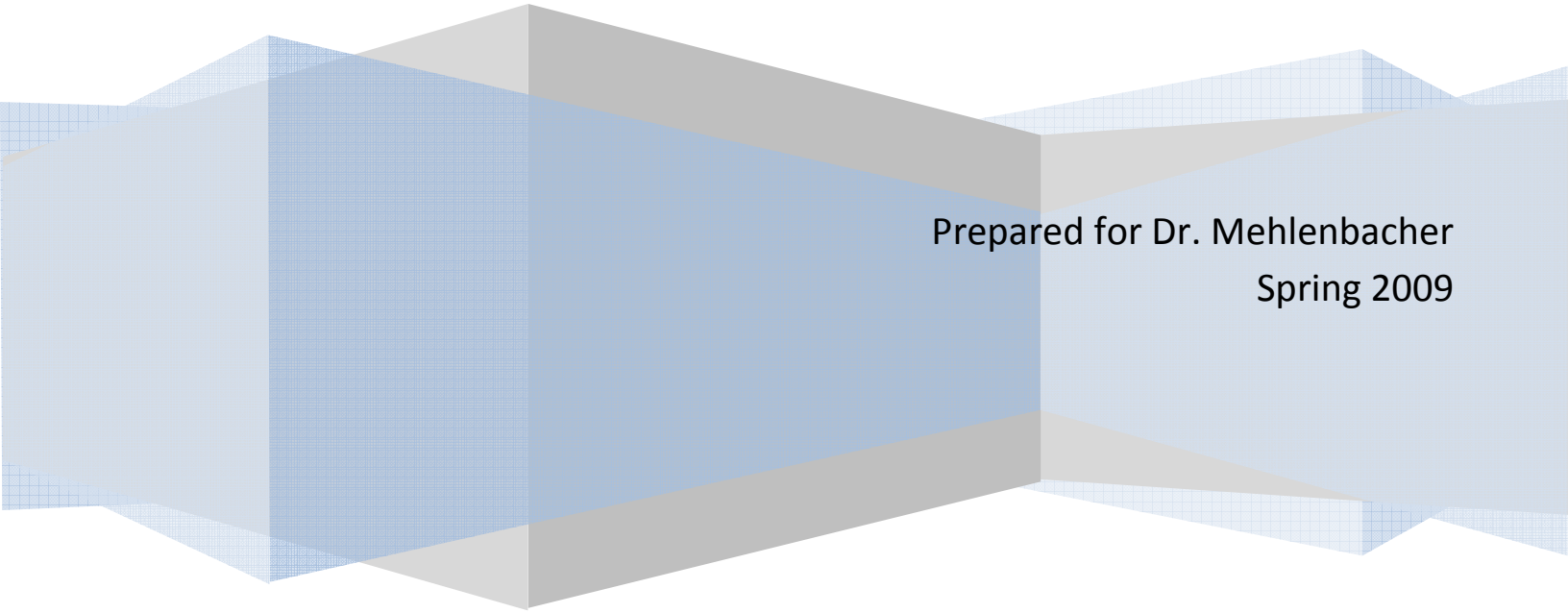
Econ 353 Final Project

Game Theory Portfolio

A Computational Exploration of the Nash Equilibrium

Nouri Najjar



Prepared for Dr. Mehlenbacher

Spring 2009

Introduction

Rather than focus on a specific economic model for my project I have decided to cover a selection of topics from the game theory discipline. For each topic I have created several models in matlab. Each of these models is coded to determine the equilibrium solutions to a specific game theory problem. For example, my first model solves for the pure strategy Nash Equilibrium in a two player two strategy static game.

In most cases these models solve basic game theory problems, so performing experiments on them is of little interest. Instead, I focus most of my analysis on the logic and coding that went into the programming of the model. Of course, I perform some basic experiments to show that the programs actually work, but I will spend relatively little time reviewing the problems. I have also included a CD with copies of each of the matlab programs I've written so that you can confirm examine the code more closely.

Economic Model

Game theory is the study of optimization in situations of strategic interaction between one or more individuals. In game theory, these strategic interactions are called games and the individuals involved in them are called players. In the game each player has to choose a strategy to play from their set of possible strategies and an individual's payoff is determined by the combination of strategies played by all players involved. Games can be either static or dynamic. A static game is only played once, while a dynamic game is played multiple times. There are two types of dynamic games: sequential games and stage games. In sequential games players play one after another and in stage games a static game is played for a finite number of repetitions.

Game theory also distinguishes between types of information in the game environment: complete vs. incomplete and perfect vs. imperfect. For this project I always assume information is both complete and perfect. For information to be complete all players must know all the parameters and rules of the game. For information to be perfect each player must know all previous moves made in the game at the time they are to play.

The cornerstone of game theory is the Nash Equilibrium – the combination of each player's strategies such that each player has their highest payoff given every other player is playing their equilibrium strategy. Essentially, in a Nash Equilibrium each player is responding to the others' strategies in the best possible way. A Nash Equilibrium is the outcome that will occur when all players are playing their best responses. For an example of this, a symmetric two person two strategy Prisoner's Dilemma game is shown in figure 1.1. In this game, if player two plays cooperate then player one's best response is to defect. If player two plays defect then player one's best response is also to defect. As the two players are symmetric the reverse holds for player two. Therefore, the Nash Equilibrium is for both players to defect.

		Player 2	
		Cooperate	Defect
Player 1	Cooperate	3, 3	0,4
	Defect	4,0	1,1

Figure 1.1: Prisoner's Dilemma

The simplest game is static and involves two players each with two strategies. Games get more complicated with the addition of more strategies and players, dynamics, or incomplete or imperfect information.

Computational Model

Technically speaking, I do not use any one computational model in this experiment. However, to solve game theory problems requires a unique blend of logical and computational methods. So the "models" I use are generally a mix of standard game theory logical methods, such as iteration, elimination, and dominance, mixed with some programming and computational methods, such as if-else statements and for loops. As each problem requires a new model, I will describe the logic and programming methods used in the specific model's section.

As the games are modified the Nash Equilibrium definition has to be altered. Each of my models analyzes one of these equilibrium concepts. For static games there are two forms of equilibria: pure strategy and mixed strategy. A Nash Equilibrium is defined in pure strategies if it only includes one strategy for each player. For example, in the Prisoner's Dilemma game the Nash Equilibrium is pure as player one plays defect and player two plays defect. Although only one strategy is specified, there may be zero, one, or many pure strategy equlibria for a given game. For example, if the Prisoner's Dilemma game was modified so that cooperating was more appealing to both players given the other is cooperating then there are two pure strategy Nash Equilibria: both cooperate and both defect. This modified game is shown in figure 2.1.

