Why are some personalities less plastic?

A. Description of the model following the ODD protocol

1. Overview

1.1 Purpose

The purpose of the model is to predict when individual differences in consistent traits (i.e. fighting ability) should be related to differences in strategy use and behavioural plasticity. More precisely, individuals with different fighting abilities compete by pairwise interactions for resources by using either the fixed peaceful (Dove) or aggressive (Hawk) strategies or a plastic Assessor strategy and the model allows to determine the best strategy use for each fighting ability.

1.2 State variables and scales

The model comprises three hierarchical levels: individuals, resources and groups.
Individuals are characterized by their identity (i), fighting ability (α_i), strategy S[i] that is equal to 0, 1 or 2 if the individual plays Assessor, Dove or Hawk respectively, state that is equal to 0 or 1 if the individual is available or not, and the probability of making an assessment error (ε_i).
Pairs of individuals compete for resources that are characterized by their quality.
The groups are characterized by their size, the maximum difference between two contestant individuals in terms of fighting ability (that is determined by the spatial distribution of individuals according to their fighting ability), the cost of losing a fight and the cost of assessment.

1.3 Process overview and scheduling

Each simulation lasts T time steps, that each corresponds to a feeding event. Within each time step, 4 phases are proceeded in the following order: 1) individuals are partly randomly paired, 2) the expected gain of each individual is calculated based on its behaviour and fighting ability as well as that of its opponent, 3) for each fighting ability, the average gain of each strategy is estimated, 4) for each fighting ability, one randomly chosen individual that uses the least

successful strategy is replaced by another individual of the same fighting ability that uses the most successful strategy, unless all individuals already use the same strategy.

The duration of a simulation is set at 1000 time steps to ensure that the group has reached equilibrium frequencies, although for all combinations of parameters studied, the equilibrium is reached much more rapidly. As a group with a particular set of parameters does not necessarily converge towards the same equilibrium, because of the stochastic effects, each simulation is replicated 100 times.

2. Design concepts

Emergence: Individual differences in strategy use emerge as individuals that perform poorly are eliminated and replaced by individuals that adopt the most efficient strategy.

Fitness: The model assumes that natural selection eliminates the least performing individuals (i.e. individuals whose strategy provides the lowest average expected gain) that are replaced by individuals using the most effective strategy.

Prediction: There is no individual learning included in the decision process. Individuals, therefore, do not change their decision rules over time as a consequence of their experience. They cannot either predict future conditions.

Sensing: Individuals know their own fighting ability but ignore that of their opponents. Only individuals that use the Assessor strategy estimate their opponent's fighting ability and then decide to behave aggressively or non-aggressively, accordingly.

Stochasticity: Individuals are paired partly randomly, as the difference between two contestants cannot exceed q. This variable was introduced as individuals within a group may distribute spatially according to their phenotypic or behavioural traits, which may cause interactions to occur assortatively. The value of the resources for which pairs of contestants compete is also randomly chosen at the beginning of each time step among all possible values.

Collectives: The model assumes that the degree of assortativity between contestants may vary. More precisely the parameter q represents the maximum difference in fighting ability between two contestants. Thus, if q is equal to 5 (i.e. the highest fighting ability), individuals interact randomly, while the probability of assortative interactions increases as q decreases. As such, the model implicitly assumes that individuals may be spatially distributed in the environment according to their fighting ability.

Observation: For model testing, all information relative to each phase has been checked. Specifically, the output data included for each time step the identity, fighting ability and strategy of all pair members, their expected gain, the average gain of each strategy and the identity and strategy of the individual that was eliminated and replaced by a more performing individual. This was done to find and fix all potential mistakes that could have occurred at each step. Notably, I made numerous tests to be sure make sure that the code does not continue to run after giving some variables completely wrong values (i.e. run-time errors).
Also for time step 1 (individuals' pairing), I used the state variable (COMP[i]) that represents the number of potential opponents each individual can be paired with to ensure that interactions occurred partly randomly, each individual was paired only once and each pair of individuals had a difference in fighting ability than never exceeded q. The state variable opposant[i] (i.e. the identity of the opponent of individual i) was also reinitialized to 999 at the beginning of each time step, to verify that each individual had been paired to another individual.

