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

Project Report

Axelrod's Tournament with Noise

Andermatt Samuel   Bösser Jonathan   Meier David

Zurich
Dec 2011

Agreement for free download

We hereby agree to make our source code for this project freely available for download from the web pages of the SOMS chair. Furthermore, we assure that all source code is written by ourselves and is not violating any copyright restrictions.

Andermatt Samuel

Bösser Jonathan

Meier David

Contents

1 Abstract **1**

2 Individual contributions **2**
- 2.1 Andermatt Samuel 2
- 2.2 Bösser Jonathan 2
- 2.3 Meier David 2

3 Introduction and Motivations **3**
- 3.1 The Prisoner's Dilemma 3
- 3.2 The Axelrod Experiment 3
- 3.3 Introduction of Noise 3

4 Description of the Model and Players **4**
- 4.1 Simple Players 4
  - 4.1.1 Cooperative Player 4
  - 4.1.2 Defective Player 4
  - 4.1.3 Random Player 4
- 4.2 Players from Literature 4
  - 4.2.1 Tit for Tat 4
  - 4.2.2 Friedmann 4
  - 4.2.3 Pavlov 4
  - 4.2.4 Tit for two Tat 5
  - 4.2.5 Joss 5
  - 4.2.6 Diekmann 5
  - 4.2.7 D-Downing 5
  - 4.2.8 C-Downing 5
- 4.3 Own Players 5
  - 4.3.1 Tit for Average Tat 5
  - 4.3.2 Watcher 6
  - 4.3.3 Reconciliation Tit for tat 6
  - 4.3.4 Tit for Tat with Reputation 6
  - 4.3.5 Strategy Switcher 6
  - 4.3.6 Evolutionary 6
  - 4.3.7 Limited Reconciliation Tit for tat 7
  - 4.3.8 Look Back D-Downing 7
  - 4.3.9 Look Back C-Downing 7
- 4.4 General view 7

5 Implementation 9

6 Results and Discussion 10
6.1 General Findings . . . 10
6.2 Problem caused by unreliably transmitted cooperation . . . 10
6.3 Benefit caused by unreliably transmitted defection . . . 10
6.4 Axelrods recommendations . . . 10
6.5 The Performance of each Player . . . 11
6.5.1 Cooperative Player . . . 11
6.5.2 Defective Player . . . 12
6.5.3 Random Player . . . 13
6.5.4 Tit For Tat . . . 13
6.5.5 Friedmann . . . 16
6.5.6 Pavlov . . . 17
6.5.7 Tit For 2 Tat . . . 18
6.5.8 Joss . . . 20
6.5.9 Diekmann . . . 22
6.5.10 Tit For Average Tat . . . 23
6.5.11 Reconciliation Tit for Tat . . . 24
6.5.12 CDowning and DDowning . . . 25
6.5.13 Tit For Tat with Reputation . . . 26
6.5.14 Strategy Switcher . . . 27
6.5.15 Lookback CDowning and Lookback DDowning . . . 30
6.5.16 Watcher . . . 31
6.5.17 Evolutionary . . . 32
6.5.18 Limited Reconciliation Tit For tat . . . 34
6.6 Impact of Noise on the whole system . . . 35
6.7 Comparison of the players . . . 35

7 Summary and Outlook 39

8 References 40

A Submitted Researchplan 41
A.1 General Introduction . . . 41
A.2 Fundamental Questions . . . 41
A.3 Expected Results . . . 41
A.4 References . . . 41
A.4.1 Research Methods . . . 42
A.5 Other . . . 42

