ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lecture with Computer Exercises: Modelling and Simulating Social Systems with MATLAB

Project Report

DINER‘S DILEMMA

Christian Graf, Benita Heiz

Abstract: The Diner‘s Dilemma is a situation in a restaurant. N-persons eat together and split the bill afterwards. The 2-player Diner‘s Dilemma corresponds to the Prisoner dilemma. Without memory the Nash equilibrium is the expensive meal. In a one time game it‘s the best strategy to eat the expensive meal and not to care about the others. Eating always the expensive meal is the strategy ALLD. (ALLD mean payoff= 3.082 ; T= 10‘000, mutation= 0) The strategies with memory changes the benefit of the ALLD-strategy. ALLD becomes the worst strategy. (ALLD mean payoff= 2.04; T= 10‘000, mutation= 0, no memory) The best strategy is TFT together with four other strategies. (TFT mean payoff = 4.00 ; T= 10‘000, mutation= 0, memory) Also in the 3-player algorithm the strategies with memory are better than the constant strategies too.
Further on we tested the strategies with mutations. The strategies with memory accumulated the mutation, untill they stabilized on a specific value.

ETH Zurich, 2010

Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich diese Gruppenarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen-Hilsmittel verwenden habe, und alle Stellen, die wörtlich oder sinngemäss aus veröffentlichen Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Gruppenarbeit nicht, auch nicht auszugsweise, bereits für andere Prüfung ausgefertigt wurde.

Christian Graf & Benita Heiz

Contents

1) Introduction

The **Diner‘s dilemma** is an N-player Prisoner dilemma. It‘s a problem in the game theory. N-individuals go out to eat. Before diner, they appoint to split the bill equally between all N-players. Each individual wants his own benefit. There‘s a cheap and an expensive meal. The expensive meal tastes better than the cheap one. But the price difference between the meals isn‘t worth the taste benefit. Every player has to consider, if it is worth taking a more expensive meal, because of splitting the check. [1]
The added price to the own bill by ordering the more expensive meal is very small. The reasonable choice for an individual is to take the more expensive meal. But if everybody takes the more expensive meal, then each person has to pay more than the meal is worth.

First we made a 2 player Diner‘s dilemma with no memory. (1.1) According to some studies of social behaviour, a human doesn‘t only consider the own profit.
An individual is normally touched of others benefit. Many don‘t go for the maximal gain. To elaborate this theory, we considered the memory. (1.2) Further on we checked the dilemma with 3 players. (1.3)

The research questions are

I. Which strategy is the best to gain the most?
II. What changes, if you consider a memory?
III. How does the mutation rate influence the payoff?
IV. How does the time of iterations influence the payoff?
V. What changes with a 3-player “Diner‘s Dilemma“?
VI. How do our results correlate with the references?
VI. What are the strengths and the weaknesses of our models?

1.1) Description of the model with no memory; N= 2

Situation: There‘s no memory. We consider the following matrix as the payoff matrix.

own choice: C -> cooperate= cheap D -> defect= expensive
opponent choice: C -> cooperate= cheap D -> defect= expensive

	C	D
C	2	0
D	5	1

The propability of defection and cooperation is fixed.

We consider four different strategies:
ALLC = always cooperation
ALLD = always defection
probability of cooperation is 1/3
probability of cooperation is 2/3

1.2) Description of the model with memory. Consideration of previous move with probabilities that are fixed. N = 2

The purpose of memory consideration is to find an algorithm, that is closer to reality. People often react according to previous experiences.
We defined probabilities of cooperation based on the previous move according to the p versus q representa-

tion. p stands for the probability of cooperation, if the other guy coorporated in the previous round. q is the probability of cooperation, if the other guy defected in the previous round.

1: ALLC.
2: ALLD.
3: TFT
4: p= 1, q= 1/3
5: p= 1, q= 2/3
6: player takes the opponent‘s previous decision.
7: player takes the opponent‘s previous decision, if it was more or equally successful, otherwise cooperation.
8: player takes the opponent‘s previous decision, if it was strictly more successful, otherwise cooperation.
9: player takes the opponent‘s previous decision, if it was strictly more successful, otherwise its previous decision.
10: player takes the opponent‘s previous decision, if it was more or equally successful, otherwise its previous decision.
11: player takes the opponent‘s previous decision, if it was more or equally successful, otherwise defection.
12: player takes the opponent‘s previous decision, if it was strictly more successful, otherwise defection.

1.3) Description of the method with 3 players

To get closer to reality we elaborated a case with 3 players. We achieved the following matrix as the payoff matrix according to 2.1:

a(1,1,1)=1;
a(1,2,1)=-1;
a(2,1,1)=3;
a(2,2,1)=1;
a(1,1,2)=-1;
a(1,2,2)=-4;
a(2,1,2)=1;
a(2,2,2)=-1;

1: ALLC
2: ALLD
Imitation1: If both opponent players cooperated in the previous interaction, the current player cooperates too. Otherwise he defects. [2]
Imitation2: If both opponent players defected in the previous interaction, the current player defects too. Otherwise he cooperates.[2]
Imitation3: If one player defected and the other player cooperated in the previous round, the current player cooperate. Otherwise the current player takes the previous move of the opponent player, which got the higher payoff. [2]
Reinforcement: The current player takes the previous move of the most successful player in the last round. [2]
Best reply: The current player chooses the strategy, with whom he would have received the highest payoff in the last round. [2]

2) Materials & Methods

2.1) Parameters

The **strategy template** defined the size of the payoff-matrix and the first steps. DD is the matrix characteristical for a 2-player "Diner's Dilemma".

g represents the joy of eating the expensive meal, b the joy of eating the cheap meal, h is the cost of the expensive, l the cost of the cheap meal, If $h > g > b > l$ and $g > (h+b)/2$ This conditions should be fulfilled for a classical "Diner's Dilemma". That's why we had to change our matrix. [1]
[2 0; 5 1] -> [5 0; 6 1]

According to this, an individual preferes having the expensive meal, when the others defray the payment. The expensive meal is strictly dominant and forms a Nash Equlibrium. [4]

h (= 9) > g (= 8) > b (= 4) > l (= 3)

	cooperate	defect
c	g-h	b-0.5(l +h)
d	g-0.5 (l+h)	b-l

2.2) loops

We worked with "if...(else)" loops.

```
if(rand<=1/3)
        CurrPlayerCurrentMove = 1;
    else CurrPlayerCurrentMove = 2;
```

2.3) strategy interactions

I)
```
for i = 1:NoOfStrategy
    for j = i:NoOfStrategy
```

II)
```
for i = 1:NoOfStrategy
    for j = 1:NoOfStrategy
```

resp. (3-player)
```
for i = 1:NoOfStrategy
    for j = 1:NoOfStrategy
        for h= 1:NoOfStrategy
```

More calculation steps are necessary but it's easier to display the results. (II)