		Player 2	
		Cooperate	Defect
Player 1	Cooperate	5, 5	0,4
	Defect	4,0	1,1

Figure 2.1: Modified Prisoner's Dilemma

A Nash Equilibrium is defined in mixed strategies if it is given by a set of probabilities for each pure strategy for each player. So, in the Prisoner's Dilemma game the mixed strategy Nash Equilibrium would specify the probability each player cooperates and the probability each player defects. If a game has no or multiple pure strategy Nash Equilibria then it must have a unique mixed strategy equilibrium. Incidentally, the mixed strategy Nash Equilibrium in this game is for each player to cooperate .8571 percent of the time and defect .1429 percent of the time.

For Dynamic games the Nash Equilibrium concept must also be extended. In a dynamic game a Nash Equilibrium may include strategies (called actions in dynamic games) for a potential situation that are not actually best responses if that situation were to arise. In order to account for all possible situations that arise, the subgame perfect Nash Equilibrium (SPNE) concept must be introduced. An equilibrium is subgame perfect if the strategies specified represent a Nash Equilibrium for every possible situation that could arise. This concept will be clarified later in this paper.

For this project I have written four programs: two that deal with static games and two that deal with dynamic games. The two static models cover pure and mixed strategies and the two dynamic models cover subgame perfection. In the first dynamic model the game solved involves two sequences of play. That is, player one plays then player two plays. In the second dynamic model the game solved involves three sequences. That is, player one plays first, followed by player two, followed again by player one. The models are summarized in brief in table 2.1.

Model	Game Time	Nash Equilibrium	Sequences
1	Static	Pure	NA
2	Static	Mixed	NA
3	Dynamic	Subgame Perfect	2
4	Dynamic	Subgame Perfect	3

Table 2.1: Game Theory Models

Model One – Pure Strategy Nash Equilibrium

Description

The game solved in this model is a two person two strategy static game. Some commonly used examples of this type of game are: the Prisoner's Dilemma, the game of Chicken, and the Battle of the Sexes. For simplicities sake I will use the Prisoner's Dilemma game in figure 1.1 as a solved example in this write up. However, this program will solve any two player two strategy static game.

The Programming and Logical Method

The first step in coding this model is to break the game matrix down into two sub-matrices, one representing the payoffs for each player. Player one's payoff matrix is named P1 and player two's is named P2.

Next, I create an isolated payoff vector (IPV) for each player and each strategy. In this problem player one has two IPVs, C1 and D1, and player two has two IPVs, C2 and D2. Each IPV takes the other player's strategy as given and then returns the payoffs corresponding to each of their own strategies.

Once the IPVs were created I determined the best responses for each individual given the other player's choice of strategy. The best responses are found by determining the largest scalar in each IPV. To do this the max function was used. One stipulation to this program is that the players cannot be indifferent between two possible strategies given the other player's strategy choice. That is, the scalars cannot be

equal in any given IPV. This is a result of the max function, as it cannot distinguish between two equal scalars.

The final step in solving this problem involves finding the Nash Equilibrium. This is the most complex step of the model both in terms of logic and code. From a logic perspective, this part is founded on the game theoretic concept of dominance. A strategy is dominant if it is always played regardless of the other player's choice of strategy. For example, in the Prisoner's Dilemma game defect is a dominant strategy. In a two by two game if at least one of the players has a dominant strategy then the equilibrium solution becomes very simple; the player(s) with a dominant strategy play that strategy and the other player plays her best response to that strategy. If there are no dominant strategies then the solution becomes more complex. To make things less confusing I will first define some terms:

- First best response: an individual's best response given the other player is playing their first strategy (in the PD game this would mean the other player is cooperating)
- Second best response: an individual's best response given the other player is playing their second strategy (in the PD game this would mean the other play is defecting)
- Best response is one: the individual's best response is to play the first strategy (in the PD game this means they will cooperate)
- Best response is two: the individual's best response is to play the second strategy (in the PD game this means they will defect)
- NE={1,2}: the Nash Equilibrium is for player one to play the first strategy and player two to play the second strategy

If both players' first best responses are one then one Nash Equilibrium is NE={1,1}. If both players' first best responses are two and player one's second best response is one then a Nash Equilibrium is NE={1,2). If both players' second best responses are two then one Nash Equilibrium is NE={2,2}. If both players' second best responses are one and player one's first best response is two then a Nash Equilibrium is NE={2,1}. Otherwise, if none of these conditions hold and there are no dominant strategies then there is no Nash Equilibrium.

As far as the actual matlab code went this model was fairly basic, although the logic behind the model made it more complicated. The most complicated part of programming was to do with the nested if statements. This was used heavily in the final step to find the Nash Equilibrium. Essentially, the if statement allowed me to test if certain logical conditions were satisfied, and if they were to evaluate the expression held in the body of the if statement. Nesting these statements allowed me to sequentially test if various conditions held, which was an extremely important step in parsing through the iterative logic of solving this kind of problem.

The Code

```
%Game Theory Model 1 - Pure Strategy Nash Equilibrium
```

```
%Nash Equilibrium: two player, two strategies, non-symmetric game

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 1: Initiate two payoff matrices.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
P1=[3 0;
    4 1];

P2=[3 4;
    0 1];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 2: Create an isolated payoff vector (IPV) for each player and each
strategy.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Player 1
C1=P1(:,1);
D1=P1(:,2);

%Player 2
C2=P2(1,:);
D2=P2(2,:);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 3: Find the best response functions for each player
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Player 1
[br11 bri11]=max(C1);
[br12 bri12]=max(D1);

%Player 2
[br21 bri21]=max(C2);
[br22 bri22]=max(D2);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 4: Find the Nash Equilibrium(s)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if (bri11==bri12) %P1 has a strictly dominant strategy
    if (bri21==bri22) %P2 has a strictly dominant strategy
        nash=[bri11 bri21];
    elseif (bri21~=bri22) %P2 does NOT have a strictly dominant strategy
        if (bri11==1)
            nash=[bri11 bri21];
        elseif (bri11==2)
            nash=[bri11 bri22];
        end;
    end;
elseif (bri11~=bri12) %P1 does NOT have a strictly dominant strategy
    if (bri21==bri22) %P2 has a strictly dominant strategy
        if (bri21==1)
            nash=[bri11 bri21];
        elseif (bri21==2)
            nash=[bri12 bri21];
        end;
    elseif (bri21~=bri22) %P2 does NOT have a strictly dominant strategy (no
dominant strategies)
        if (bri11==bri21)
            if (bri11==1)
                nash1=[bri11 bri21];
```

```
            elseif (bri11==2)
                    if (bri12==1)
                        nash1=[bri12 bri21];
                    elseif (bri12==2)
                        nash1=[0 0];
                    end;
            end;
        elseif (bri11~=bri21)
            nash1=[0 0];
        end;

        if (bri12==bri22)
            if (bri12==2)
                nash2=[bri12 bri22];
            elseif (bri12==1)
                if (bri11==2)
                    nash2=[bri11 bri22];
                elseif (bri11==1)
                    nash2=[0 0];
                end;
            end;
        elseif (bri12~=bri22)
            nash2=[0 0];
        end;

        if (nash1==nash2)
            nash=nash1;
        elseif (nash1~=nash2)
            nash=[nash1;
                  nash2];;
        end;
    end;
end;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 5: Display the Nash Equilibrium output.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Nash
```

The Results

The first game I will solve is the prisoner's dilemma. The game matrix associated with the problem is shown in figure 3.1. As I described earlier, the Nash Equilibrium for this game is NE={2,2}. The output is given in a matrix, where the first row contains the first Nash Equilibrium and the second row contains the second Nash Equilibrium (if it exists). The first column contains player one's strategy and the second column contains player two's strategy. If there is no Nash Equilibrium then the output will be given by a zero vector. The results of this game are shown in figure 3.2.