B Matlabcode **43**
B.1 Master.m . . . 43
B.2 win.m . . . 45
B.3 show_data.m . . . 46
B.4 playerlist.m . . . 61
B.5 player1.m . . . 61
B.6 player2.m . . . 62
B.7 player3.m . . . 62
B.8 player4.m . . . 63
B.9 player5.m . . . 63
B.10 player6.m . . . 64
B.11 player7.m . . . 64
B.12 player8.m . . . 65
B.13 player9.m . . . 66
B.14 player10.m . . . 66
B.15 player11.m . . . 67
B.16 player12.m . . . 69
B.17 player13.m . . . 71
B.18 player14.m . . . 71
B.19 player15.m . . . 74
B.20 player16.m . . . 75
B.21 player17.m . . . 77
B.22 player18.m . . . 79
B.23 player19.m . . . 80
B.24 player20.m . . . 83

Matlabcode

1 Master.m 43
2 win.m 45
3 show_data.m 46
4 playerlist.m 61
5 player1.m 62
6 player2.m 62
7 player3.m 62
8 player4.m 63
9 player5.m 63
10 player6.m 64
11 player7.m 65
12 player8.m 65
13 player9.m 66
14 player10.m 67
15 player11.m 67
16 player12.m 69
17 player13.m 71
18 player14.m 71
19 player15.m 74
20 player16.m 75
21 player17.m 77
22 player18.m 79
23 player19.m 80
24 player20.m 83

List of Figures

1 Reward plot of the Cooperative Player 12
2 Reward plot of the Player Defect 13
3 Reward plot of the Random Player 14
4 Reward plot of the Player Tit For Tat 15
5 Reward plot of the Player Friedmann 16
6 Reward plot of the Player pavlov 18
7 Reward plot of the Player Tit For 2 Tat 19
8 Reward plot of the Player Joss 20
9 Reward plot of the Player Diekmann 22
10 Reward plot of the Player Tit For Average Tat 23

11 Reward plot of the Player Reconciliation Tit For Tat 24
12 Reward plot of the Player CDowning 26
13 Reward plot of the Player DDowning 27
14 Reward plot of the Player Tit For Tat with Reputation 28
15 Reward plot of the Player Strategy Switcher 29
16 Reward plot of the Player Lookback CDowning 30
17 Reward plot of the Player Lookback DDowning 31
18 Reward plot of the Player Watcher 32
19 Reward plot of the Player Evolutionary 33
20 Reward plot of the Player Limited Reconcilation Tit For tat 34
21 Reward plot of the Player Limited Reconcilation Tit For tat 36
22 Reward plot of the Player Limited Reconcilation Tit For tat 37
23 Reward versus Noise with the best players 37
24 Reward of players with both noises equals to zero 38

List of Tables

1 General view of all the players and their characteristics 8
2 Reward Matrix 9
3 Performance of TFT playing against TFT 15
4 Cooperation of TFT depending on the Noise 16
5 Cooperation of Friedmann dependant on the Noise 17
6 Cooperation of Pavlov depending on the Noise 18
7 Cooperation of TF2T depending on the Noise 19
8 Cooperation of Joss depending on the Noise 20
9 Cooperation of Diekmann depending on the noise 22
10 Cooperation of Tit For Average Tat depending on the noise 24
11 Cooperation of Reconcilation Tit For Tat depending on the noise . . 25
12 Cooperations of CDowning depending on the Noise 26
13 Cooperations of CDowning depending on the Noise 27
14 Cooperations of Tit For Tat with Reputation depending on the Noise 28
15 Cooperations of Strategy Switcher depending on the Noise 29
16 Cooperations of the Lookback CDowning depending on the Noise . . 31
17 Cooperations of the Lookback DDowning depending on the Noise . . 32
18 Cooperations of the Watcher depending on the Noise 33
19 Cooperations of the Evolutionary depending on the Noise 34
20 Cooperations of the Evolutionary depending on the Noise 35
21 Comparison of the two players Limited Tit Fot Tat and Tit For Tat . 35

1 Abstract

2 Individual contributions

2.1 Andermatt Samuel

- Further development of the game master
- Development and implemention of multiple players
- Data analysis and interpretation
- Contributions to the report
- Contributed experience in game theory

2.2 Bösser Jonathan

- Explore and explain GitHub [www.github.com]
- Development and implementation of multiple players
- Data analysis and interpretation
- Contributions to the report
- Literature study