For model analysis, the output data correspond to the number of individuals using the Dove, Hawk and Assessor strategies after T time steps and for each fighting ability as well as the average expected gain (Model B).

3. Details

3.1 Initialization

At time t=0, each individual is assigned a fighting ability and a strategy.

3.2 Inputs

The environment is assumed to be constant, so the model has no input data.

3.3 Submodels

Individuals' pairing

At the beginning of each time step, all individuals are considered available (i.e. state[*i*]=0). The model considers successively all available individuals and randomly chooses one opponent for each of them. To do that, it draws a number between 0 and G and verifies that the opponent (whose identity corresponds to this random number) is acceptable (in terms of availability and fighting ability) or repeats the procedure until finding an acceptable opponent. Once paired, both contestants become unavailable (state[*i*]=1) for other individuals and for both of them the state variable opponent[i] takes as value the identity of their opponent.

Calculation of individuals' expected gain

For each pair, the model estimates the average payoff of both constants (gain[*i*]). As individuals can adopt three different strategies, there are 9 possible combinations of strategies. The equations used to calculate the payoffs are those given in Table 2, expect that the probability of making an assessment error (ε) is not necessarily fixed but may vary among individuals depending on their fighting ability (see lines 129-134 in the article).

Calculation of average expected gains

To determine which strategy provides the highest and lowest payoff, the model calculates, for each fighting ability, the average expected payoff of individuals that play Dove, Assessor and Hawk. The variables gain_D[*i*], gain_A[*i*] and gain_H[*i*], that correspond to the average gains of individuals of fighting ability *i* playing Dove, Assessor or Hawk respectively, are first incremented by the payoffs of individuals, and then divided by the number of individuals playing the different strategies D[*i*], A[*i*] and H[*i*].

Replacement of one randomly chosen individual

For each fighting ability, the model seeks the strategy that provides individuals with the lowest and highest average payoffs, then randomly chooses one individual that uses the least effective strategy and changes its strategy for the most successful one.

This step is performed only if individuals of a given fighting ability still use at least two different strategies.

B. Codes of the two models

Model A. Less successful individuals are replaced by individuals with similar fighting ability

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<time.h>

const int n=20; /*initial number of individuals with a particular fighting ability and a particular strategy */
const int nb_strategies=3; /*Dove (0), Assessor (1), Hawk (2)*/
const int nb_classes=6; /*number of fighting abilities*/

const int Vmoy=10;
const int ecart=0; /*deviation around the mean*/
const double C=10;/*cost of losing a fight*/
const int q=5; /*maximal difference in fighting ability between two contestants*/
const int T=1000; /*number of time steps per simulation*/
const double erreur_min=0;
const double erreur_dif=0;
const int nb_simul=100;
const double cost_a=2;

int G;
int V;
int Vmax, Vmin;
 int strategie[1000];
int capacite[1000];
double gain[1000];
double p[1000];
int opposant[1000];
int i,t,j,k,y,s;
int state[1000];
COMP[1000]; /*Number of potential opponents each individual can be paired with*/
int CHOIX,VA,X;
int Low[100], High[100];
int proba;
double erreur[1000];
int D[20],A[20],H[20];
double gain_D[100],gain_A[100],gain_H[100];
double moy_gain_D[100],moy_gain_A[100],moy_gain_H[100];
int Cumul_D[100],Cumul_A[100],Cumul_H[100];


void main (void){
```

```
	srand((unsigned)time(NULL));

	G=nb_classes*nb_strategies*n;
	X=n*nb_strategies;

/*The following variables correspond to the sum of the number of individuals whose fighting ability is j and that use the strategies Dove, Assessor or Hawk after s simulations*/
	for (j=0;j<nb_classes;j++){
		Cumul_D[j]=0;
		Cumul_A[j]=0;
		Cumul_H[j]=0;
	}

	for (s=0;s<nb_simul;s++){
		printf("\nSimulation %d",s);

y=0;
/*At the beginning of each simulation, each individual y is assigned a strategy and a fighting ability that determines its probability of making an assessment error*/
for (i=0;i<nb_classes; i++){
	for (j=0;j<nb_strategies;j++){
		for (k=0;k<n;k++){
			capacite[y]=i;
			strategie[y]=j;
			erreur[y]=erreur_min+i*erreur_dif;
			if(erreur[y]>0.5)erreur[y]=0.5;
			y+=1;
		}
	}
}


