```
//*****************************************************************************¥¥
¥¥* FILE NAME: MultiAgent.c                                                    *//
//* PROGRAM NAME: Multi Agent Modeling                                          *¥¥
¥¥* Version 1.01                                                               *//
//*****************************************************************************¥¥
¥¥*                                                                            *//
//* (c) S. Barbaq Quarmal & Satoru Ozawa                                       *¥¥
¥¥* Hitachi, 2011-12                                                          *//
//*                                                                            *¥¥
¥¥*****************************************************************************//

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include "sgl.h"              //Include files for graphics
#define r 0.005               // Radius of disc
#define r2 0.0055             // Radius of disc
#define n 100                 // Number of steps
#define N 10                  // Number of the 'knowledge sufixes' to be picked up by random number
#define A 4                   // Number of Agents
#define turnMax 15            // Number of message exchanges between the agents in a certain period
#define fact 5.      // A parameter used to keep the graphic output within the boundary of X-window

void wait ( int seconds )          //time function used for putting intervals in order to make movie like presentation
{
  clock_t endwait;
  endwait = clock() + seconds * CLOCKS_PER_SEC/2;
  while (clock() < endwait){
  }
}

int main(int argc, char *argv [])
{
  int a, i, j, l, flag[A], c[A], win1, win2, sender, receiver, turn, time;
  int CHNL0CNT, CHNL1CNT, CHNL2CNT, CHNL3CNT, CHNL4CNT, CHNL5CNT, CHNL6CNT,
CHNL7CNT;

/*random number generator seeds (all the seeds used in the study)*/
  //unsigned short int parmvec[7]={0x0, 0x0, 0x1, 0xABDF, 0xFBEC, 0xAFD, 0xF}; //seed 1
  //unsigned short int parmvec[7]={0x0, 0x0, 0x0, 0x6DEF, 0x7FFF, 0x4FFE, 0xB}; //seed 2
  //unsigned short int parmvec[7]={0x0, 0x0, 0x1, 0x65ED, 0x7EDF, 0x4DFF, 0xC}; //seed 3
```

```
  //unsigned short int parmvec[7]={0x0, 0x0, 0x1, 0xFBCF, 0xABDE, 0xACBC, 0xD}; //seed 4
  //unsigned short int parmvec[7]={0x0, 0x0, 0x1, 0x6AFD, 0x7FEE, 0x4EEF, 0xD}; //seed 5
  //unsigned short int parmvec[7]={0x0, 0x0, 0x1, 0xDBA7, 0xABCD, 0xCEF3, 0xC}; //seed 6
  //unsigned short int parmvec[7]={0x0, 0x0, 0x1, 0xABDF, 0xABEC, 0xAFD, 0xA}; //seed 7
  //unsigned short int parmvec[7]={0x0, 0x0, 0x1, 0xDCF, 0xACD, 0xCFA, 0xB}; //seed 8
  //unsigned short int parmvec[7]={0x0, 0x0, 0x1, 0x2DCB, 0x4AEC, 0x3AFA, 0xB}; //seed 9
  //unsigned short int parmvec[7]={0x0, 0x0, 0x1, 0x4FEF, 0x4FDE, 0x2DEC, 0xB}; //seed 10
  //unsigned short int parmvec[7]={0x0, 0x0, 0x0, 0xABD3, 0xBAE5, 0xACF9, 0xB}; //seed 11
  //unsigned short int parmvec[7]={0x0, 0x0, 0x2, 0xBED3, 0xDAD2, 0xCEC7, 0xA}; //seed 12
  //unsigned short int parmvec[7]={0x0, 0x0, 0x1, 0xDBAF, 0xABCD, 0xCEFA, 0xB}; //seed 13
  //unsigned short int parmvec[7]={0x0, 0x0, 0x2, 0xABBD, 0xCFFD, 0xEDC3, 0xC}; //seed 14