2.3 Meier David

- Write the first version of the program
- Development and implementation of multiple players
- Responsible for the report
- Literature study

3 Introduction and Motivations

3.1 The Prisoner's Dilemma

The prisoner's dilemma is a model from game theory. 2 people are suspected to have done a crime together. Now they are examined separately in different rooms. In this situation, they can either whistle-blowing the other person to protect oneself or keep silent. Over all, it is of advantage, if both keep silent. But for the single person it is better to betray the other person. The risk of betraying is the following: if both accused people betray the other, the penalty for both is the highest. This problem is in game theory called "Prisoner's dilemma" [2].

3.2 The Axelrod Experiment

In the year 1981, Robert Axelrod invited for a competition to the iterated prisoner's dilemma. People from different fields like mathematics, politics, economy or psychology have been asked to develop a winning strategy for this competition. All the different strategies were playing against another to find the most successive strategy. Interestingly, the very simple strategy "Tit for Tat" (TFT) won the tournament. During the first round, TFT keeps silent (cooperation) and during the rest of the game, just does, what its counter player did the round before. This sort of experiment is very interesting, because the results can be applied in many different fields in real life. Just one out of many examples: 2 countries make an agreement on their amount of weapons. For the single country it is of advantage to haves more wmilitary strenth than the other nation. But as in the prisoner's dilema, if both nations rise their military strenth, for both it is just a loss money and an increase in danger. [3]

3.3 Introduction of Noise

A further development in the Axelrod Experiment is the introduction of noise. This means, cooperation is wrongly understood as defection and vise versa. The introduction of noise to the axelrod experiment ist nothing new, but very important, because in real world, noise and small distortions is always present. This can lead to serious complications. Just as an example: *"On September 1,1983 a South Korean airliner mistakenly flew over the Soviet Union (Hersh 1989). It was shot down by the Soviets, killing all 269 people aboard. The Americans and Soviets echoed their anger at each other in a short, but sharp escalation of cold war tensions."*[7]

4 Description of the Model and Players

4.1 Simple Players

4.1.1 Cooperative Player

The player 1 is a very simple player: He always cooperates. This "decision" does not depend on any circumstances like the decisions of its antagonist.

4.1.2 Defective Player

Also the player 2 is a very simple player: He always defects.

4.1.3 Random Player

Like all players from this subsection, the decision of the random player does not depend on the results of the previous tournaments. The decision is randomly distributed and no decision is preferred.

4.2 Players from Literature

All players in this subsection are taken from the first Axelrod's Tournament and implemented by us. Source: Lecture "Game Theory" [1]

4.2.1 Tit for Tat

The Player 4 during to the Axelrod Turnament the most successive player of all[1]. The decision is the decision of the counter player from the last tournament. In the first round, the decision is cooperation. If the counter player cooperated during the last round, this player will cooperate in the current round.

4.2.2 Friedmann

The Player "Friedmann" cooperates until its counter player defects once. After that, Friedmann now deflects for the rest of the game. This corresponds to "everlasting death".

4.2.3 Pavlov

Pavlov changes its decision every time when the counter player defects. But if the counter player cooperates, Pavlov gives the same decision as in the round before. The first decision is cooperation.

4.2.4 Tit for two Tat

The first decision is cooperation. If the counter player cooperates, Tit for 2Tat" cooperates as well. Tit for 2 Tat only defects, if the counter player defected the last 2 rounds.

4.2.5 Joss

This is basically the same player like the player "Tit for Tat". The only difference: 10% of the cooperative decisions are randomly defected. www.socio.ethz.ch/vlib/pesb/pesb9.pdf

4.2.6 Diekmann

The player "Diekmann" plays basically Tit for Tat. The difference is, that every 10th move, he playes cooperative twice. www.socio.ethz.ch/vlib/pesb/pesb9.pdf

4.2.7 D-Downing

Starts always with defection and is calculating afterwards the expected value of the reward if he cooperates or defects. The decision is taken based on the bigger expected value.