		Player 2	
		Cooperate	Defect
Player 1	Cooperate	3, 3	0,4
	Defect	4,0	1,1

Figure 3.1: Prisoner's Dilemma

PD Output

nash =

2 2

Figure 3.2

Next, I solve the game of Chicken. This game has two Nash Equilibria: NE={1,2} and NE={2,1}. The game matrix is shown in figure 3.3 and the equilbria are highlighted yellow. The results of this game are shown in figure 3.4

		Player 2	
		Swerve	Straight
Player 1	Swerve	0, 0	-1,1
	Straight	1,-1	-10,-10

Figure 3.3: Chicken

Chicken Output

nash =

1 2
2 1

Figure 3.4

Finally, I solve the Battle of the Sexes game. Again, this game has two Nash Equilibria except this time the equilibria are cooperative: NE={1,1} and NE={2,2}. The game matrix is shown in figure 3.5 and the results are shown in figure 3.6.

		Player 2	
		Opera	Football
Player 1	Opera	3, 2	0,0
	Football	0,0	2,3

Figure 3.5: Battle of the Sexes

BOTS Output

nash =

1 1
2 2

Figure 3.6

Model Two – Mixed Strategy Nash Equilibrium

Description

This model is an extension of the first model. Any game with multiple or no pure strategy equilibria will have a Nash Equilibrium in mixed strategies. As the first step in solving for a mixed strategy equilibrium is to determine if there are multiple or no pure strategy equilibria, this model includes the code from the first model. I will focus my write up on the additions to the code as I have already discussed the first model in depth.

The Programming and Logical Method

Once the pure strategy Nash Equilibrium is known, the first step in coding this model is to determine if the equilibrium is unique (and non zero). To do this I had to create several temporary storage variables. The first of these storage variables, tempnash, is used to determine if there is no Nash Equilibrium. The variables nashsizecalc, sizenash, and tempsizenash are used to determine the size of the Nash matrix. If tempsizenash is equal to one then there is only one pure strategy Nash Equilibrium. If it is equal to two then there are two pure strategy equilibria. Finally, the remainder of the code is enclosed in an if statement. This statement tests if there is no pure equilibrium (tempnash=0) or if there are multiple pure equilibria (tempsizenash=2) and evaluates the remainder of the code if either condition is satisfied.

Next, a warning is produced that alerts the programmer that there is a mixed strategy solution to this problem.

The second step is to create scalar variables that hold the payoffs for each individual for every possible combination of strategies. As both players have two strategies there are four possible outcomes, so four scalars are needed for each player. Technically speaking this step is redundant; however, it makes the coding process clearer and easier to understand.

The final step is to create and solve the expected utility functions for both players. Solving these functions will give the equilibrium probability values for each strategy for each player. Each expected utility function equates the payoffs for a given player for both choices of strategies. Figure 4.1 gives a generic representation of how this is done. Player one chooses strategy one with probability Rho1 and chooses strategy two with one minus that probability. Player two chooses strategy one with probability Rho2 and chooses strategy two with one minus that probability. Assuming player one chooses strategy one, his payoff is given in the first row of the last column. Assuming he chooses strategy two, his payoff is given in the second row of the last column. Equating these two equations gives player one's expected utility function. The same is done for player two, only her expected payoffs are given in the last row of the matrix.

			Player 2		EU1
			Rho2	1-Rho2	
			S1	S2	
Player 1	Rho1	S1	SS111, SS211	SS112, SS212	**SS111*Rho2 + SS112*(1-Rho2)**

	1-Rho1	S2	SS121, SS221	SS122, SS222	**SS121*Rho2 + SS122*(1-Rho2)**
EU2			**SS211*Rho1 + SS221*(1-Rho1)**	**SS212*Rho1 + SS222*(1-Rho1)**	

Figure 4.1: Expected Utility Matrix

Each player's function has only one unknown variable, the probability that that player will choose the first strategy (Rho1 for player one and Rho2 for player 2), which can be solved for. Once the probability of playing the first strategy is known subtracting that value from one gives the probability the second strategy is played. Once these values are known all that is left is to print out the mixed strategy Nash equilibrium. This is done in a two by two matrix, where the first row contains player one's strategy and the second row contains player two's. The first column contains the probability of playing the first strategy and the second column contains the probability of playing the second strategy.

The Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 1: Solves for the Mixed Strategy Nash Equilibrium in the case of
%multiple or non-existent Nash equilibria
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

tempnash=nash(1,1); %Creates a temp storage scalar (=0 if no pure NE)

nashsizecalc=nash(:,1); %Creates a temp col vector of the nash matrix
sizenash=size(nashsizecalc); %Returns the size of the nash matrix as a vector
tempsizenash=sizenash(:,1); %Returns a scalar of the size of the nash matrix
(1 or 2)

if ((tempnash==0) || (tempsizenash==2))

    NonUniqueSolutionWarning='There is no unique pure strategy Nash
Equilibrium. However, there is a mixed strategy Nash Equilibrium, which is
given in the matrix MSNE. The first row of MSNE contains player one's mixed
strategy and the second row contains player two's mixed strategy. The first
column shows the probability of playing strategy 1 and the second column
shows the probability of playing strategy 2.';
    NonUniqueSolutionWarning

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Step 2: Create players' strategy scalars
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %Player 1
    SS111=P1(1,1);
    SS112=P1(1,2);
    SS121=P1(2,1);
    SS122=P1(2,2);

    %Player 2
    SS211=P2(1,1);
    SS212=P2(1,2);
    SS221=P2(2,1);
```

```
SS222=P2(2,2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 3: Create expected utility functions and solve for players' mixed
%strategies
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%Player 1's mixed strategy
q=solve('(rho2*SS111 + (1-rho2)*SS112)-(rho2*SS121 + (1-rho2)*SS122)=0');
rho2=eval(q);
altrho2=1-rho2;

