success = 0, Failer = 0;

protected int State = 0;

//int State = 1;

public void Initiate()

{

Object o = new Object();

lock (o)

{

IsPenalty = false;

IsReward = false;

}

}

public LearningKrinskyAtamata(int r0, int m0, int k0)

{

Object o = new Object();

lock (o)

{

IsReward = new bool();

IsPenalty = new bool();

IsReward = false;

IsPenalty = false;

Success = new int();

State = new int();

beta = new bool();

beta = true;

Reward = new double();

Penalty = new double();

r = new int();

m = new int();

k = new int();

if (r0 >= m0)

{

r = r0;

m = m0;

k = k0;

alpha = new double[r];

fi = new double[k];

fi = new double[r];

for (int i = 0; i < r; i++)

alpha[i] = 1.0 / (double)r;

for (int i = 0; i < k; i++)

fi[i] = 1.0 / (double)k;

//Reward[i] = (double)(new Random()).Next(0, 100000) / 100000.0;

Reward = 1.0 / (double)r;

//Penalty[i] = (double)(new Random()).Next(0, 100000) / 100000.0;

Penalty = 1.0 / (double)r;

}

}

}

public void Clone(ref QuantumAtamata AA)

{

Object o = new Object();

lock (o)

{

AA.r = this.r;

AA.m = this.m;

AA.k = this.k;

alpha = new double[AA.r];

for (int i = 0; i < AA.r; i++)

AA.alpha[i] = this.alpha[i];

AA.beta = this.beta;

AA.Failer = this.Failer;

fi = new double[AA.k];

for (int i = 0; i < AA.k; i++)

AA.fi[i] = this.fi[i];

AA.IsPenalty = this.IsPenalty;

AA.IsReward = this.IsReward;

AA.Reward = this.Reward;

AA.Penalty = this.Penalty;

AA.Success = this.Success;

AA.Failer = this.Failer;

AA.State = this.State;

}

}

public void FailureState()

{

Object o = new Object();

lock (o)

{

Failer++;

if (Success < Failer && State < r - 1)

State++;

else if (State > 0)

State--;

}

}

public void SuccessState()

{

Object o = new Object();

lock (o)

{

Success++;

if (Success > Failer && State < r - 1)

State++;

else if (State > 0)

State--;

}

}

public int IsSecondDerivitionIsPositive()

{

Object o = new Object();

lock (o)

{

for (int i = 0; i < r - 2; i++)

{

if (((alpha[i + 2] - 2 \* alpha[i + 1] + alpha[i]) / (1.0 / (double)r)) < 0)

return -1;

}

return 1;

}

}

public double LearningAlgorithmRegard()

{

Object o = new Object();

lock (o)

{

SuccessState();

this.IsReward = true;

this.IsPenalty = false;

alpha[State] += Reward \* (1 - alpha[State]);

for (int i = 0; i < r; i++)

if (i != State)

alpha[i] -= Reward \* alpha[i];

beta = false;

return alpha[State];

}

}

public int IsRewardAction()

{

Object o = new Object();

lock (o)

{

if (this.IsReward)

return 1;

return -1;

}

}

public double IsPenaltyAction()

{

Object o = new Object();

lock (o)

{

if (this.IsPenalty)

return 0;

return -1;

}

}

public double LearningAlgorithmPenalty()

{

Object o = new Object();

lock (o)

{

FailureState();

this.IsPenalty = true;

this.IsReward = false;

alpha[State] -= Penalty \* alpha[State];

for (int i = 0; i < r; i++)

if (i != State)

{

alpha[i] -= Penalty \* alpha[i];

alpha[i] += (Penalty / (r - 1));

}

beta = true;

return alpha[State];

}

}

}

}