2.4) Graphical representation

"mod" is used to save **every 100th mean payoff** (up to this time) for the time period 0 -> T

```
 for t = 1:T
          k = mod(t,100);
          t2 = floor( t/100 );
```

Plotting results

T2: each 100th timestep

meanpayoff2: meanpayoff up to each 100th timestep and summed over all opponent strategy and therefore divided by number of strategies.

```
for n = 1:NoOfStrategy
   figure;
    plot(T2,meanpayoff2(n,T2)‘/NoOfStrategy);
```

resp. (3-players)

```
 for n = 1:NoOfStrategy
    figure;
     plot(T2,meanpayoff2(n,T2)‘/NoOfStrategy/ NoOfStrategy);
```

Displaying the result

```
disp(meanpayoff/NoOfStrategy)
```

resp. (3-players)

```
disp(meanpayoff/NoOfStrategy/NoOfStrategy)
```

3) Results & Discussion

3.2) Method without memory and 2 players

	meanpayoff
ALLC	0.834
ALLD	2.667
prob. of coop. 1/3	2.057
prob. of coop. 2/3	2.055

Table 1: Mean payoff for constant strategies; DD: [2 0; 5 1]; T= 10‘000, mutation= 0

	meanpayoff
ALLC	2.087
ALLD	3.082
prob. of coop. 1/3	2.748
prob. of coop. 2/3	2.749

Table 2: Mean payoff for constant strategies; DD: [6 0; 5 1]; T= 10‘000, mutation= 0

The algorithm shows the benefit of one strategy according to an other strategy. There are 10000 interactions. The distribution of cooperation is equally in both algorithms. The first payoff matrix (DD: [2 0; 5 1]) shows that the defection strategy is by far the best and the cooperation strategy is by far the worst. In the second payoff matrix (DD: [6 0; 5 1]) the defection strategy is still the best, but the mean payoff differences between the strategies aren‘t as large as before.
It‘s interesting that, without memory, ALLD is a very good strategy. Defection builds the Nash equilibrium in a one round game, defection is always the best choice to gain the most.
The mutation rate of 0.001 doesn‘t change the payoff matrix significantly, since the relative difference is smaller than 1 %. With a mutation rate of 0.1, you receive a change in the mean payoff of maximal 5%.
If we change the time steps from 10000 to 100000, the mean payoff doesn‘t change much. The relative difference is less than 1% and thus negligible.To display the mean payoff versus time doesn‘t make sense, because there‘s only a small fluctuation due to statistics as we have pointed out in the previous sentence.

3.2) Method with memory and 2 players

	meanpayoff
ALLC	3.75
ALLD	2.04
TFT	4.00
4	3.92
5	3.83
6	4.00
7	4.00
8	3.88
9	4.00
10	4.00
11	2.04
12	2.04

With memory & 12 strategies (Table 3)
The strategy with the best payoff is TFT together with four other strategies (TFT, 6,7, 9 and 10). [5] The worst strategies are those who prefer defecting (ALLD, 11 and 12). ALLD strategy is very bad compared to the method without memory. That's because other strategies punish this player. The strategy's payoff depends much on the other players strategies.
We would have expected, that TFT would lead to the highest mean payoff in the algorithm with memory. It's indeed a very good strategy, but doesn't have no rival. [5]

Table 3: Mean payoff for strategies with memory; DD: [6 0; 5 1], T= 100'000; mutation rate= 0

	meanpay-off
ALLC	3.89
ALLD	3.39
TFT	4.11
4	4.04
5	3.96
6	4.11
8	4.00
9	4.11
12	2.39

With memory & 9 strategies (Table 4)
It doesn't make sense to consider all strategies. The strategies 6, 7, 9, 10 and, 11, 12 lead to more or less the same results. That's why we cancelled strategy 7, 10 and 11. Without this three strategies the mean payoff changes slightly, the worst strategy is now 12. That means it's bad to copy the opponents previous decision, if it was strictly more successfull and otherwise defects.

Table 4: Mean payoff for strategies with memory without strategy 7, 10,11; DD: [6 0; 5 1], T= 100'000; mutation rate= 0

	meanpay-off
ALLC	3.52
ALLD	2.39
TFT	3.09
4	3.60
5	3.58
6	3.08
8	3.23
9	2.43
12	2.39

With memory & 9 strategies & mutation (Table 5)

TFT isn't anymore a very good strategy. Mutations accumulated. (fig. 1)
Strategy 9, takes the opponents previous decision, if it was strictly more successful, otherwise its previous decision. Strategy 9 is the worst strategy, because of the mutation accumulation. The forth strategy is the best one. Relatively to the other strategies the percentage of cooperation is high. Cooperation probably tends to yield a high payoff.

Table 5: Mean payoff for strategies with memory without strategies 7, 10,11; DD: [6 0; 5 1], T= 100'000; mutation rate= 0.001

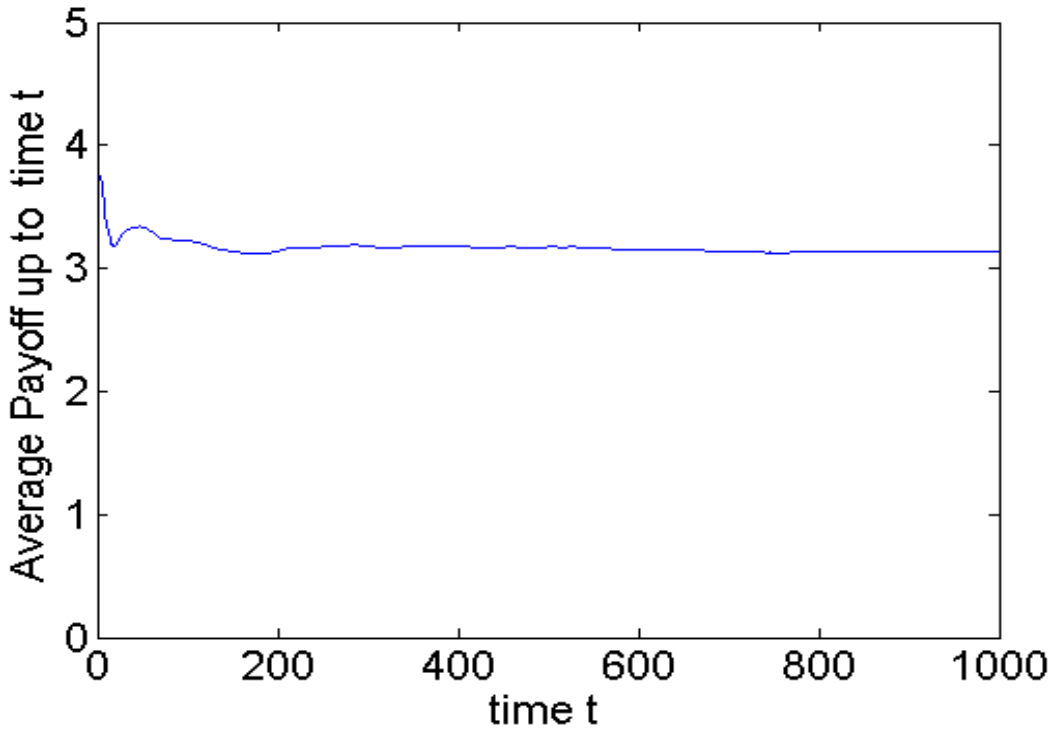

fig. 1: The payoff of TFT diminishes roughly after the simulation initiation. With time the mean payoff remains constant at a low level; TFT, mutation= 0.001, T= 100‘000