%Player 2's mixed strategy
p=solve('(rho1*SS211 + (1-rho1)*SS221)-(rho1*SS212 + (1-rho1)*SS222)=0');
rho1=eval(p);
altrho1=1-rho1;

MSNE=[rho1 altrho1;
      rho2 altrho2];
MSNE

end;
```

The Results

I first test this model on the game of Chickens problem used in the first model. As we found out, Chicken has multiple pure Nash Equilibria and so must have a unique mixed strategy equilibrium. To check that the program works I have solved this game for the MSNE by hand. The utility matrix for this problem is given in figure 4.2

			Player 2		
			Rho2	1-Rho2	**EU1**
			Swerve	Straight	
Player 1	Rho1	Swerve	0,0	-1,1	**0*Rho2 - 1*(1-Rho2)**
	1-Rho1	Straight	1,-1	-10,-10	**1*Rho2 - 10*(1-Rho2)**
EU2			**0*Rho1 - 1*(1-Rho1)**	**1*Rho1 - 10*(1-Rho1)**	

Figure 4.2: Chicken's Expected Utility Matrix

Solving for the expected utility functions gives Rho2=.9 and Rho1=.9. This corresponds to a mixed strategy equilibrium of both players choosing swerve with .9 probability and straight with .1 probability (MSNE={(.9,.1),(.9,.1)}). Solving this game with my program returns the same results. The output for this problem is given in figure 4.3.

Chicken Mixed Strategy Output
nash =

1 2
2 1

NonUniqueSolutionWarning =

There is no unique pure strategy Nash Equilibrium. However, there is a mixed strategy Nash Equilibrium, which is given in the matrix MSNE. The first row of MSNE gives player ones mixed strategy and the second row gives player twos mixed strategy. The first column gives the probability of playing strategy 1 and the second column gives the probability of playing strategy 2.

MSNE =

0.9000 0.1000
0.9000 0.1000

Figure 4.3

Next I solve the Battle of the Sexes game. The results for this game are shown in figure 4.4. Unlike the game of Chicken, BOTS doesn't have a symmetric equilibrium. The mixed strategy Nash equilibrium for this game is for player one to choose opera with .6 probability and football with .4 probability and player two to choose opera with .4 probability and football with .6 probability.

BOTS Mixed Strategy Output

nash =

1 1
2 2

NonUniqueSolutionWarning =

There is no unique pure strategy Nash Equilibrium. However, there is a mixed strategy Nash Equilibrium, which is given in the matrix MSNE. The first row of MSNE gives player ones mixed strategy and the second row gives player twos mixed strategy. The first column gives the probability of playing strategy 1 and the second column gives the probability of playing strategy 2.

MSNE =

0.6000 0.4000
0.4000 0.6000

Figure 4.4

Finally, I solve a new game: Matching Pennies. The matrix for this game is shown in figure 4.5. I picked this game as it has no Nash equilibrium and I wanted to test the program for this scenario. Solving this game out by hand gives the mixed equilibrium MSNE={(.5,.5),(.5,.5)}.

	Player 2	
	Heads	Tails

Player 1	Heads	1,-1	-1,1
	Tails	-1,1	1,-1

Figure 4.5: Matching Pennies

Solving this game with the program also returns the same answer. The output for this game is shown in figure 4.6.

Matching Pennies Output

```
nash =

   0    0

NonUniqueSolutionWarning =

There is no unique pure strategy Nash Equilibrium. However, there is a mixed strategy Nash
Equilibrium, which is given in the matrix MSNE. The first row of MSNE gives player ones mixed
strategy and the second row gives player twos mixed strategy. The first column gives the probability
of playing strategy 1 and the second column gives the probability of playing strategy 2.