  //unsigned short int parmvec[7]={0x0, 0x0, 0x1, 0xBADF, 0xACDE, 0xCABD, 0xA}; //seed 15
  //unsigned short int parmvec[7]={0x0, 0x0, 0x1, 0xCBEB, 0xEBFD, 0xFDBA, 0xA}; //seed 16
  //unsigned short int parmvec[7]={0x0, 0x0, 0x1, 0x2DCB, 0x3CEB, 0x4CAD, 0xB}; //seed 17
  //unsigned short int parmvec[7]={0x0, 0x1, 0x0, 0x7ADF, 0x2BDC, 0x4DFB, 0xB}; //seed 18
  //unsigned short int parmvec[7]={0x0, 0x0, 0x1, 0x2ABD, 0x3BAE, 0x5FDB, 0xB}; //seed 19
  //unsigned short int parmvec[7]={0x0, 0x0, 0x2, 0xBDC3, 0xDAC7, 0xADB2, 0xB}; //seed 20
  //unsigned short int parmvec[7]={0x0, 0x0, 0x0, 0xBDC3, 0xDAC7, 0xADB2, 0xA}; //seed 21
  //unsigned short int parmvec[7]={0x0, 0x0, 0x2, 0xBDC3, 0xDAC5, 0xADB5, 0xD}; //seed 22
  //unsigned short int parmvec[7]={0x0, 0x0, 0x2, 0xBDC3, 0xDAC5, 0xADB5, 0xA}; //seed 23
  //unsigned short int parmvec[7]={0x0, 0x0, 0x0, 0xBDC5, 0xDAC5, 0xADB5, 0xF}; //seed 24
  //unsigned short int parmvec[7]={0x0, 0x0, 0x2, 0xBDC5, 0xDAC5, 0xADB5, 0xA}; //seed 25
  //unsigned short int parmvec[7]={0x0, 0x0, 0x2, 0xBDA7, 0xDBC5, 0xADB2, 0xA}; //seed 26
  //unsigned short int parmvec[7]={0x0, 0x0, 0x0, 0xBAD, 0xFEC5, 0xACB4, 0xB}; //seed 27
  //unsigned short int parmvec[7]={0x0, 0x0, 0x1, 0x2BEF, 0xEED5, 0xFCE4, 0x9}; //seed 28
  //unsigned short int parmvec[7]={0x0, 0x0, 0x1, 0x2BEF, 0xEDD5, 0xFCE4, 0xB}; //seed 29

  double Rand, drand48(), dx, ch;
  double ksum[A], Ksum[A], Xmean[A], Xvar2[A], Sigma[A], dsum[A], msgsum[A], kmodsum[A];
  double T[A][n+2], X[A][N+2], d[A][n+2], msg[A][n+2], kmod[A][n+2];
double COK[A], COKmod[A], PeakValue[A], PeakValueMod[A], PeakValuePosition[A],
PeakValuePositionMod[A];
double k[A][n+2]=       /* Array for the knowledge functions of the agents (obtained from questionnaire
survey) */
  {
    {
0, 0.000399755, 0.00079951, 0.001199264, 0.001599019, 0.001998774, 0.002398529, 0.002798284, 0.003198039,
0.003464542, 0.003731045, 0.003997548, 0.004264051, 0.004530555, 0.004797058, 0.005063561, 0.005330064,
0.005529942, 0.005729819, 0.005929697, 0.006129574, 0.006529329, 0.006929084, 0.007195587, 0.00746209,
0.007728593, 0.007995096, 0.0082616, 0.008528103, 0.008794606, 0.009061109, 0.009327613, 0.009660742,
0.009993871, 0.010327, 0.010660129, 0.010993258, 0.011326387, 0.011659516, 0.011992645, 0.012072596,
0.012152547, 0.012232498, 0.012312449, 0.012392399, 0.01247235, 0.012552301, 0.012632252, 0.012712203,
```

0.012792154, 0.012792154, 0.012792154, 0.012658903, 0.012525651, 0.012392399, 0.012259148, 0.01247235, 0.012685553, 0.012898756, 0.013111958, 0.013325161, 0.013538363, 0.013751566, 0.013964768, 0.014177971, 0.014391174, 0.014657677, 0.01492418, 0.015190683, 0.015457186, 0.01572369, 0.015990193, 0.016256696, 0.016523199, 0.016789703, 0.017011788, 0.017233873, 0.017455958, 0.017678043, 0.017900128, 0.018122219, 0.017216108, 0.016309997, 0.015403886, 0.014497775, 0.013591664, 0.012685553, 0.011779442, 0.010873331, 0.00996722, 0.009061109, 0.008154998, 0.007248887, 0.006342777, 0.005436666, 0.004530555, 0.003624444, 0.002718333, 0.001812222, 0.000906111, 0

},

{