/*At the beginning of each time step the value of the contested resource (V) is randomly chosen among all possible values raging from (Vmoy-ecart) and (Vmoy+ecart) */
for(t=0;t<T;t++){
	if(ecart==0)V=Vmoy;
	else{
	Vmax=Vmoy+ecart;
	Vmin=Vmoy-ecart;
	/*printf("\nTemps %d",t);*/
	V=rand()%(Vmax-Vmin);
	V=V+Vmin;
	}
```

```
/*These variables correspond to the number of individuals whose fighting ability is j and that play Dove, Assessor and Hawk. They are reinitialized to zero at the beginning of each time step*/
for (j=0;j<nb_classes;j++){
        D[j]=0;
        A[j]=0;
        H[j]=0;
}

/*For each fighting ability the model counts the number of individuals that play Dove, Assessor and Hawk*/
for (i=0;i<G;i++){
        if(strategie[i]==0) D[capacite[i]]+=1;
        else if (strategie[i]==1) A[capacite[i]]+=1;
        else H[capacite[i]]+=1;
}


/*Each individual is characterized by its state (available : 0, or unvailable :1), and its gain. These two state variables are reinitialized to zero at the beginning of each time step.*/
for (i=0;i<G;i++){
                state[i]=0;
                gain[i]=0;
                COMP[i]=0;
                opposant[i]=999;
        }

/*CHOICE OF AN OPPONENT : for each individual the model chooses partly randomly an opponent. Once an individual has been paired, it is no more available (and so its state variable becomes equal to 1)*/

for(i=0;i<G;i++){
                if(state[i]==0){
                for (j=i+1;j<G;j++){
if ((state[j]==0)&&(capacite[j]>(capacite[i]-q)) &&
(capacite[j]<(capacite[i]+q)))COMP[i]+=1;
                }

                if((COMP[i]>0)){
                do CHOIX=rand()%G;
                while
((state[CHOIX]==1)||(capacite[CHOIX]>(capacite[i]+q))||(capacite[CHOIX]<(capacite[i]-
q)));
                opposant[i]=CHOIX;
                opposant[CHOIX]=i;
                state[i]=1;
```

```
            state[CHOIX]=1;
        }

        else i++;
        }
        }
```

/*GAIN OF EACH PLAYER : for each individual, the model estimates its expected payoff that depends on its strategy and that of its opponent, as well as one their relative fighting ability and risk of assessment error*/

```
        for (i=0;i<G;i++){
            if(opposant[i]!=999){

        /*Case 1: HH Both individuals play Hawk*/
            if((strategie[i]==2)&&(strategie[opposant[i]]==2)){
                if(capacite[i]>capacite[opposant[i]]) gain[i]=V;
                else if (capacite[i]<capacite[opposant[i]]) gain[i]=-C;
                else gain[i]=(V-C)/2;
            }
        /*Case 2:HD Individual i plays Hawk and its opponent plays Dove*/
            if((strategie[i]==2)&&(strategie[opposant[i]]==0)){
                gain[i]=V;
            }

        /*Case3: HA Individual i plays Hawk and its opponent plays Assessor */
            if((strategie[i]==2)&&(strategie[opposant[i]]==1)){
                if(capacite[i]>capacite[opposant[i]]) gain[i]=V-cost_a;
                else if(capacite[i]<capacite[opposant[i]])
gain[i]=erreur[opposant[i]]*V-(1-erreur[opposant[i]])*C-cost_a;
                else gain[i]=(1-erreur[opposant[i]])*(V-C)/2+erreur[opposant[i]]*V-
cost_a;
            }

        /*case 4:DH Individual i plays Dove and its opponent plays Hawk */
            if((strategie[i]==0)&&(strategie[opposant[i]]==2)){
                gain[i]=0;
            }

        /*Case 5:DD Both individuals play Dove */
            if((strategie[i]==0)&&(strategie[opposant[i]]==0)){
                gain[i]=V/2;
            }