MSNE =

   0.5000   0.5000
   0.5000   0.5000
```

Figure 4.6

Model Three – Two-Sequence Subgame Perfect Nash Equilibrium

Description

This is the first of my two dynamic models. In this model I solve a two player two strategy sequential game. As dynamic games have a time element to them, the standard game matrix is not suitable for presenting these problems. Instead, these games are represented by game trees, which allow the sequential nature of these games to be captured. An example of a game tree is shown in figure 5.1. This tree is broken down into nodes and edges connecting the nodes; a node represents a player and an edge represents a player's potential action. For example, in this tree at the first node player one has to choose between doing action S1 and action S2. Once this choice is made player two has to make his choice of action. The game ends after player two's choice is made and the result is determined by following the edges from the first to last node. For example, if both player one and two choose S1 their respective payoffs would be P11 and P21. As this game has two player nodes it is a two-sequence game.

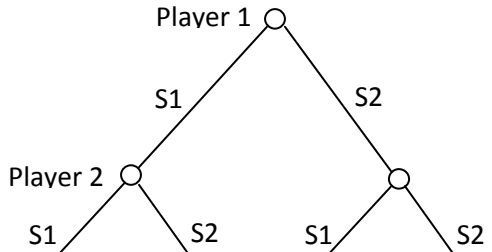


Figure 5.1: Sample Game Tree

As a solved example of this type of game I use the sequential Matching Pennies (SMP) game. This game is similar to the MP game solved in the previous model except that the first player plays first and the second player responds to player one's choice of action. The game tree to this game is shown in figure 5.2.

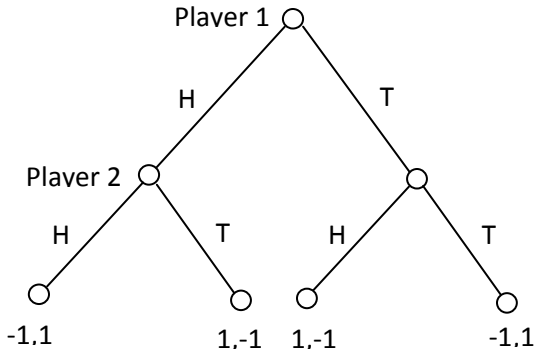


Figure 5.2: Sequential Matching Pennies Tree

The Programming and Logical Method

Preliminary Logic

Before I describe the method to coding this problem I must describe some preliminary logic. It is standard game theory notation to represent sequential games in a normal form matrix. This matrix lists all possible combinations of strategies and their associated outcomes. From this matrix the Nash Equilibria can be determined. However, these Nash equilibria are not always subgame perfect and so this notation does not work for this model. The normal form representation of the sequential Matching Pennies game is shown in figure 5.3.

		Player 2			
		H,H	H,T	T,H	T,T
Player 1	H	-1,1	-1,1	1,-1	1,-1
	T	1,-1	-1,1	1,-1	-1,1

Figure 5.3: SMP Normal Form Matrix

As an alternative to the normal form I've developed what I call an outcome-based approach to representing sequential games. In the outcome-based matrix the payoffs for both players are shown for

any combination of actions. As both players have two possible actions this matrix will contain four outcomes. The outcome-based matrix for this game is shown in figure 5.4. This matrix is very similar to the static game matrix; however, the sequential game is solved using backwards induction.

		Player 2	
		H	T
Player 1	H	-1,1	1,-1
	T	1,-1	-1,1

Figure 5.4: SMP Outcome-Based Matrix

Solving by Backwards Induction

Once the outcome-based matrix is known the logic to solving this problem is very similar to that used in solving the pure strategies of a static game. The first step in solving this problem is to list both players' outcome-based matrices. As in the static game, player one's matrix is named P1 and player two's is P2.

Next the sequential game is broken down into two subgames. Each subgame takes the first player's strategy as given and then maps the payoffs for both players into a matrix. The rows in this matrix represent player two's strategies and the columns represent the payoffs for player one and two respectively. An example of one of these subgames is shown in figure 5.5. In this subgame the first player chooses action H and the outcomes for both players are shown for both of player two's possible action choices. To see where this matrix comes from I've reproduced the SMP game tree in figure 5.6 and I've enclosed the first subgame in red brackets.

		Player 1	Player 2
Player 2	H	-1	1
	T	1	-1

Figure 5.5: SMP Subgame 1

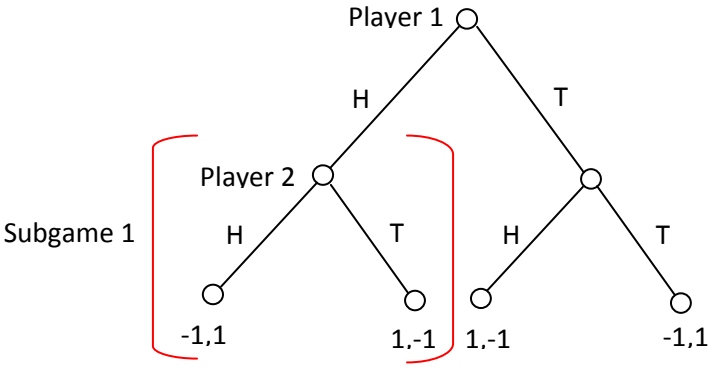


Figure 5.6: SMP Subgame 1

For each subgame an isolated payoff vector (SGIPV) is created for player two. These SGIPV provide a storage mechanism for determining player two's best responses. Player two will have two SGIPVs.

Player two's best action and the outcomes for both players must be determined for each subgame. Player one then uses the outcome information to decide what strategy to play. Player two's outcomes

are given by the best response value variables, but player one's outcomes must be calculated. Both players' outcomes are stored in the same vector, and there will be two outcome vectors for this problem. As is the game theory convention player one is listed first followed by player two.

With these outcome vectors an outcomes matrix is formed. This matrix takes player two's best actions and shows the payoffs for each player under either of player one's action choices. The rows contain player one's strategy options and the columns contain the payoffs for player one and two respectively.

Player one's best outcome strategy is determined by finding the maximum value in the first column. In this program I've accounted for the possibility of player one being indifferent between her action choices. That is, the outcome for player one in both circumstances may be identical. This addition was necessary to solve this version of SMP; however, the three-sequence model doesn't include this caveat.

The penultimate step is to determine the proper subgame perfect Nash Equilibrium(s) for the game. The proper SPNE is expressed as a two by two matrix. The first column contains player one's strategy and the second column contains player two's strategy. Player two always has two actions in her strategy and player one always has one action. Although, if player one is indifferent between her two actions she will have two possible strategies. In this case there are two SPNE. Player two's strategy may be read as follows: row one, column two represents his action if player one plays the first strategy and row two, column two represents his action if player one plays the second strategy.

The final step is to determine the equilibrium strategy vector. This vector represents the final outcome of the game. It is not the same as the subgame perfect equilibrium as it doesn't account for all possible contingencies of player two. Player one's strategy is given in the first column and player two's is in the second.

The Code

```
%Econ 353 Final Project - Nouri Najjar
%Game Theory Problem #3
%Subgame Perfect Nash Equilibrium: Two Player, Two Strategy Sequential Game

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 1: First state each player's outcome-based matrix.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

P1=[2 3;
    2 8];

P2=[6 1;
    1 9];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 2: The next step involves breaking the sequential game into two
%subgames.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Given player 1 plays first strategy
SG1=[P1(1,:)' P2(1,:)'];
```

```
%Given player 1 plays second strategy
SG2=[P1(2,:)' P2(2,:)'];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 3: Next, player two's subgame isolated payoff vectors (SGIPV's) are
%calculated.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Player two's IPV given player one plays strategy 1
SGIPV21=SG1(:,2);

%Player two's IPV given player one plays strategy 2
SGIPV22=SG2(:,2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 4: Player two's best action must now be determined given player 1's
%choice of strategy.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Player two's best action given player one plays strategy 1
[sgbr21 sgbri21]=max(SGIPV21);

%Player two's best action given player one plays strategy 2
[sgbr22 sgbri22]=max(SGIPV22);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 5: The outcomes for both players must be determined for both subgames.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Player one and two's outcomes given player one plays strategy 1
Outcome1=[P1(1,sgbri21) sgbr21];

%Player one and two's outcomes given player one plays strategy 2
Outcome2=[P1(2,sgbri22) sgbr22];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 6: With these outcome vectors an outcomes matrix is formed.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

OM=[Outcome1;
    Outcome2];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 7: Player one's best outcomes strategies are determined by finding the
%maximum values in the first row of the outcome matrix.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if (OM(1,1)==OM(2,1))%Determines if player one's best outcomes are unique
    [br1 bri1stor]=max(OM(:,1));
    bri1=[1 2];%Player one has two best responses if both return the same
value
elseif (OM(1,1)~=OM(2,1))
    [br1 bri1]=max(OM(:,1));%Otherwise player one has one best response
```

```
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Determines the proper subgame perfect nash equilibrium for this game.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[storsize1 storesize2]=size(bri1);

if (storesize2==2)%There are multiple best strategies for player one
    bri11=bri1(1,1);
    bri12=bri1(1,2);
    SPNE=[bri11 sgbri21;
          bri12 sgbri22];
elseif (storesize2==1)%There is a unique best strategy for player one
    SPNE=[bri1 sgbri21;
          0    sgbri22];
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%The equilibrium strategy vector is now calculated.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if (storesize2==2)
    EquilStrats=SPNEStrats;
elseif (storesize2==1)
    if (bri1==1)
       EquilStrats=[bri1 sgbri21];
    elseif (bri1==2)
       EquilStrats=[bri1 sgbri22];
    end;
end;