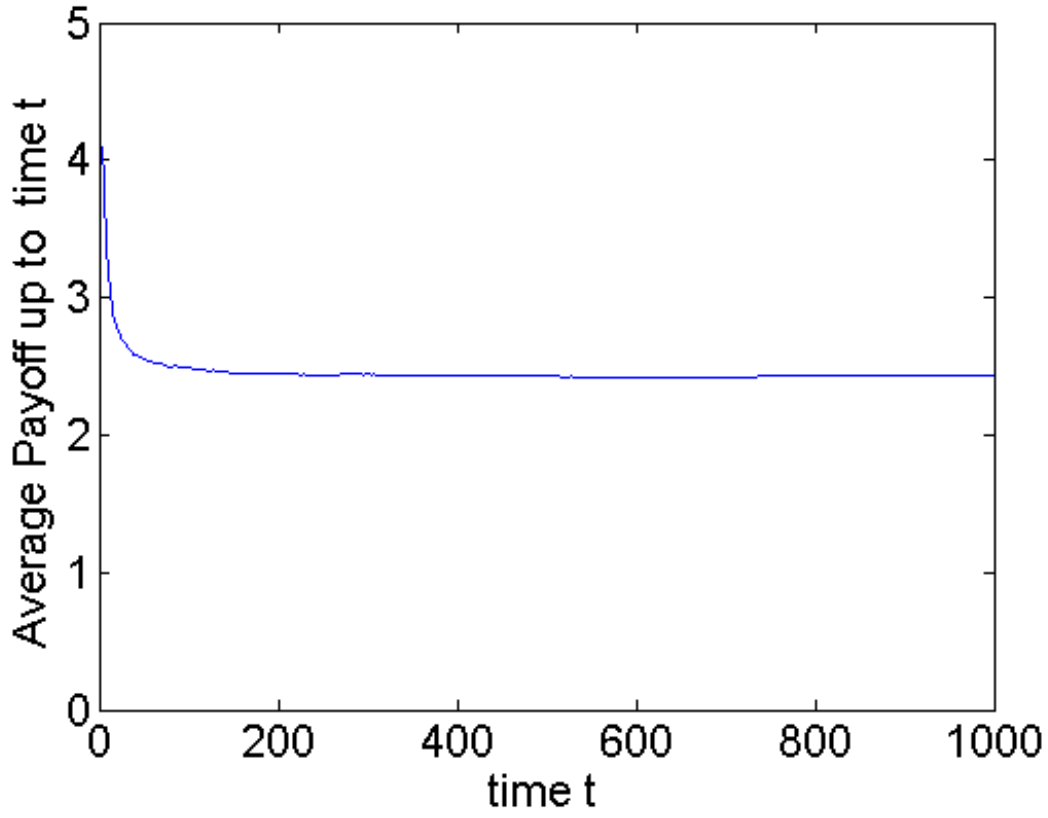

fig. 2: The mutations accumulate much, untill the mean payoff stabilizes; Strategy 9, mutation= 0.001, T= 100‘000

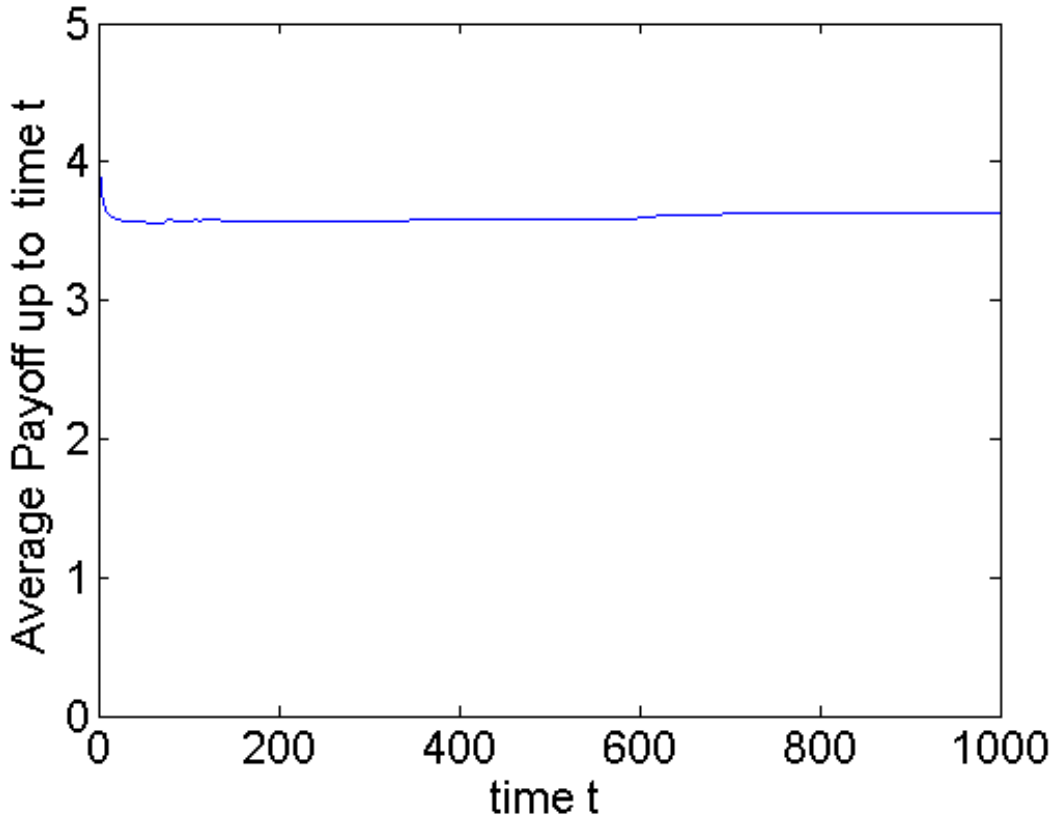

fig. 3: First the curve drops, but enhances continuously on a high level; strategy 4, mutation= 0.001, T= 100‘000