        /*case6:DA Individual i plays Dove and its opponent plays Assessor*/
```

```
		if((strategie[i]==0)&&(strategie[opposant[i]]==1)){
			gain[i]=0;
		}


	/*case7:AH Individual i plays Assessor and its opponent plays Hawk */
		if((strategie[i]==1)&&(strategie[opposant[i]]==2)){
			if(capacite[i]>capacite[opposant[i]]) gain[i]=(1-erreur[i])*V-cost_a;
			else if (capacite[i]<capacite[opposant[i]]) gain[i]=0-erreur[i]*C-cost_a;
			else gain[i]=(1-erreur[i])*(V/2-C/2)-cost_a;
		}

	/*case8:AD Individual i plays Assessor and its opponent plays Dove */
		if((strategie[i]==1)&&(strategie[opposant[i]]==0)){
			gain[i]=V-cost_a;
		}

	/*case9:AA Both individuals play Assessor */
		if((strategie[i]==1)&&(strategie[opposant[i]]==1)){
			if (capacite[i]>capacite[opposant[i]]) gain[i]=(1-
erreur[i])*V+erreur[i]*(1-erreur[opposant[i]])*(V/2)-cost_a;
			else if (capacite[i]<capacite[opposant[i]]) gain[i]=(1-
erreur[i])*erreur[opposant[i]]*V/2+erreur[i]*(erreur[opposant[i]]*V-(1-
erreur[opposant[i]])*C)-cost_a;
			else gain[i]=(1-erreur[i])*((1-erreur[opposant[i]])*(V-
C)/2+erreur[opposant[i]]*V)+erreur[i]*erreur[opposant[i]]*V/2-cost_a;
		}
	}
	}

/*For each fighting ability, the model estimates the average payoff associated with each
strategy */
	for (i=0;i<nb_classes;i++){
		gain_A[i]=0;
		gain_D[i]=0;
		gain_H[i]=0;
		Low[i]=9;
		High[i]=9;
	}

	for(i=0;i<G;i++){
		if(strategie[i]==0) gain_D[capacite[i]]+=gain[i];
		else if (strategie[i]==1) gain_A[capacite[i]]+=gain[i];
		else if(strategie[i]==2) gain_H[capacite[i]]+=gain[i];
		}
```

```
	for (i=0;i<nb_classes;i++){
		if (D[i]>0) moy_gain_D[i]=gain_D[i]/D[i];
		if(A[i]>0) moy_gain_A[i]=gain_A[i]/A[i];
		if(H[i]>0)moy_gain_H[i]=gain_H[i]/H[i];
	}

/*For each fighting ability, the model seeks the strategies that provide individuals with the lowest and highest average payoff */

	for(i=0;i<nb_classes;i++){
		/*printf("\nC:%d, gain_D:%lf, gain_A:%lf, gain_H:%lf",i,gain_D[i],gain_A[i],gain_H[i]);*/


		/*case 1: The 3 strategies are still present*/
		if((D[i]>0)&&(A[i]>0)&&(H[i]>0)){

	if((moy_gain_H[i]<=moy_gain_D[i])&&(moy_gain_H[i]<=moy_gain_A[i])) Low[i]=2;
			else if ((moy_gain_A[i]<=moy_gain_H[i])&&(moy_gain_A[i]<=moy_gain_D[i])) Low[i]=1;
			else Low[i]=0;


	if((moy_gain_D[i]>=moy_gain_H[i])&&(moy_gain_D[i]>=moy_gain_A[i])) High[i]=0;
			else if ((moy_gain_A[i]>=moy_gain_H[i])&&(moy_gain_A[i]>=moy_gain_D[i])) High[i]=1;
			else High[i]=2;
		}


		/*case 2: Only H and D are still present*/
		if((D[i]>0)&&(A[i]==0)&&(H[i]>0)){
			if(moy_gain_H[i]<=moy_gain_D[i]) {
				Low[i]=2;
				High[i]=0;
			}

			else {
				Low[i]=0;
				High[i]=2;
			}
		}


		/*case 3: Only H and A are still present*/
		if((D[i]==0)&&(A[i]>0)&&(H[i]>0)){
```