4.2.8 C-Downing

Same algorithm as D-Downing, but is starting with cooperation. This algorithm is better than D-Downing, actually C-Downing would have won the tournament of Axelrod [?], but there was only D-Downing implemented.

4.3 Own Players

4.3.1 Tit for Average Tat

Based on the idea "Tit for Tat", we developed a player who averages the decisions of its opponent over the most recent Rounds. The first few rounds he plays Tit for Tat. Then he starts averaging over the most recent rounds and reacts to the opponents most frequent decision. After a fixed number of rounds, the player restarts from the very beginning. This can prevent beeing stuck in mutual defection.

4.3.2 Watcher

For this player, we investigated one possible concept for a learning algorithm. The idea is to learn by observing and copy the moves of the most succesful player. During the first few rounds, Watcher plays Tit for tat.

4.3.3 Reconciliation Tit for tat

TFT has a disadvantage. Once the players start a mutual defection it is stable. This makes TFT very susceptible to miscommunications and performs poorly against players like Joss. The approach here is to break this cycle by adding cooperative moves. The risk of adding cooperative moves is that the opponents exploit this strategy. This strategy tries to make these moves without becoming exploitable. In case the opponent defects, his recent performance gets calculated. It is also calculated, how good the opponent would have performed if both players were cooperating. In case the damage by the mutual defections is large enough that by defecting the reconciliation attempt he cannot gain enough to outperform cooperation. This way the strategy is not explotaible.

4.3.4 Tit for Tat with Reputation

This is a further Tit for Tat mutant. The basic strategy remains the same, but the opponents moves against other players are also observed. In case the opponent is mostly cooperative against others, then defection of the oponent is regarded as miscommunication and interpreted as cooperation.

4.3.5 Strategy Switcher

The strategy switcher is another example for a learning player. The player is equipped with a set of predefined strategies. In our case we chose the strategies TFT, TF2T, Pavlov, always cooperate and always defect. Initially he tries out all five strategies. After he tried out every strategy he calculates each strategies performance. In the subsequent turns he always plays the most successful strategy. After a given set of turns he will reevaluate the performance of the current strategy and compare it with the others. If one of its other strategies has a higher performance this strategy is chosen instead.

4.3.6 Evolutionary

This player tries to find the optimal sequence of moves by an evolutionary algorithm. The strategy of the player consists of a given set of moves. To determine the first set of moves he plays TFT in the first rounds. Once he has a sequence of moves he creates clones of this sequence and adds mutations to them. A mutation means that the decision in one move is altered. In the next step he plays each clone. After all clones are played he evaluates their performance. The clones are split into segments, for each segment the winnings are calculated. The performance includes the one move after the segment ended, otherwise the players would always reject in the last move. Then a new parent strategy is formed. Each segment is evaluated, if the first segment of a clone performed stronger then the parent strategy, the parents segment is replaced by the more successful segment. This parent is then played and cloned again. There is an assumption that with increasing simulation length the sequence becomes closer to the optimal sequence. At this point the mutations become a disadvantage. Therefore the mutability is lowered with time (but does not go to zero).

4.3.7 Limited Reconciliation Tit for tat

This strategy is similar to the Reconciliation Tit for tat. In this case however the number of reconciliation attempts is limited. Some players tend to reject all reconciliation attempts and while this does not exploit this player, it still limits its performance. This player will stop to try to reconcile after he was unsuccessful doing so for a given time. However, if the opponent has two consecutive cooperative rounds the counter for the reconciliation attempts is reseted.

4.3.8 Look Back D-Downing

Is basically the same as D-Downing, but this one is looking back two rounds on his own decisions and just one round on the opponent's decisions. To be able to look back two rounds the player always defects the first two rounds. The idea behind this is, to see the reaction of the opponent on the decisions the player took before. By means of this the player has an advantage (in theory) compared to D-Downing, because the player is looking at two decisions, whereof one is the action (own decision) and the other (opponent's decision) is the reaction.