SPNE
EquilStrats
```

The Results

Solving this game by backwards induction yields two subgame perfect equilibria: player one is indifferent between heads and tails and player two will match whatever player one chooses. Therefore, there are two possible subgame perfect equilibrium outcomes: heads and heads or tails and tails. The results of the sequential Matching Pennies game are shown in figure 5.7 and are identical to the backwards induced solutions.

Sequential Matching Pennies Output

SPNE =

1	**1**
2	**2**

EquilStrats =

1	**1**
2	**2**

Figure 5.7

To test this model further I modify the sequential Matching Pennies game so that player one is no longer indifferent between her strategy choices. The game tree for this game is shown in figure 5.8.

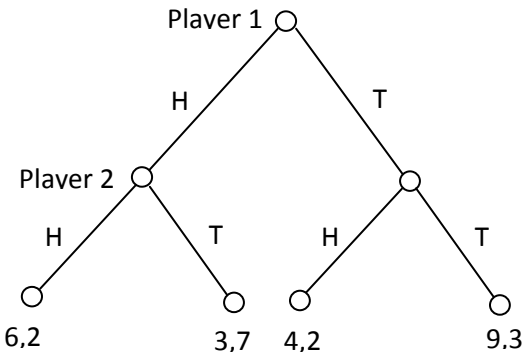


Figure 5.8: Modified SMP Game Tree

Solving this game by backwards induction finds that player two always chooses tails and with this knowledge player one chooses tails as well. Therefore, the SPNE is for player one to choose tails and player two to choose tails regardless of player one's action. That is also the SPNE found by the program, the output of which is shown in figure 5.9. The equilibrium outcome for this game is for both players to choose tails.

Sequential Game #2 Output

SPNE =

2 2

0 2

EquilStrats =

2 2

Figure 5.9

Model Four – Three-Sequence Subgame Perfect Nash Equilibrium

Description

By adding another level to the sequential game this model becomes much more complicated. In game tree form a three sequence game will have an extra node layer and twice as many potential outcomes. A three sequence Matching Pennies game tree is shown in figure 6.1.

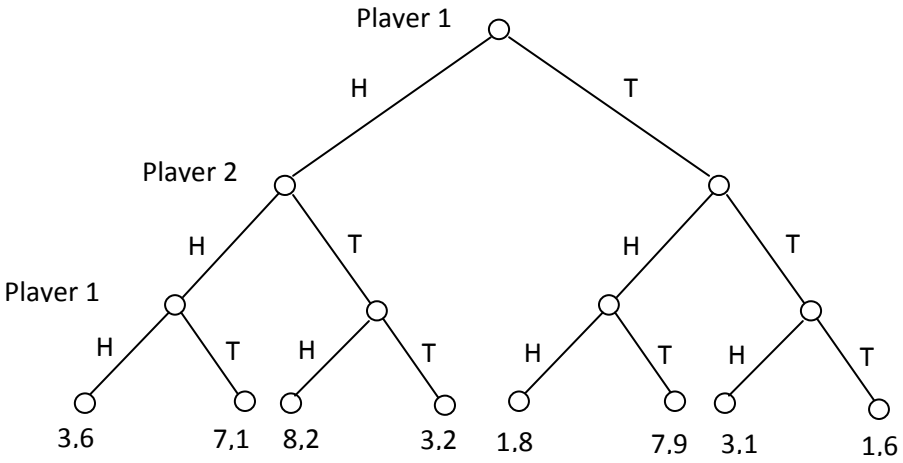


Figure 6.1: Three-Sequence SMP Game Tree

Representing three-sequence games in normal form is also much more complicated. The normal form for the three-sequence SMP game is shown in figure 6.2. According to this matrix there are four Nash Equilbria, however, solving this game by backwards induction has only one unique subgame perfect Nash Equilibrium. This highlights one of the problems with normal form representation and the Nash Equilibrium.

		Player 2			
		H,H	H,T	T,H	T,T
Player 1	H,H,H	3,6	3,6	8,2	8,2
	H,H,T	3,6	3,6	3,2	3,2
	H,T,H	7,1	7,1	8,2	8,2
	H,T,T	7,1	7,1	3,2	3,2
	T,H,H	1,8	3,1	1,8	3,1
	T,H,T	1,8	1,6	1,8	1,6
	T,T,H	7,9	3,1	7,9	3,1
	T,T,T	7,9	1,6	7,9	1,6

Figure 6.2: Three-Sequence SMP Normal Form Representation

The Programming and Logical Method

Preliminary Logic

As in the third model, I start solving this problem by presenting the game in its outcome-based form. In the three-sequence case the rows contain the first strategy of both player one and two and the columns contain player one's final action. The outcome based matrix contains the payoffs for both players under any combination of actions. The outcome based matrix for this game is shown in figure 6.3. In each payoff cell player one's payoff is listed first and player two's is second.

		Player One	
		H	T
Player One, Player Two	H,H	3,6	3,2
	H,T	8,2	3,2
	T,H	1,8	7,9
	T,T	3,1	1,6

Figure 6.3: Three-Sequence SMP Outcome Based Representation

Solving by Backwards Induction

As is customary with dynamic games, I solve this problem using backwards induction. The first step in coding this problem is to state each player's outcome-based matrix. The next step involves breaking the sequential game into subgames. Unlike the third model there are two sets of subgames. The first set takes both player's initial actions as given and then maps the payoffs for both players into a matrix. In this matrix the rows represent player one's strategies and the columns represent the payoffs for player one and two respectively.

Player one's subgame isolated payoff vectors (SGIPV's) are now calculated followed by her best response (2^{nd}) action.

Next, the outcomes for both players must be determined for all subgames. Player two then uses this information to decide on what strategy to play given player one's choice of initial action. Player one's outcomes are given by the best response value variables, but player two's outcomes must be calculated.

With these outcome vectors the first outcome matrix is formed. This matrix takes player one's best responses and shows the payoffs for each player under all strategy combinations. Rows give the combinations of player one and two's strategy options and columns give the payoffs for player one and two respectively. This matrix stores the outcomes of the game assuming player one's first play is fixed and player two's response is fixed. As this is a game of complete information and player two knows what player one's response to his action will be, player two is able to determine which action is best for him given player one's choice of initial action. To do this another set of subgames must be calculated. These subgames fix player one's initial action and then determine player one and two's payoffs for each of player two's available strategies (assuming player one's final move is individually rational and payoff maximizing).

Next, player two's subgame isolated payoff vectors (SGIPV's) are calculated followed by his best action. Each player's outcome must be determined for each of the second set of subgames, as was done with the initial set of subgames. These outcomes are crucial in determining the SPNE. Player two's outcomes are given by the best action value variables, but player one's must be calculated.

The second outcome matrix is now created. It stores the outcomes of the game given player one's initial action. The first row contains the outcomes for both players given player one plays action one and the second row contains the outcomes given player one plays action two. Player one's best initial action is determined by finding the maximum values in the first row of this outcome matrix.

Given the first player's best initial action, player two's best action can be calculated. Once both players' initial actions are known player one's final action can be determined. The proper SPNE can now also be determined. The SPNE is represented in matrix form, where the first column represents the play at the first node, the second column represents the play at the second node, and the third column represents the play at the third node. That means player one's best first action is given in the first column, player two's best action is given in the second column for all contingencies, and player one's final action is given in the third column for all contingencies. Player one only has one initial action, given in the first row of the matrix. Player two has two possible contingencies, where the first row of the matrix contains his choice given player one's initial action is one and row two contains his choice given player one's initial action is two. Player one has four possible final action contingencies, where the first row of the matrix contains her final action given her initial action is one and player two's action is one (1,1), the second row contains her final action given her initial action is one and player two's action is two (1,2), the third row contains her final action if (2,1) is played, and the fourth row contains her final action if (2,2) occurs.

Finally, the equilibrium strategy vector is calculated. This is the final outcome of the game. Player one's first action is given in the first column, player two's action is in the second column, and player one's final action is in the third column.

The Code