```
                if(moy_gain_H[i]<=moy_gain_A[i]) {
                        Low[i]=2;
                        High[i]=1;
                }
                else {
                        Low[i]=1;
                        High [i]=2;
                }
        }


        /*case 4: Only D and A are still present*/
        if((D[i]>0)&&(A[i]>0)&&(H[i]==0)){
                if(moy_gain_A[i]<=moy_gain_D[i]) {
                        Low[i]=1;
                        High[i]=0;
                }
                else {
                        Low[i]=0;
                        High[i]=1;
                }
        }
}

/*For each fighting ability, the model randomly chooses one individual that uses the least efficient strategy and changes its strategy for the most efficient one*/
        for(i=0;i<nb_classes;i++){
                if((Low[i]!=9)&&(High[i]!=9)){
                        do {
                                VA=rand()%X+i*X;
                        }
                        while (strategie[VA]!=Low[i]);
                        strategie[VA]=High[i];
                }
                }

}


/*Simulation outputs: results for the simulation s*/
for(i=0;i<nb_classes;i++){
                printf("\nFighting ability: %d, D:%d, A:%d, H:%d",i,D[i],A[i],H[i]);
                Cumul_D[i]+=D[i];
                Cumul_A[i]+=A[i];
                Cumul_H[i]+=H[i];
        }
```

```
}

/*Simulation outputs: results for all S simulations*/
printf("\nResults after %d Simulations", s);
for(i=0;i<nb_classes;i++){
        printf("\nFighting ability: %d, D:%d, A:%d,
H:%d",i,Cumul_D[i],Cumul_A[i],Cumul_H[i]);
}