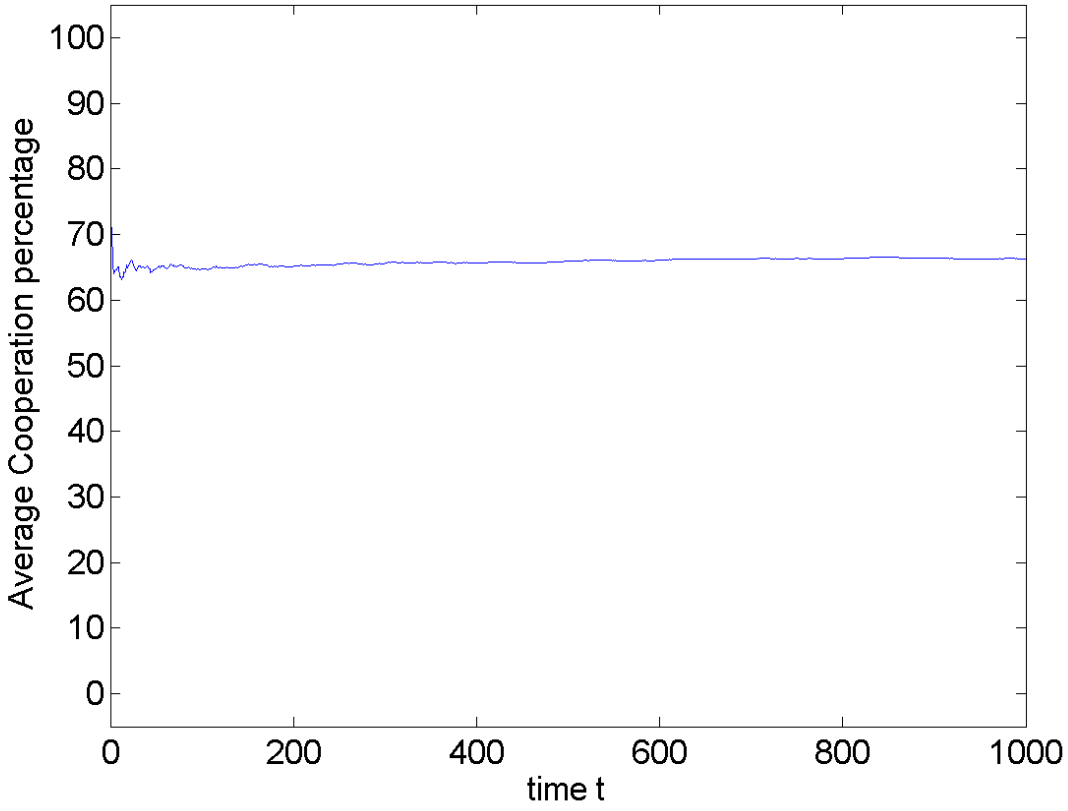

fig. 4: First the curve shows a strong oscillation, followed by a damping; strategy 4, mutation= 0.001, T= 100‘000

3.2) Method with 3 players

	mean payoff
ALLC	0.39
ALLD	0.47
Imitation1	0.63
Imitation2	0.51
Imitation3	0.51
Reinforcement	0.63
best reply	0.63

Table 6: method with 3 players; mutation= 0, T= 100‘000

As we can see the strategy who use memory receive higher payoffs than the one without memories. (ALLC, ALLD)
The figures are not interesting, because without mutations they are more or less stable.

All in all we would need more approaches to reality. In reality the personal relation between the 2 players is very important.
Because of the different payoffs and of the different strategies we can‘t compare the 2-player with the 3-player dilemma.

The best strategy was often the expected one,the results correlated with some references. [4,5]

4.) Acknowledgement

We thank A. Johansson & W. Yu for their lectures and their advices.

5) References

[1] http://en.wikipedia.org/wiki/Unscrupulous_diner%27s_dilemma
[2] Lecture with Computer Exercises: Modelling and Simulating Social Systems with MATLAB; lesson 9, page 14
[3] Project report: Axelrod Tournament with Mutations; Sri Krishna Suresh Kumar, Navish Kumar Batchu, 2009
[4] http://de.wikipedia.org/wiki/Nash_Equilibrium
[5] http://de.wikipedia.org/wiki/Tit_for_tat