0, 0.0004111, 0.000822201, 0.001233301, 0.001644401, 0.002055502, 0.002466602, 0.002877702, 0.003288802, 0.00342583, 0.003562858, 0.003699886, 0.003836914, 0.003973942, 0.00411097, 0.004247998, 0.004385026, 0.004522054, 0.004659082, 0.00479611, 0.004933204, 0.004933204, 0.004933204, 0.005029927, 0.005126651, 0.005223375, 0.005320098, 0.005416822, 0.005513546, 0.005610269, 0.005706993, 0.005803717, 0.00590044, 0.005997164, 0.006093888, 0.006190612, 0.006287335, 0.006384059, 0.006480783, 0.006577605, 0.006742045, 0.006906485, 0.007070925, 0.007235365, 0.007399806, 0.007564246, 0.007728686, 0.007893126, 0.008057566, 0.008222006, 0.009044207, 0.009866407, 0.010277508, 0.010688608, 0.011099708, 0.011510809, 0.012058937, 0.012607065, 0.01315521, 0.013625032, 0.014094854, 0.014564675, 0.015034497, 0.015504319, 0.015974141, 0.016444012, 0.016992141, 0.017540269, 0.018088414, 0.018636542, 0.01918467, 0.019732798, 0.020280927, 0.020829055, 0.021377216, 0.021925344, 0.022473472, 0.023021601, 0.023569729, 0.024117857, 0.024666018, 0.023432717, 0.022199417, 0.020966116, 0.019732815, 0.018499514, 0.017266213, 0.016032912, 0.014799611, 0.01356631, 0.012333009, 0.011099708, 0.009866407, 0.008633106, 0.007399806, 0.006166505, 0.004933204, 0.003699903, 0.002466602, 0.001233301, 0

},

{

0, 0.000523015, 0.001046029, 0.001569044, 0.002092059, 0.002615074, 0.003138088, 0.003661103, 0.004184118, 0.004707133, 0.005230147, 0.005753162, 0.006276177, 0.006799192, 0.007322206, 0.007845221, 0.008368236, 0.009065589, 0.009762942, 0.010460295, 0.011157648, 0.011855001, 0.012552354, 0.012859189, 0.013166024, 0.01347286, 0.013779695, 0.01408653, 0.014393366, 0.014700201, 0.015007036, 0.015341766, 0.01394706, 0.012552354, 0.011157648, 0.009762942, 0.009065589, 0.008368236, 0.007670883, 0.00697353, 0.007113, 0.007252471, 0.007391942, 0.007531412, 0.007670883, 0.007810353, 0.007949824, 0.008089295, 0.008228765, 0.008368236, 0.008368236, 0.008368236, 0.008716912, 0.009065589, 0.009414265, 0.009762942, 0.010223195, 0.010683448, 0.011157648, 0.011355696, 0.011553744, 0.011751793, 0.011949841, 0.012147889, 0.012345937, 0.012552354, 0.013012607, 0.01347286, 0.01394706, 0.014407313, 0.014867566, 0.015327819, 0.015788072, 0.016248325, 0.016736472, 0.016967993, 0.017199514, 0.017431035, 0.017662556, 0.017894078, 0.018131178, 0.017224619, 0.01631806, 0.015411501, 0.014504942, 0.013598383, 0.012691824, 0.011785265, 0.010878707, 0.009972148, 0.009065589, 0.00815903, 0.007252471, 0.006345912, 0.005439353, 0.004532794, 0.003626236, 0.002719677, 0.001813118, 0.000906559, 0

},

{

0, 0.001182564, 0.002365129, 0.003547693, 0.004730258, 0.005912822, 0.007095387, 0.008277951, 0.009460515, 0.009697028, 0.009933541, 0.010170054, 0.010406567, 0.01064308, 0.010879593, 0.011116106, 0.011352619, 0.011589131, 0.011825644, 0.012062157, 0.01229867, 0.012771696, 0.013244722, 0.013665185, 0.014085648,

```
0.014506111, 0.014926574, 0.015347037, 0.0157675, 0.016187964, 0.016608427, 0.017028928, 0.017738466,
0.018448005, 0.019157544, 0.019867082, 0.020103595, 0.020340108, 0.020576621, 0.020813134, 0.020150898,
0.019488662, 0.018826426, 0.01816419, 0.017501954, 0.016839717, 0.016177481, 0.015515245, 0.014853009,
0.014190773, 0.013244722, 0.01229867, 0.011589131, 0.010879593, 0.010170054, 0.009460515, 0.008829821,
0.008199126, 0.007568412, 0.007433269, 0.007298125, 0.007162982, 0.007027838, 0.006892695, 0.006757552,