}
```

Model B. Less successful individuals are replaced by individuals with the highest payoff regardless of their competitive ability

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<time.h>

const int n=20;
const int nb_strategies=3;
const int nb_classes=6;

const int Vmoy=10;
const int ecart=10;
const double C=20;
const int T=1000;
const int q=5;
 const double erreur_min=0;
const double erreur_dif=0;
const int nb_simul=100;
const double cost_a=1;

int G;
int V,Vmax,Vmin;
int strategie[1000];
int capacite[1000];
double gain[1000];
int opposant[1000];
double erreur[1000];
int i,t,j,k,y,s;
int state[1000];
int CHOIX,VA,X;
int Low, High;
double Min,Max;
double gain_min, gain_max;
int capacite_min, capacite_max;
int proba;
int D[100],A[100],H[100];
int COMP[1000];
int Sum_D[20],Sum_A[20],Sum_H[20];
double gain_D[100],gain_A[100],gain_H[100];
double moy_gain_D[100],moy_gain_A[100],moy_gain_H[100];
int Cumul_D[100],Cumul_A[100],Cumul_H[100];
double gain_moy;
double gain_moy_all_simul;
```

```
void main (void){
	srand((unsigned)time(NULL));

	G=nb_classes*nb_strategies*n;
	X=n*nb_strategies;
	gain_moy_all_simul=0;

	for (j=0;j<nb_classes;j++){
		Cumul_D[j]=0;
		Cumul_A[j]=0;
		Cumul_H[j]=0;
	}

	for (s=0;s<nb_simul;s++){
		printf("\nSimulation %d",s);
		gain_moy=0;

y=0;
for (i=0;i<nb_classes; i++){
	for (j=0;j<nb_strategies;j++){
		for (k=0;k<n;k++){
			capacite[y]=i;
			strategie[y]=j;
			erreur[y]=erreur_min+(i*erreur_dif);
			if(erreur[y]>0.5)erreur[y]=0.5;
			y+=1;
		}
	}
}

/*The value of the contested resource is chosen randomly among all possible values*/
for(t=0;t<T;t++){
	Vmax=Vmoy+ecart;
	Vmin=Vmoy-ecart;
	if(ecart==0)V=Vmoy;
	else{
		V=rand()%(Vmax-Vmin);
		V+=Vmin;
	}

for (j=0;j<nb_classes;j++){
	D[j]=0;
	A[j]=0;
	H[j]=0;
}
```

```
for (j=0;j<nb_classes;j++){
	Sum_D[j]=0;
	Sum_A[j]=0;
	Sum_H[j]=0;
}


for (i=0;i<G;i++){
	if(strategie[i]==0) D[capacite[i]]+=1;
	if (strategie[i]==1) A[capacite[i]]+=1;
	if (strategie[i]==2) H[capacite[i]]+=1;
}


/*Each individual is characterized by its state (available : 0, or unvailable :1), and its gain.
These two state variables are reinitialized to zero at the beginning of each time step*/
for (i=0;i<G;i++){
		state[i]=0;
		gain[i]=0;
		COMP[i]=0;
		opposant[i]=999;
	}

	for(i=0;i<G;i++){
	if(state[i]==0){
		for (j=i+1;j<G;j++){
			if ((state[j]==0)&&(capacite[j]>(capacite[i]-
q))&&(capacite[j]<(capacite[i]+q))) COMP[i]+=1;
		}
		if(COMP[i]>0){
		do CHOIX=rand()%G;
		while
((state[CHOIX]==1)||(capacite[CHOIX]>(capacite[i]+q))||(capacite[CHOIX]<(capacite[i]-
q)));
		opposant[i]=CHOIX;
		opposant[CHOIX]=i;
		state[i]=1;
		state[CHOIX]=1;
	}

	else  i++;
	}
	}

/*Calculation of the expected gain of each individual*/
```

```
	for (i=0;i<G;i++){
		if(opposant[i]!=999){


	/*Case 1: HH*/
		if((strategie[i]==2)&&(strategie[opposant[i]]==2)){
			if(capacite[i]>capacite[opposant[i]]) gain[i]=V;
			else if (capacite[i]<capacite[opposant[i]]) gain[i]=-C;
			else gain[i]=(V-C)/2;
		}
	/*Case 2:HD*/
		if((strategie[i]==2)&&(strategie[opposant[i]]==0)){
			gain[i]=V;
		}

	/*Case 3: HA*/
		if((strategie[i]==2)&&(strategie[opposant[i]]==1)){
			if(capacite[i]>capacite[opposant[i]]) gain[i]=V;
			else if(capacite[i]<capacite[opposant[i]])
gain[i]=erreur[opposant[i]]*V-(1-erreur[opposant[i]])*C;
			else gain[i]=(1-erreur[opposant[i]])*(V-C)/2+erreur[opposant[i]]*V;
		}

	/*case 4:DH*/
		if((strategie[i]==0)&&(strategie[opposant[i]]==2)){
			gain[i]=0;
		}

	/*Case 5:DD*/
		if((strategie[i]==0)&&(strategie[opposant[i]]==0)){
			gain[i]=V/2;
		}

	/*case 6:DA*/
		if((strategie[i]==0)&&(strategie[opposant[i]]==1)){
			gain[i]=0;
		}


	/*case 7:AH*/
		if((strategie[i]==1)&&(strategie[opposant[i]]==2)){
			if(capacite[i]>capacite[opposant[i]]) gain[i]=(1-erreur[i])*V-cost_a;
			else if (capacite[i]<capacite[opposant[i]]) gain[i]=0-erreur[i]*C-cost_a;
			else gain[i]=(1-erreur[i])*(V/2-C/2)-cost_a;
		}
```

```
	/*case 8:AD*/
		if((strategie[i]==1)&&(strategie[opposant[i]]==0)){
			gain[i]=V-cost_a;
		}

	/*case 9:AA*/
		if((strategie[i]==1)&&(strategie[opposant[i]]==1)){
			if (capacite[i]>capacite[opposant[i]]) gain[i]=(1-
erreur[i])*V+erreur[i]*(1-erreur[opposant[i]])*(V/2)-cost_a;
			else if (capacite[i]<capacite[opposant[i]]) gain[i]=(1-
erreur[i])*erreur[opposant[i]]*V/2+erreur[i]*(erreur[opposant[i]]*V-(1-
erreur[opposant[i]])*C)-cost_a;
			else gain[i]=(1-erreur[i])*((1-erreur[opposant[i]])*(V-
C)/2+erreur[opposant[i]]*V)+erreur[i]*erreur[opposant[i]]*V/2-cost_a;
		}
		gain_moy+=gain[i];

	}

	}

		gain_moy=gain_moy/G;
		if(t==(T-1)){
			gain_moy_all_simul+=gain_moy;
			printf("  Gain moy: %lf Gain all sim:%lf",gain_moy,
gain_moy_all_simul);
		}

	for (i=0;i<nb_classes;i++){
		gain_A[i]=0;
		gain_D[i]=0;
		gain_H[i]=0;
	}
		Low=9;
		High=9;
		Min=100;
		Max=-100;


	for(i=0;i<G;i++){
		if(strategie[i]==0) gain_D[capacite[i]]+=gain[i];
		else if (strategie[i]==1) gain_A[capacite[i]]+=gain[i];
		else if(strategie[i]==2) gain_H[capacite[i]]+=gain[i];
		}
```

```
	for (i=0;i<nb_classes;i++){
		if (D[i]>0) moy_gain_D[i]=gain_D[i]/D[i];
		if(A[i]>0) moy_gain_A[i]=gain_A[i]/A[i];
		if(H[i]>0)moy_gain_H[i]=gain_H[i]/H[i];
	}
```