6) Appendix

I Algorithm without memory, DD: [5 0; 6 1], strategies ALLC, ALLD, c. p. 1/3, c. p. 2/3, 2-player
II Algorithm with memory, DD: [5 0; 6 1], strategies 1-12, 2-player
III Algorithm with memory, DD: [5 0; 6 1], strategies ALLC, ALLD, Imitation1-3, reinforcement, best reply, 3-player

```
% This algorithm describes a static behaviour of the diner‘s dilemma in the
% sense that there is no memory and that the probability for cooperation
% or defection is fixed. The mutation is not included.


% parametric variables
% matrix with the diners dillema payoff scores (matrix choosen such that it
% corresponds to  prisoner‘s dilemma)
DD = [5 0;6 1];

% This is the total number of times each player meets the other player,
% that is the total number of times the players should interact with each
% other.
T = 100000;



% non parametric variables
% The player matrix has 5 rows. Here is what each row means:
% 1-4 : The probabilities of cooperation is based on the previous moves
% Column 4 : The initial move of the strategy 1 if cooperated, 2 if defected
% Stratgies considered 1 : ALLC. 2 : ALLD. 3:.prob. of coorp. = 1/3.
% 4: prob. of coorp. = 2/3
StrategyTemplate = [
   1 1 1 1
   0 0 0 2
   1 0 0 1
   1 1 0 1
   ];

% variable for storing the total number os strategies in question
NoOfStrategy = size(StrategyTemplate,1);

% the final payoff matrix (square matrix)
PayOff = zeros(NoOfStrategy,NoOfStrategy);

% variables for storing different moves
CurrPlayerCurrentMove = 0;
OpponentPlayerCurrentMove = 0;
CurrentPlayerPayoff = 0;
OpponentPlayerPayoff = 0;



% every strategy interacts with every other strategy
for i = 1:NoOfStrategy
   for j = i:NoOfStrategy
      % initialisation
      CurrPlayerPreviousMove = StrategyTemplate(i,4);
      OpponentPlayerPreviousMove = StrategyTemplate(j,4);
```

```
CurrentPlayerPayoff = 0;
OpponentPlayerPayoff = 0;

% for a given duration of time
for t = 1:T

    % calculating the current player‘s move
    if(i==1)
       CurrPlayerCurrentMove = 1;
    end
     if(i==2)
       CurrPlayerCurrentMove = 2;
     end

    if(i==3)
      if(rand<=1/3)
         CurrPlayerCurrentMove = 1;
       else CurrPlayerCurrentMove = 2;
      end
    end

    if(i==4)
      if(rand<=2/3)
         CurrPlayerCurrentMove = 2;
       else CurrPlayerCurrentMove = 1;
      end
    end



    % calculating the opponent player‘s move
     if(j==1)
       OpponentPlayerCurrentMove = 1;
     end
     if(j==2)
       OpponentPlayerCurrentMove = 2;
     end

    if(j==3)
      if(rand<=1/3)
         OpponentPlayerCurrentMove = 1;
       else OpponentPlayerCurrentMove = 2;
      end
    end

    if(j==4)
      if(rand<=2/3)
         OpponentPlayerCurrentMove = 2;
       else OpponentPlayerCurrentMove = 1;
      end
    end
```

```
        % updating the values for each strategy
        CurrentPlayerPayoff = CurrentPlayerPayoff + DD(CurrPlayerCurrentMove,OpponentPlayerCurrent
Move);
        OpponentPlayerPayoff = OpponentPlayerPayoff + DD(OpponentPlayerCurrentMove,CurrPlayerCur
rentMove);


      % updating the payoff in the payoff matrix
      if (i~=j)
         PayOff(i,j) = CurrentPlayerPayoff/T;
         PayOff(j,i) = OpponentPlayerPayoff/T;
      else
         PayOff(i,i) = ((CurrentPlayerPayoff + OpponentPlayerPayoff)/2)/T;
      end


      end
   end
end



% displaying the result
disp(PayOff)

meanpayoff =zeros(NoOfStrategy,1);
b = 0;
for i = 1:NoOfStrategy
   for j = 1:NoOfStrategy
   b = PayOff(i,j);
   meanpayoff(i) = meanpayoff(i) + b;
   end
end

% displaying the mean payoff
disp(meanpayoff/NoOfStrategy)
```

```
% This algorithm describes a dynamic behaviour of the diner‘s dilemma in the
% sense that there is  a memory of the previous round influencing the
% probability for cooperation or defection in the current round.

% parametric variables
% matrix with the diners dillema payoff scores (matrix choosen such that it
% corresponds to prisoner‘s dilemma)
PD = [5 0;6 1];

% This is the total number of times each player meets the other player,
% that is the total number of times the players should interact with each
% other.
T = 100000;



% non parametric variables
% The player matrix has 3 rows. Here is what each row means:
% 1-4 : The probabilities of cooperating is based on the previous moves
% according to the p-q representation. p = probability of cooperation if
% the other guy did cooperate in the previous round; q = probability of
% cooperation, if the other guy did defect in the previous round.
% Strategies considered 1 : ALLC. 2 : ALLD. 3:TFT 4:.prob. of coorp. = 1 if
% the other person did coorporate, but = 1/3 if the other guy did defect.
% 5: prob. of coorp. = 1 if the other person did coorperate, but = 2/3 if the
% other guy did defect.
% row 6: entries 2 2 means just that it does not behave according to a p-q
% model, but takes the opponent‘s previous decision.
% row 7:entries 3 3 means just that it does not behave according to a p-q
% model, but takes the opponent‘s previous decision, if it was more or
% equally succesfull, otherwise coorperation.
% row 8:entries 3 3 means just that it does not behave according to a p-q
% model, but takes the opponent‘s previous decision, if it was strictly more
% succesfull, otherwise coorperation.
% row 9:entries 3 3 means just that it does not behave according to a p-q
% model, but takes the opponent‘s previous decision, if it was strictly more
% succesfull, otherwise its previous decision.
% row 10:entries 3 3 means just that it does not behave according to a p-q
% model, but takes the opponent‘s previous decision, if it was  more or
% equally succesfull, otherwise its previous decision.
% row 11:entries 4 4 means just that it does not behave according to a p-q
% model, but takes the opponents previous decision, if it was more or
% equally succesfull, otherwise defection.
% row 12:entries 4 4 means just that it does not behave according to a p-q
% model, but takes the opponent‘s previous decision, if it was strictly more
% succesfull, otherwise defection.
% Notice: This model includes all considered strategies (with strategy
% 7,10,11).
```

```
% Column 3 : The initial move of the strategy 1 if cooperated, 2 if
% defected

StrategyTemplate = [
   1 1      1
   0 0      2
   1 0      1
   1 1/3    1
   1 2/3    1
   2 2      1
   3 3      1
   3 3      1
   3 3      1
   3 3      1
   4 4      2
   4 4      2
   ];

% the mutation scale
Mutation = 0.001;

% variable for storing the total number of strategies in question
NoOfStrategy = size(StrategyTemplate,1);

% the final payoff matrix (square matrix)