0.006622361, 0.006622361, 0.006622361, 0.006622361, 0.00678003, 0.006937699, 0.007095368, 0.007253037,
0.007410706, 0.007568412, 0.007647247, 0.007726081, 0.007804916, 0.00788375, 0.007962585, 0.008514464,
0.008088741, 0.007663017, 0.007237294, 0.006811571, 0.006385848, 0.005960125, 0.005534402, 0.005108678,
0.004682955, 0.004257232, 0.003831509, 0.003405786, 0.002980062, 0.002554339, 0.002128616, 0.001702893,
0.00127717, 0.000851446, 0.000425723, 0
      }
   }
//******** Initialization ********¥¥
¥¥*******************************//

   sgl_winsize(500,500);                // set window size to 500x500
   sgl_initm(2);                        // initialize graphics
   win1 = 0;                            // window index starts at 0
   win2 = 1;
   sgl_select_win(win1);
   sgl_pos_win(0,0);                    // window1 at top left corner of screen
   sgl_color_background(sgl_colordef("RoyalBlue3"));   //background color is set to Royal Blue
sgl_select_win(win2);
sgl_pos_win(550,0);                     // window2 on the right of window1
sgl_color_background(sgl_colordef("RoyalBlue3"));

   for(a=0; a<A; a++){
      COK[a]=0.;
      COKmod[a]=0.;
      PeakValue[a]=0.;
      PeakValueMod[a]=0.;
      PeakValuePositionMod[a]=0.;
      ksum[a]=0.;
      Ksum[a]=0.;
      Sigma[a]=0.;
      dsum[a]=0.;
      msgsum[a]=0.;
      kmodsum[a]=0.;
}

   for(a=0; a<A; a++){
      for(j=0; j<N; j++){
```

```
        X[a][j]=0.;
      }
   }
      for(a=0; a<A; a++){
      for(i=0; i<n; i++){
        T[a][i]=0.;
        d[a][i]=0.;
        msg[a][i]=0.;
        kmod[a][i]=0.;
      }
}
   dx=1./(float)n;
   c[0] = sgl_colordef("Green2");              //color for Agent[0] >>> GP
   c[1] = sgl_colordef("Red");                 //color for Agent[2] >>> GOV
   c[2] = sgl_colordef("GhostWhite");                  //color for Agent[3] >>> MM
c[3] = sgl_colordef("Yellow");                 //color for Agent[4] >>> SNM

sgl_select_win(win1);

//*** Initial Knowledge Functions ***¥¥
¥¥*************************************//
   for(a=0; a<A; a++){      //Normalization
      ksum[a]=0.;
      for(i=0; i<n; i++){
        ksum[a] = ksum[a] + k[a][i]*dx;
      }
}
   for(a=0; a<A; a++){
      for(i=0; i<n; i++){
        k[a][i] = k[a][i]/ksum[a];
        sgl_color(c[a]);
        sgl_disc(dx*i, k[a][i]/fact, r);
      }
}

//*** Calculating Center of initial Knowledge Functions ***¥¥

   for(a=0; a<A; a++){
      COK[a]=0.;
      for(i=0; i<n; i++){
        COK[a]=COK[a]+i*dx*k[a][i]*dx;
      }
```

```
    COK[a]=COK[a];
    printf("COKinit[%d]=%f ¥n", a, COK[a]);
  }
//*** Calculating the Pick value position of the knowledge functions ***¥¥
¥¥*****************************************************************************//
  for(a=0; a<A; a++){
    PeakValue[a]=k[a][0]*dx;
    PeakValuePosition[a]=0.;
    for(i=0; i<n; i++){
      if (k[a][i]*dx >= PeakValue[a]){
        PeakValue[a]= k[a][i]*dx;
        PeakValuePosition[a]= i*dx;
      }
     else{
      PeakValue[a]= PeakValue[a];
      PeakValuePosition[a]= PeakValuePosition[a];
      }
    }
  printf("PVinit[%d]=%f ¥n", a, PeakValue[a]);
  }
  for(a=0; a<A; a++){
    printf("PVPinit[%d]=%f ¥n", a, PeakValuePosition[a]);
  }