/*The model seeks the strategy and fighting ability that are associated with the lowest expected gain. They are represented by the variables Low and capacite_min, respectively. More precisely the model considers sequentially the Dove, Assessor and Hawk players and compare the gain of each individual to a predefined minimum value (Min) which was set to 100: if the gain of the individual is lower than Min, “Min”, “Capacite_min” and “Low” take as values the gain, fighting ability and strategy of that individual*/

```
for (i=0;i<nb_classes;i++){
	if(D[i]>0){
		if(gain_D[i]<Min){
			Min=gain_D[i];
			Low=0;
			capacite_min=i;
		}
	}

	if(H[i]>0){
		if(gain_H[i]<Min){
			Min=gain_H[i];
			Low=2;
			capacite_min=i;
		}
	}

		if(A[i]>0){
		if(gain_A[i]<Min){
			Min=gain_A[i];
			Low=1;
			capacite_min=i;
		}
	}
}
```

/*The model seeks the strategy and fighting ability that are associated with the highest expected gain. They are represented by the variables High and capacite_max, respectively. More precisely it considers sequentially the Dove, Assessor and Hawk players and compare the gain of each individual to a predefined maximal value (Max) which was set to -100: if

```
the gain of the individual is larger than Max, “Max”, “Capacite_max” and “high”take as values
the gain, fighting ability and strategy of that individual*/
for (i=0;i<nb_classes;i++){
        if(D[i]>0){
                if(gain_D[i]>Max){
                        Max=gain_D[i];
                        High=0;
                        capacite_max=i;
                }
        }


        if(H[i]>0){
                if(gain_H[i]>Max){
                        Max=gain_H[i];
                        High=2;
                        capacite_max=i;
                }
        }

        if(A[i]>0){
                if(gain_A[i]>Max){
                        Max=gain_A[i];
                        High=1;
                        capacite_max=i;
                }
        }
}

/*The model randomly chooses one individual from the least successful type (i.e. that uses
the strategy Low and has a fighting ability equal to capacite_min) and then changes its
strategy and fighting ability to High and capacite_max respectively)*/
                if((Low!=9)&&(High!=9)){
                        do {
                                VA=rand()%G;
                        }
                        while ((strategie[VA]!=Low)||(capacite[VA]!=capacite_min));
                        strategie[VA]=High;
                        capacite[VA]=capacite_max;
                }

}

for (i=0;i<G;i++){

        if(strategie[i]==0)Sum_D[capacite[i]]=D[capacite[i]];
```

```
        else if (strategie[i]==1) Sum_A[capacite[i]]=A[capacite[i]];
        else Sum_H[capacite[i]]=H[capacite[i]];

}


        for(i=0;i<nb_classes;i++){
                Cumul_D[i]+=Sum_D[i];
                Cumul_A[i]+=Sum_A[i];
                Cumul_H[i]+=Sum_H[i];
        }

}

        printf("\n\nSimulation outputs for %d simulations", s);
for(i=0;i<nb_classes;i++){
        printf("\nFighting ability %d, D:%d, A:%d,
H:%d",i,Cumul_D[i],Cumul_A[i],Cumul_H[i]);
}
printf("\n Average gain : %lf", gain_moy_all_simul/s);


}
```