PayOff = zeros(NoOfStrategy,NoOfStrategy);

% variables for storing different moves
CurrentPlayerCurrentMove = 0;
CurrentPlayerPreviousMove = 0;
OpponentPlayerCurrentMove = 0;
OpponentPlayerPreviousMove = 0;
CurrentPlayerPayoff = 0;
OpponentPlayerPayoff = 0;

% variable used to display the mean playoff vs. time and the percentage of
% cooperation vs. time
Payoffattimet = zeros(NoOfStrategy,NoOfStrategy,T/100);
T2 = [1:T/100]';
meanpayoff2 =zeros(NoOfStrategy,T/100);
b2 = zeros(T/100,1);
b1 = zeros(T/100,1);
Cooppercentage1 =  zeros(NoOfStrategy,NoOfStrategy);
Cooppercentage2 = zeros(NoOfStrategy,NoOfStrategy,T/100);
Cooppercentage3 = zeros(NoOfStrategy,T/100);


% every strategy interacts with every other strategy
for i = 1:NoOfStrategy
   for j = 1:NoOfStrategy
      % initialisation
      CurrentPlayerCurrentMove = StrategyTemplate(i,3);
      OpponentPlayerCurrentMove = StrategyTemplate(j,3);
```

```
CurrentPlayerPayoff = 0;
OpponentPlayerPayoff = 0;

% for a given duration of time  i.e. how many interactions take place
for t = 1:T
   k = mod(t,100);
   t2 = floor( t/100 );

% update the previous moves
CurrentPlayerPreviousMove = CurrentPlayerCurrentMove;
OpponentPlayerPreviousMove =  OpponentPlayerCurrentMove;

   % calculating the current player‘s move

  if(i==1)
     CurrentPlayerCurrentMove = 1;
   end
   if(i==2)
     CurrentPlayerCurrentMove = 2;
   end

   if(i==3)
     if(OpponentPlayerPreviousMove==1)
        CurrentPlayerCurrentMove = 1;
     else
        CurrentPlayerCurrentMove = 2;
     end

   end

   if(i==4)
     if(OpponentPlayerPreviousMove==1)
        CurrentPlayerCurrentMove = 1;
        else
           if(rand<=1/3)
              CurrentPlayerCurrentMove = 1;
           else
              CurrentPlayerCurrentMove = 2;
           end
     end
   end

   if(i==5)
     if(OpponentPlayerPreviousMove==1)
        CurrentPlayerCurrentMove = 1;
        else
           if(rand<=2/3)
              CurrentPlayerCurrentMove = 1;
           else
              CurrentPlayerCurrentMove = 2;
           end
     end
   end
```

```
        if(i==6)
           CurrentPlayerCurrentMove=OpponentPlayerPreviousMove;
        end


        if(i==7)
           if(PD(CurrentPlayerPreviousMove,OpponentPlayerPreviousMove) <= PD(OpponentPlayerPrevio
usMove,CurrentPlayerPreviousMove))
           CurrentPlayerCurrentMove=OpponentPlayerPreviousMove;
           else CurrentPlayerCurrentMove =1;
           end
        end
        if(i==8)
           if(PD(CurrentPlayerPreviousMove,OpponentPlayerPreviousMove) < PD(OpponentPlayerPrevious
Move,CurrentPlayerPreviousMove))
           CurrentPlayerCurrentMove=OpponentPlayerPreviousMove;
           else CurrentPlayerCurrentMove =1;
           end
        end

        if(i==9)
           if( PD(CurrentPlayerCurrentMove,OpponentPlayerCurrentMove) < PD(OpponentPlayerPrevious
Move,CurrentPlayerPreviousMove))
           CurrentPlayerCurrentMove=OpponentPlayerPreviousMove;
           else CurrentPlayerCurrentMove = CurrentPlayerPreviousMove;
           end
        end
        if(i==10)
           if(PD(CurrentPlayerPreviousMove,OpponentPlayerPreviousMove) <= PD(OpponentPlayerPrevio
usMove,CurrentPlayerPreviousMove))
           CurrentPlayerCurrentMove=OpponentPlayerPreviousMove;
           else CurrentPlayerCurrentMove = CurrentPlayerPreviousMove;
           end
        end

        if(i==11)
           if(PD(CurrentPlayerPreviousMove,OpponentPlayerPreviousMove) <= PD(OpponentPlayerPrevio
usMove,CurrentPlayerPreviousMove))
           CurrentPlayerCurrentMove=OpponentPlayerPreviousMove;
           else CurrentPlayerCurrentMove =2;
           end
        end
        if(i==12)
           if(PD(CurrentPlayerPreviousMove,OpponentPlayerPreviousMove) < PD(OpponentPlayerPrevious
Move,CurrentPlayerPreviousMove))
           CurrentPlayerCurrentMove=OpponentPlayerPreviousMove;
           else CurrentPlayerCurrentMove =2;
           end
        end
```

```
% calculating the opponent player‘s move
if(j==1)
   OpponentPlayerCurrentMove = 1;
end
if(j==2)
   OpponentPlayerCurrentMove = 2;
end

 if(j==3)
   if(CurrentPlayerPreviousMove==1)
      OpponentPlayerCurrentMove = 1;
   else
      OpponentPlayerCurrentMove = 2;
   end

end

if(j==4)
   if(CurrentPlayerPreviousMove==1)
      OpponentPlayerCurrentMove = 1;
      else
         if(rand<=1/3)
            OpponentPlayerCurrentMove = 1;
         else
            OpponentPlayerCurrentMove = 2;
         end
   end
end

if(j==5)
   if(CurrentPlayerPreviousMove==1)
      OpponentPlayerCurrentMove = 1;
      else
         if(rand<=2/3)
            OpponentPlayerCurrentMove = 1;
         else
            OpponentPlayerCurrentMove = 2;
         end
   end
end

if(j==6)
   OpponentPlayerCurrentMove=CurrentPlayerPreviousMove;
end

if(j==7)
   if(PD(CurrentPlayerPreviousMove,OpponentPlayerPreviousMove) >= PD(OpponentPlayerPrevio
usMove,CurrentPlayerPreviousMove))
   OpponentPlayerCurrentMove=CurrentPlayerPreviousMove;
   else OpponentPlayerCurrentMove =1;
   end
end
if(j==8)
```

```
            if(PD(CurrentPlayerPreviousMove,OpponentPlayerPreviousMove) > PD(OpponentPlayerPrevious
Move,CurrentPlayerPreviousMove))
            OpponentPlayerCurrentMove=CurrentPlayerPreviousMove;
            else OpponentPlayerCurrentMove =1;
            end
        end

        if(j==9)
            if(PD(CurrentPlayerPreviousMove,OpponentPlayerPreviousMove) > PD(OpponentPlayerPrevious
Move,CurrentPlayerPreviousMove))
            OpponentPlayerCurrentMove=CurrentPlayerPreviousMove;
            else OpponentPlayerCurrentMove = OpponentPlayerPreviousMove;
            end
        end
        if(j==10)
            if(PD(CurrentPlayerPreviousMove,OpponentPlayerPreviousMove) >= PD(OpponentPlayerPrevio
usMove,CurrentPlayerPreviousMove))
            OpponentPlayerCurrentMove=CurrentPlayerPreviousMove;
            else OpponentPlayerCurrentMove = OpponentPlayerPreviousMove;
            end
        end

        if(j==11)
            if(PD(CurrentPlayerPreviousMove,OpponentPlayerPreviousMove) >= PD(OpponentPlayerPrevio
usMove,CurrentPlayerPreviousMove))
            OpponentPlayerCurrentMove=CurrentPlayerPreviousMove;
            else OpponentPlayerCurrentMove =2;
            end
        end
        if(j==12)
            if(PD(CurrentPlayerPreviousMove,OpponentPlayerPreviousMove) > PD(OpponentPlayerPrevious
Move,CurrentPlayerPreviousMove))
            OpponentPlayerCurrentMove=CurrentPlayerPreviousMove;
            else OpponentPlayerCurrentMove =2;
            end
        end









        % updating the values for each strategy