  sgl_display();
 sgl_select_win(win2);
/*** Selecting Channel ***/
CHNL0CNT=0;
CHNL1CNT=0;
CHNL2CNT=0;
CHNL3CNT=0;
CHNL4CNT=0;
CHNL5CNT=0;
CHNL6CNT=0;
CHNL7CNT=0;
//lcong48(parmvec);         // initialize random number generator. drand48() will be initialized with
// custom seeds if this statement is active
// drand48() will use buit-in initializer if this statement is commented
  for(turn=0; turn<turnMax; turn++){
    Rand= drand48();
    if(Rand <= 0.32){ //Channel_GOV >>> MM
      sender= 1;
      receiver= 2;
```

```
        ch=1.5;
        CHNL0CNT= CHNL0CNT+1;
    }

    else if(Rand> 0.32 && Rand<= 0.45){ //Channel_MM >>> GOV
        sender= 2;
        receiver= 1;
        ch=0.6;
        CHNL1CNT= CHNL1CNT+1;
    }

    else if(Rand>=0.45 && Rand<0.77){ //Channel_MM >>> GP
        sender= 2;
        receiver= 0;
        ch=1.5;
        CHNL2CNT= CHNL2CNT+1;
    }
    else if(Rand>=0.77 && Rand<0.82){ //Channel_MM >>> SNM
        sender= 2;
        receiver= 3;
        ch=0.21;
        CHNL3CNT= CHNL3CNT+1;
    }

else if(Rand>=0.82 && Rand<0.89){ //Channel_SNM >>> MM
        sender= 3;
        receiver= 2;
        ch=0.35;
        CHNL4CNT= CHNL4CNT+1;
    }
    else if (Rand> 0.89 && Rand <=0.96){ //Channel_SNM >>> GP
        sender= 3;
        receiver= 0;
        ch=0.35;
        CHNL5CNT= CHNL5CNT+1;
    }
    else if(Rand> 0.96 && Rand <=0.98){ //Channel_GP >>> SNM
        sender= 0;
        receiver= 3;
        ch=0.05;
        CHNL6CNT= CHNL6CNT+1;
    }
```

```
    else{ //Channel_GP >>> MM
      sender= 0;
      receiver= 2;
      ch=0.08;
      CHNL7CNT= CHNL7CNT+1;
    }

//*** Picking Up Knowledge ***¥¥
¥¥*********************************//
    for(a=0; a<A; a++){
      if(a==sender){
        T[sender][0]=k[sender][0]*dx;
        for(i=1; i<n; i++){
          T[sender][i]= T[sender][i-1]+ k[sender][i]*dx;
        }
        Ksum[sender]=0.;
        for(j=0; j<N; j++){
          flag[sender]=0;
          Rand=drand48();
          i=0;

          while (flag[sender]==0){
            if(Rand<T[sender][i]){
              X[sender][j]= (i+1)*dx;
             flag[sender]=1;
            }
            i++;
          }
          Ksum[sender]+= X[sender][j];
        }
        Xmean[sender]=0.;
        Xmean[sender]= Ksum[sender]/N;
        Xvar2[sender]=0.;
        for(j=0; j<N; j++){
          Xvar2[sender]= Xvar2[sender]+(Xmean[sender]-X[sender][j])*(Xmean[sender]-X[sender][j]);
        }
        Xvar2[sender]=Xvar2[sender]/N;
        Sigma[sender]=0.;
        Sigma[sender]= sqrt(Xvar2[sender]);

//*************************¥¥
¥¥*** Making Decision ***//
```

```
            for(i=0; i<n; i++){
               d[sender][i] = exp(-(dx*i-Xmean[sender])*(dx*i-Xmean[sender])/(Sigma[sender]*Sigma [sender]));
            }
            dsum[sender]=0.;
            for(i=0; i<n; i++){
               dsum[sender] =dsum[sender]+d[sender][i]*dx;
            }
            for(i=0; i<n; i++){ //normalization
               d[sender][i]= d[sender][i]/dsum[sender];
               //sgl_color(c[sender]);   // these statements are to show the
               //sgl_disc(dx*i, d[sender][i]/fact, r2);       // decision functions and can be commented out if not
necessary.