```
%Econ 353 Final Project - Nouri Najjar
%Game Theory Problem #4
%Subgame Perfect Nash Equilibrium: Two Player, Three Sequence Game

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 1: State each player's outcome-based matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

P1=[3 7;
    8 3;
    1 7;
    3 1];

P2=[6 1;
    2 2;
    8 9;
    1 6];
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 2: The next step involves breaking the sequential game into subgames.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Given P1,P2 = {1,1}
SG11=[P1(1,:)' P2(1,:)'];

%Given P1,P2 = {1,2}
SG12=[P1(2,:)' P2(2,:)'];

%Given P1,P2 = {2,1}
SG13=[P1(3,:)' P2(3,:)'];

%Given P1,P2 = {2,2}
SG14=[P1(4,:)' P2(4,:)'];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 3: Next, player one's subgame isolated payoff vectors (SGIPV's) are
%calculated.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Player one's IPV given P1,P2 = {1,1}
SGIPV11=SG11(:,1);

%Player one's IPV given P1,P2 = {1,2}
SGIPV12=SG12(:,1);

%Player one's IPV given P1,P2 = {2,1}
SGIPV13=SG13(:,1);

%Player one's IPV given P1,P2 = {2,2}
SGIPV14=SG14(:,1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 4: Player one's best action must now be determined, given player 1 and
2's choice of initial actions.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Player two's best action given P1,P2 = {1,1}
[sgbr11 sgbri11]=max(SGIPV11);

%Player two's best action given P1,P2 = {1,2}
[sgbr12 sgbri12]=max(SGIPV12);

%Player two's best action given P1,P2 = {2,1}
[sgbr13 sgbri13]=max(SGIPV13);

%Player two's best action given P1,P2 = {2,2}
[sgbr14 sgbri14]=max(SGIPV14);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 5: The outcomes for both players must be determined for all subgames.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%Player one and two's outcomes given P1,P2 = {1,1}
Outcome11=[sgbr11 P2(1,sgbri11)];

%Player one and two's outcomes given P1,P2 = {1,2}
Outcome12=[sgbr12 P2(2,sgbri12)];

%Player one and two's outcomes given P1,P2 = {2,1}
Outcome13=[sgbr13 P2(3,sgbri13)];

%Player one and two's outcomes given P1,P2 = {2,2}
Outcome14=[sgbr14 P2(4,sgbri14)];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 6: With these outcome vectors the first outcome matrix is formed.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

OM1=[Outcome11;
     Outcome12;
     Outcome13;
     Outcome14];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 7: Calculate the second set of subgames
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Given P1 plays action 1
SG21=OM1(1:2,:);

%Given P1 plays action 2
SG22=OM1(3:4,:);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 8: Next, player two's subgame isolated payoff vectors (SGIPV's) are
%calculated.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Player two's IPV given P1 plays 1
SGIPV21=SG21(:,2);

%Player one's IPV given P1 plays 2
SGIPV22=SG22(:,2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 9: Player two's best action must now be determined, given player 1's
%choice of initial action.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Player two's best action given P1 plays 1
[sgbr21 sgbri21]=max(SGIPV21);

%Player two's best action given P1 plays 2
[sgbr22 sgbri22]=max(SGIPV22);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 10: Each player's outcome must be determined for each subgame as was
done with
%the initial set of subgames.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Player one and two's outcomes given P1 plays 1
Outcome21=[SG21(sgbri21,1) sgbr21];

%Player one and two's outcomes given P1 plays 2
Outcome22=[SG22(sgbri22,1) sgbr22];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 11: The second outcome matrix is now created.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

OM2=[Outcome21;
     Outcome22];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 12: Player one's best initial action is determined by finding the
%maximum values in the first row of the outcome matrix.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[br11 bri11]=max(OM2(:,1));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 12: Given P1's best initial action is bri11, can now determine P2's
action.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for k=1:2;
    if (bri11==k)
        bri21=eval(sprintf('sgbri2%d',k));
    end;
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 13: Given P1's best initial action is bri11 and P2's best action is
bri21, can
%now determine P1's best response action.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if (bri11==1)
   if (bri21==1)
       bri12=sgbri11;
   elseif (bri21==2)
       bri12=sgbri12;
   end;
elseif (bri11==2)
    if (bri21==1)
        bri12=sgbri13;
    elseif (bri21==2)
        bri12=sgbri14;
    end;
```

```
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 14: The proper SPNE is now determined.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

SPNE=[bri11 sgbri21 sgbri11;
      0     sgbri22 sgbri12;
      0     0       sgbri13;
      0     0       sgbri14];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 15: The equilibrium strategy vector is now calculated. This is the
final outcome
%of the game.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

EquilStrats=[bri11 bri21 bri12];

SPNE
EquilStrats
```

The Results

Solving this game by backwards induction gives a unique subgame perfect Nash Equilibrium: player one's initial action is to play heads, player two's strategy is to play the opposite of whatever player one plays, and player one's final action is to play the opposite of whatever player two plays. The program's results for this game are shown in figure 6.4. The SPNE is identical to that found by backwards induction. The equilibrium outcome for this game is for player one to first play heads, player two to play tails, and player one to respond with heads.

Three-Sequence SMP Output

SPNE =

1	**2**	**2**
0	**1**	**1**
0	**0**	**2**
0	**0**	**1**

EquilStrats =

1	**2**	**1**

Figure 6.4

Conclusion

In this project I have reviewed some relatively simple yet fundamental game theory problems and concepts. To create these models I used both standard game theory concepts, such as dominance, backwards induction, and normal form representation, as well as some of my own logical methods, such

as the outcome-based representation for dynamic games. I feel these models are both logically robust and computationally complex, however, I must admit I spent a large amount of time attempting to complete one last model. Unfortunately, I couldn't finish this final model in its entirety as the size and complexity of it probably warranted an entire project unto itself. However, I still feel I should include the portion I have completed as it represents a computational progression of some of the concepts I've already introduced. A copy of the commented code with a brief description is included in Appendix A.

Appendix A: Model Five – The MxN Static Game

My goal with this game was to code a matlab program that would solve the Nash Equilibrium for any two player static game. That is, I wanted to write a program that would produce the equilibrium answer for any M by N game, where player one has M strategies to choose from and player two has N strategies to choose from. As I've already mentioned, I failed! But, I did come pretty close. Essentially, my program will solve for the Nash Equilibrium of any M by N game *provided one of the player's has a dominant strategy*. Basically, this occurs because I didn't finish writing the program. Without knowing the specific number of strategies available it is very complicated to complete the logic when no dominant strategies are present in the game.

I have attached the code below with comments explaining each step. The two most important programming features I use are:

- The for loop
  - This allows me to go through many iterations of the same logical loop.
- The sprintf function
  - This function allows me to add an index number to the end of a variable. For example the code:

```
for i=1:3;

        temp=i;
        eval([sprintf('test%d=temp;',i)]);
end;
```

    creates three variables; test1, test2 and test3. Where test1=1, test2=2, and test3=3. I found this very handy in combination with the for loop, and I use it heavily in my code when indexing or calling an indexed variable.

I created this program as a function, so it must be called from another matlab program. For this purpose I've included a program titled "finaltest.m" on the CD.

The Code

```
%Econ 353 Final Project - Nouri Najjar
%Game Theory Model #5
%Nash Equilibrium: two player, mxn game

function[nash]=nasheq(m,n,P1,P2)
%
%[nash]=nasheq(m,n,P1,P2)
```

```
%
%Computes the nash equilibrium for a two player mxn game
%
%Nouri Najjar
%March 12, 2009

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Clear the variables to be used in this code
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear j;
clear k;
clear t;
clear u;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 1: Find the best response functions for each player. Player 1 will have
n
%best responses and player 2 will have m best responses. The sprintf
%function indexes the bri1j variable through all values of j, creating and
%saving the best responses for player one given every j. It does the same
%for player two and indexes through k.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for j=1:n;
    [temp1 temp1i]=max(P1(:,j));
    eval([sprintf('bri1%d=temp1i;',j)]);
end;

for k=1:m;
    [temp2 temp2i]=max(P2(k,:));
    eval([sprintf('bri2%d=temp2i;',k)]);
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 2: Create best response storage vectors for both players that sum
%the index numbers associated with each best response. The average best
%response index number is then calculated. This number is used in
%determining dominated strategies.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

storbri1=0;

for j=1:n;
    storbri1 = storbri1 + eval(sprintf('bri1%d',j));
end;

meanbri1=storbri1/n; %Player 1's average best response index

storbri2=0;

for k=1:m;
    storbri2 = storbri2 + eval(sprintf('bri2%d',k));
end;
```

```
meanbri2=storbri2/m; %Player 2's average best response index

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Step 3: Find the Nash Equilibrium(s)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
nash=[];

if (bri11==meanbri1) %P1 has a strictly dominant strategy
    if (bri21==meanbri2) %P2 has a strictly dominant strategy
        nash=[bri11 bri21];
    elseif (bri21~=meanbri2) %P2 does NOT have a strictly dominant strategy
        for k=1:m;
            if (bri11==k)
                nash=[bri11 eval(sprintf('bri2%d',k))];
            end;
        end;
    end;
elseif (bri11~=meanbri1) %P1 does NOT have a strictly dominant strategy
    if (bri21==meanbri2) %P2 has a strictly dominant strategy
        for j=1:n;
            if (bri21==j)
                nash=[eval(sprintf('bri1%d',j)) bri21];
            end;
        end;
    elseif (bri21~=meanbri2) %P2 does NOT have a strictly dominant strategy
(no dominant strategies)
        %The rest of this code is interesting, however it is irrelevant as it
is unfinished.
        %I've included it merely to show the framework I created for
        %removing dominated strategies

        dominatedstratstorcalc1=0;%tracks which strategies are dominated for
P1
        dominatedstratstorcalc2=0;%tracks which strategies are dominated for
P2


        dominatedtemp1=0;
        for t=1:m; %Determine if P1 has a strictly dominated strategy
            dominatedstorage1=dominatedtemp1;
            for j=1:n;
                if (eval(sprintf('bri1%d',j))==t)
                    dominatedstorage1=dominatedstorage1 + 1;
                    eval([sprintf('dominated1%d=dominatedstorage1;',t)]);
                elseif (eval(sprintf('bri1%d',j))~=t)
                    eval([sprintf('dominated1%d=dominatedstorage1;',t)]);
                end;
            end;
        end;

        dominatedtemp2=0;
        for u=1:n; %Determine if P2 has a strictly dominated strategy
            dominatedstorage2=dominatedtemp2;
            for k=1:m;
                if (eval(sprintf('bri2%d',k))==u)
                    dominatedstorage2=dominatedstorage2 + 1;
```

```
                    eval([sprintf('dominated2%d=dominatedstorage2;',u)]);
                elseif (eval(sprintf('bri2%d',k))~=u)
                    eval([sprintf('dominated2%d=dominatedstorage2;',u)]);
                end;
            end;
        end;

        r=m-dominatedstratstorcalc1;
        c=n-dominatedstratstorcalc2;

        for s=1:r;
            P1subgame=P1;
            P2subgame=P2;
            dominatedstratstor1=0;
            if (eval([sprintf('dominated1%d',s)])==0)%If strategy s for
player 1 is dominated, remove from game and create subgame for both players
                P1subgame(s,:)=[];
                P2subgame(s,:)=[];
                dominatedstratstorcalc1=dominatedstratstorcalc1 + 1;
                dominatedstratstor1=dominatedstratstor1 + s;

eval([sprintf('dominatedstratstor1%d=dominatedstratstor1;',s)]);
            end;
        end;

        for t=1:c;
            %P1subgame=P1subgame;
            %P2subgame=P2subgame;
            dominatedstratstor2=0;
            if (eval([sprintf('dominated2%d',t)])==0)%If strategy t for
player 2 is dominated, remove from game and create subgame for both players
                P1subgame(:,t)=[];
                P2subgame(:,t)=[];
                dominatedstratstorcalc2=dominatedstratstorcalc2 + 1;
                dominatedstratstor2=dominatedstratstor2 + t;

eval([sprintf('dominatedstratstor2%d=dominatedstratstor2;',t)]);
            end;
        end;

        dominatedtest=dominatedstratstor1+dominatedstratstor2; %Stores if
there are dominated strategies for either player

        if (dominatedtest~=0)%Tests if there are dominated strategies for
either player
            [q,r]=size(P1subgame);
            nasheq(q,r,P1subgame,P2subgame);
        elseif (dominatedtest==0);%True if there are no dominated strategies
            nash=[0 0];
        end;

end;
```

```
end
```