        CurrentPlayerPayoff = CurrentPlayerPayoff + PD(CurrentPlayerCurrentMove,OpponentPlayerCurre
ntMove);
        OpponentPlayerPayoff = OpponentPlayerPayoff + PD(OpponentPlayerCurrentMove,CurrentPlayerC
urrentMove);

        % update percentage of cooperation such that 0 stands for cooperation and 1 for
        % decooperation
```

```
        Cooppercentage1(i,j) =  Cooppercentage1(i,j)+CurrentPlayerCurrentMove-1;


        % updating the payoff in the payoff matrix such that the payoff is the
        % total value, not yet divided by T

        PayOff(i,j) = CurrentPlayerPayoff;


        % used to display time evolution of average payoff values
        % (average values up to time t)
        %  to be able to calculate the meanplayoff for different strategies
        % used to display time evolution of cooperation percentage
        % (average values up to time t)

       if( k == 0)
        Payoffattimet(i,j,t2)= PayOff(i,j)/t;

       Cooppercentage2(i,j,t2) =100* (1-Cooppercentage1(i,j)/t);
       end



     end
   end



end

 % displaying the time development of the average payoff for each strategy
 for m = 1:NoOfStrategy
    for s = 1: (T/100)
            for l = 1:NoOfStrategy


                b1 = Cooppercentage2(m,l,s);
                Cooppercentage3(m,s) = b1 + Cooppercentage2(m,l,s);

                b2 =  Payoffattimet(m,l,s);
                meanpayoff2(m,s) =  meanpayoff2(m,s) + b2 ;
            end
    end
 end
% total percentage of cooperation
Totcoopperc = zeros(NoOfStrategy,1);
 for j= 1:NoOfStrategy
    Totcoopperc(i) =  Cooppercentage2(i,j,T/100)+Totcoopperc(i);
 end
    % Displaying the result
disp(Totcoopperc/NoOfStrategy)
```

```
 % displaying the result for the average cooperation percentage vs. time t
        for n = 1:NoOfStrategy
        figure;
        plot(T2, Cooppercentage3(n,T2)‘);
        set(gca, ,Fontsize‘,16);
        xlabel(,time t‘);
        ylabel(,Average Cooperation percentage ,);
        ylim([-5,105]);
        xlim([0,T/100]);
        end

% displaying the result for the average payoff vs. time t
        for n = 1:NoOfStrategy
        figure;
        plot(T2,meanpayoff2(n,T2)‘/NoOfStrategy);
        set(gca, ,Fontsize‘,16);
        xlabel(,time t‘);
        ylabel(,Average Payoff up to  time t‘);
        ylim([0,4]);
        xlim([0,T/100]);
        end


% displaying the result
disp(PayOff/T)

% calculate the meanplayoff of the different results
meanpayoff =zeros(NoOfStrategy,1);
b = 0;
for i = 1:NoOfStrategy
    for j = 1:NoOfStrategy
    b = PayOff(i,j)/T;
    meanpayoff(i) = ( meanpayoff(i) + b) ;
    end
end

% displaying the result
disp(meanpayoff/NoOfStrategy)
```

```
% creating a payoff matrix for n=3 players according to the diner‘s dilemma
% game with the parameters h=9 (high price), g= 8 (joy of eating expensive
% menu), b=4 (joy of eating cheap menu) l = 3 (low price)
% no mutation is included

a = zeros(2,2,2);

% choosen such that the 2 stands for the high price menu, 1 for the low price menu

a(1,1,1)=1;
a(1,2,1)=-1;
a(2,1,1)=3;
a(2,2,1)=1;

a(1,1,2)=-1;
a(1,2,2)=-2;
a(2,1,2)=1;
a(2,2,2)=-1;



% total number of times each player meets the other player, that is total
% number of times the players should interact with each other
T = 100000;



% non parametric variables
% The player matrix has 3 rows. Here is what each row means:
% 1-2 : The probabilities of cooperation is based on the previous move
% according to the p-q representation. p = probability of cooperating if
% the other guy did cooperate in the previous round; q = probability of
%cooperation, if the other guy did defect in the previous round
% strategies considered row 1 : ALLC. and  row 2 : ALLD.
% row 3-8: first entries are random numbers of no importance,  meaning
% that they do not behave according to a p-q model, further explanations
% above the particular strategies.
% Column 3 : The initial move of the strategy 1: if cooperated, 2 if defected
% cooperation means low price, defection high price
StrategyTemplate = [
   1 1     1
   0 0     2
   2 2     1
   2 2     1
   2 2     1
   2 2     1
   2 2     1
   ];
```

```
% variable for storing the total number of strategies in question
NoOfStrategy = size(StrategyTemplate,1);

% the final payoff matrix (cubic matrix)
PayOff = zeros(NoOfStrategy,NoOfStrategy,NoOfStrategy);

% variables for storing different moves
CurrentPlayerCurrentMove = 0;
CurrentPlayerPreviousMove = 0;
OpponentPlayerCurrentMove = 0;
OpponentPlayerPreviousMove = 0;
OpponentPlayer2CurrentMove = 0;
OpponentPlayer2PreviousMove = 0;
CurrentPlayerPayoff = 0;
OpponentPlayerPayoff = 0;
OpponentPlayer2Payoff = 0;

% variables used to display the mean playoff vs. time and the percentage of
% cooperation vs. time
Payoffattimet = zeros(NoOfStrategy,NoOfStrategy,NoOfStrategy,T/100);
T2 = [1:T/100]';
meanpayoff2 =zeros(NoOfStrategy,T/100);
b2 = zeros(T/100,1);



% every strategy interacts with every other strategy
for i = 1:NoOfStrategy
   for j = 1:NoOfStrategy
      for h = 1: NoOfStrategy
      % initialisation
      CurrentPlayerCurrentMove = StrategyTemplate(i,3);
      OpponentPlayerCurrentMove = StrategyTemplate(j,3);
      OpponentPlayer2CurrentMove = StrategyTemplate(h,3);
      CurrentPlayerPayoff = 0;
      OpponentPlayerPayoff = 0;
      OpponentPlayer2Payoff = 0;

      % for a given duration of time
      for t = 1:T
         k = mod(t,100);
         t2 = floor( t/100 );

      % update the previous moves
      CurrentPlayerPreviousMove = CurrentPlayerCurrentMove;
      OpponentPlayerPreviousMove =  OpponentPlayerCurrentMove;
      OpponentPlayer2PreviousMove =  OpponentPlayer2CurrentMove;

        % calculating the current players move

         if(i==1)
            CurrentPlayerCurrentMove = 1;
```

```
end
if(i==2)
   CurrentPlayerCurrentMove = 2;
end
 % imitation
if(i==3)
  if(OpponentPlayerPreviousMove==1 && OpponentPlayer2PreviousMove==1)
     CurrentPlayerCurrentMove = 1;
  else
     CurrentPlayerCurrentMove = 2;
  end

end
 % imitation
if(i==4)
  if(OpponentPlayerPreviousMove==2 && OpponentPlayer2PreviousMove==2)
     CurrentPlayerCurrentMove = 2;
     else
      CurrentPlayerCurrentMove = 1;

  end
end

% imitation
if(i==5)
  if(OpponentPlayerPreviousMove+OpponentPlayer2PreviousMove==3)
     CurrentPlayerCurrentMove = 1;
  else  if(a(OpponentPlayerPreviousMove,CurrentPlayerPreviousMove,OpponentPlayer2PreviousM
ove)>=a(OpponentPlayer2PreviousMove,CurrentPlayerPreviousMove,OpponentPlayerPreviousMove))
       CurrentPlayerCurrentMove= OpponentPlayerPreviousMove;
       else
        CurrentPlayerCurrentMove= OpponentPlayer2PreviousMove;