          }




/***************************/
/*** Making Message ***/

            for(i=0; i<n; i++){
               msg[sender][i]= k[sender][i]+d[sender][i];
            }
            msgsum[sender]=0.;
            for(i=0; i<n; i++){
               msgsum[sender] = msgsum[sender]+msg[sender][i]*dx;
            }
            for(i=0; i<n; i++){ //normalization
               msg[sender][i]= msg[sender][i]/msgsum[sender];
               //sgl_color(c[sender]);
               //sgl_circle(dx*i, msg[sender][i]/fact, r2);    // these statements are to show the message functions
            }
         }
      }

//*********************************¥¥
¥¥*** Knowledge Modification ***//

      for(a=0; a<A; a++){
         if(a==receiver){
            for(i=0; i<n; i++){
```

```
            kmod[receiver][i]= k[receiver][i]+msg[sender][i]*ch;
          }
          for(i=0; i<n; i++){                    //normalization
            k[receiver][i]= kmod[receiver][i];
          }
          kmodsum[receiver]=0.;
          for(i=0; i<n; i++){
            kmodsum[receiver] = kmodsum[receiver]+k[receiver][i]*dx;
          }
          for(i=0; i<n; i++){
            k[receiver][i]= k[receiver][i]/kmodsum[receiver];
            sgl_color(c[receiver]);
            sgl_disc(dx*i, k[receiver][i]/fact, r2);
          }
        }
      }
      for(a=0; a<A; a++){
        for(i=0; i<n; i++){
          sgl_color(c[a]);
          sgl_disc(dx*i, k[a][i]/fact, r2);
        }
      }
      for (time=1; time>0; time--){   //putting intervals
        wait (1);
      }
      sgl_display();
    }                                        //End of Message Exchange loop

//*** Calculating Center of Knowledge and PeakValue of Modified knowledge function ***¥¥
¥¥*****************************************************************************************//

    for(a=0; a<A; a++){
      COKmod[a]=0;;
      for(i=0; i<n; i++){
        COKmod[a]=COKmod[a]+i*dx*k[a][i]*dx;
      }
      COKmod[a]=COKmod[a];
      printf("COKmod(%d turns)[%d]=%f ¥n", turn, a, COKmod[a]);
    }

    for(a=0; a<A; a++){
      PeakValueMod[a]=k[a][i]*dx;
```

```
      PeakValuePositionMod[a]=0.;
      for(i=0; i<n; i++){
        if (k[a][i]*dx >= PeakValueMod[a]){
          PeakValueMod[a]= k[a][i]*dx;
          PeakValuePositionMod[a]= i*dx;
        }
        else{
          PeakValueMod[a]= PeakValueMod[a];
          PeakValuePositionMod[a]= PeakValuePositionMod[a];
        }
      }
      printf("PVMod(%d turns)[%d]=%f ¥n", turn, a, PeakValueMod[a]);
}

   for(a=0; a<A; a++){
      printf("PVPMod(%d turns)[%d]=%f ¥n", turn, a, PeakValuePositionMod[a]);
   }
   printf("channel0 was selected [%d] times¥n", CHNL0CNT);
   printf("channel1 was selected [%d] times¥n", CHNL1CNT);
   printf("channel2 was selected [%d] times¥n", CHNL2CNT);
   printf("channel3 was selected [%d] times¥n", CHNL3CNT);
   printf("channel4 was selected [%d] times¥n", CHNL4CNT);
   printf("channel5 was selected [%d] times¥n", CHNL5CNT);
   printf("channel6 was selected [%d] times¥n", CHNL6CNT);
   printf("channel7 was selected [%d] times¥n", CHNL7CNT);
j=getchar();
} //end of main()
```