       end
  end
end
% reinforcement
if(i==6)
   if(OpponentPlayerPreviousMove>=OpponentPlayer2PreviousMove)
      if(OpponentPlayerPreviousMove>=CurrentPlayerPreviousMove)
      CurrentPlayerCurrentMove= OpponentPlayerPreviousMove;
      else CurrentPlayerCurrentMove= CurrentPlayerPreviousMove;
      end
   else
      if(OpponentPlayer2PreviousMove>=CurrentPlayerPreviousMove)
      CurrentPlayerCurrentMove= OpponentPlayer2PreviousMove;
      else CurrentPlayerCurrentMove= CurrentPlayerPreviousMove;
      end
   end
end



% best reply (Notice: hch yields the higher payoff than cch)
if(i==7)
```

```
    if(OpponentPlayerPreviousMove==OpponentPlayer2PreviousMove)
       if(OpponentPlayerPreviousMove==1)
          CurrentPlayerCurrentMove=1;
       else
          CurrentPlayerCurrentMove=2;
       end
    else
       CurrentPlayerCurrentMove=2;
    end
end




if(j==1)
   OpponentPlayerCurrentMove = 1;
end
if(j==2)
   OpponentPlayerCurrentMove = 2;
end

if(j==3)
  if(CurrentPlayerPreviousMove==1 && OpponentPlayer2PreviousMove==1)
     OpponentPlayerCurrentMove = 1;
  else
     OpponentPlayerCurrentMove = 2;
  end

end

if(j==4)
  if(CurrentPlayerPreviousMove==2 && OpponentPlayer2PreviousMove==2)
     OpponentPlayerCurrentMove = 2;
     else
      OpponentPlayerCurrentMove = 1;

  end
end



if(j==5)
  if(CurrentPlayerPreviousMove+OpponentPlayer2PreviousMove==3)
     OpponentPlayerCurrentMove = 1;
  else  if(a(CurrentPlayerPreviousMove,OpponentPlayerPreviousMove,OpponentPlayer2PreviousM
ove)>=a(OpponentPlayer2PreviousMove,CurrentPlayerPreviousMove,OpponentPlayerPreviousMove))
       OpponentPlayerCurrentMove= CurrentPlayerPreviousMove;
       else
        OpponentPlayerCurrentMove= OpponentPlayer2PreviousMove;
       end
  end
end

if(j==6)
```

```
    if(CurrentPlayerPreviousMove>=OpponentPlayer2PreviousMove)
       if(CurrentPlayerPreviousMove>=OpponentPlayerPreviousMove)
       OpponentPlayerCurrentMove= CurrentPlayerPreviousMove;
       else OpponentPlayerCurrentMove= OpponentPlayerPreviousMove;
       end
    else
       if(OpponentPlayer2PreviousMove>=OpponentPlayerPreviousMove)
       OpponentPlayerCurrentMove= OpponentPlayer2PreviousMove;
       else OpponentPlayerCurrentMove= OpponentPlayerPreviousMove;
       end
    end
end

if(j==7)
    if(CurrentPlayerPreviousMove==OpponentPlayer2PreviousMove)
       if(CurrentPlayerPreviousMove==1)
          OpponentPlayerCurrentMove=1;
       else
          OpponentPlayerCurrentMove=2;
       end
    else
       OpponentPlayerCurrentMove=2;
    end
end



if(h==1)
    OpponentPlayer2CurrentMove = 1;
end
if(h==2)
    OpponentPlayer2CurrentMove = 2;
end

if(h==3)
   if(CurrentPlayerPreviousMove==1 && OpponentPlayer2PreviousMove==1)
      OpponentPlayerCurrentMove = 1;
   else
      OpponentPlayerCurrentMove = 2;
   end

end

if(h==4)
   if(CurrentPlayerPreviousMove==2 && OpponentPlayerPreviousMove==2)
      OpponentPlayer2CurrentMove = 2;
      else
       OpponentPlayer2CurrentMove = 1;

   end
end



if(h==5)
```

```
            if(CurrentPlayerPreviousMove+OpponentPlayer2PreviousMove==3)
               OpponentPlayer2CurrentMove = 1;
            else  if(a(CurrentPlayerPreviousMove,OpponentPlayerPreviousMove,OpponentPlayer2PreviousM
ove)>=a(OpponentPlayerPreviousMove,CurrentPlayerPreviousMove,OpponentPlayer2PreviousMove))
                 OpponentPlayer2CurrentMove= CurrentPlayerPreviousMove;
                 else
                  OpponentPlayer2CurrentMove= OpponentPlayer2PreviousMove;
                 end
            end
          end

          if(h==6)
            if(CurrentPlayerPreviousMove>=OpponentPlayerPreviousMove)
               if(CurrentPlayerPreviousMove>=OpponentPlayer2PreviousMove)
               OpponentPlayer2CurrentMove= CurrentPlayerPreviousMove;
               else OpponentPlayer2CurrentMove= OpponentPlayer2PreviousMove;
               end
            else
               if(OpponentPlayerPreviousMove>=OpponentPlayer2PreviousMove)
               OpponentPlayer2CurrentMove= OpponentPlayerPreviousMove;
               else OpponentPlayer2CurrentMove= OpponentPlayer2PreviousMove;
               end
            end
          end

          if(h==7)
            if(CurrentPlayerPreviousMove==OpponentPlayerPreviousMove)
               if(CurrentPlayerPreviousMove==1)
                  OpponentPlayer2CurrentMove=1;
               else
                  OpponentPlayer2CurrentMove=2;
               end
            else
               OpponentPlayer2CurrentMove=2;
            end
          end



          % updating the values for each strategy
          CurrentPlayerPayoff = CurrentPlayerPayoff + a(CurrentPlayerCurrentMove,OpponentPlayerCurrent
Move,OpponentPlayer2CurrentMove);
          OpponentPlayerPayoff = OpponentPlayerPayoff + a(OpponentPlayerCurrentMove,CurrentPlayerCur
rentMove,OpponentPlayer2CurrentMove);
          OpponentPlayer2Payoff = OpponentPlayer2Payoff + a(OpponentPlayer2CurrentMove,CurrentPlayer
CurrentMove,OpponentPlayerCurrentMove);


        % updating the payoff in the payof matrix such that PayOff(i,j,h) is the
        % total payoff upt to time t, not yet divided by t

          PayOff(i,j,h) = CurrentPlayerPayoff;
```

```
      % used to display time evolution of average payoff values
          % (average values up to time t)
          %  to be able to calculate the meanplayoff for different
          %  strategies

         if( k == 0)
          Payoffattimet(i,j,h,t2)= PayOff(i,j,h)/t;


         end



      end
      end
   end



end

% calculating the time development of the average payoff for each strategy
 for m = 1:NoOfStrategy
    for t2 = 1: (T/100)
            for l = 1:NoOfStrategy
               for p = 1:NoOfStrategy



                b2 =  Payoffattimet(m,l,p,t2);
                meanpayoff2(m,t2) =  meanpayoff2(m,t2) + b2 ;
               end
            end
    end
 end




% displaying the result for the average payoff vs. time t
       for n = 1:NoOfStrategy
       figure;
       plot(T2,meanpayoff2(n,T2)‘/NoOfStrategy/NoOfStrategy);
       set(gca, ,Fontsize‘,16);
       xlabel(,time t‘);
       ylabel(,Average Payoff up to  time t‘);
       ylim([-1,1]);
       xlim([0,T/100]);
       end
```

```
% displaying the result
disp(PayOff/T)

% calculate the meanplayoff of the different results
meanpayoff =zeros(NoOfStrategy,1);
b = 0;
for i = 1:NoOfStrategy
    for j = 1:NoOfStrategy
        for p = 1:NoOfStrategy
    b = PayOff(i,j,p)/T;
    meanpayoff(i) = ( meanpayoff(i) + b) ;
        end
    end
end

% displaying the result
disp(meanpayoff/NoOfStrategy/NoOfStrategy)
```