4.3.9 Look Back C-Downing

Same algorithm as Look Back D-Downing, but is starting with two times cooperation.

4.4 General view

In table 1 every player is listed and some of theirs characteristics are evaluated.

	Cooperates in First Round	responsive	Memory	Exploitable	Can Exploit	Global view	Learning
Cooperative Player	×		0	×			
Defective Player			0		×		
Random Player	random		0		×		
Tit for Tat	×	×	1				
Friedmann	×	×	inf		×		
Pavlov	×	×	1				
Tit for two Tat	×	×	2	×			
Joss	90%	×	1		×		
Diekmann	×	×	1	×			
D-Downing		×	inf	×	×		(×)
C-Downing	×	×	inf	×	×		(×)
Tit for Average Tat	×	slow	5	×			
Watcher	×		6	×	×	×	×
Recon Tit for Tat	×	×	20				
TfT with Reputation	×	×	1	×		×	
Strategy Switcher	×	×	inf	×	×		×
Evolutionary	×	very slow	31	×	×		×
Lim. Recon. TFT	×	×	20				
Look back D-Downing		×	inf	×	×		(×)
Look back C-Downing	×	×	inf	×	×		(×)

Table 1: General view of all the players and their characteristics

5 Implementation

As described in the Introduction, the tournament is a repeated prisoners dilemma. the payoff matrix for this game is shown in figure ....

. To make the simulation more realistc, we added noise to it. Noise means that defection can be transmitted as cooperation and vice versa. The noise applied on cooperation and defection was varied independantly. The two noiselevels are set independantly to 0, 5, 10 and 15%. The noise only changed the information the players recieved, but not their payoff. For each combination of noise, we performed a tournament with 20000 rounds. In each round, every player plays agains all others and himself. To make the decisions, the player is provided with all the decisions made in the previsous rounds by all players. The playersdo not have information about the noise level.

The basic structure of a player is shown .... . And an example (TFT) of such a player would be:

In the following table 1 is shown the payoff matrix applied in our program.

Table 2: Reward Matrix

	Player B cooperates	Player B defects
Player A cooperates	A:3 B:3	A:0 B:5
Player A defects	A:5 B:0	A:1 B:1

- Spielablauf
- Informationen, die die Spieler sehen knnten
- Art des Noises
-

6 Results and Discussion

6.1 General Findings

- A noise that interprets cooperation as defection decreases cooperation drastically.
- A noise that interprets defections as cooperations increases cooperation, but the effect is weaker.
- Perfect information is not whats best for the system. If decisions are transmitted better than they are, the whole system gets more efficient.
- Friendly/cooperative players performance drastically decreases if the chance that cooperation is transmitted correct is less than 100%.
- Players that do not react immediately to defections look non-responsive.

6.2 Problem caused by unreliably transmitted cooperation

Many of the friendly players perform well without noise, because they have an infinitely long sequence of mutual cooperations with other friendly players. Noise will trigger defections. This state requires then a way to come back into cooperation. Most cooperative players do not have a mechanism to reestablish cooperation if it has been destroyed, because they rely on a cooperation caused by the first turn decision being cooperative.

6.3 Benefit caused by unreliably transmitted defection

Some players try out defective moves. For TFT mutants this can likely result in mutual defections. A noise that inserts cooperative moves results in the TFT player reacting cooperative again, driving the game in mutual cooperation again. Another thing is that hiding defections allows aggresive players to exploit players that would retaliate otherwise, and an aggresive player exploiting a weak one is better than if both players are defecting each other.

6.4 Axelrods recommendations

Axelrod proposed a certain behavior to be successful. This behavior is: Be Nice, Retaliatory, Forgiving and Clear. Under noise the opponent will see defections from a player, even if he never defected. This diminishes the use of being nice. It is more successful to find out if he responds to defections and exploit the opponent if he is exploitable. In Axelrods tournament this was not the case, because this attempt might have long lasting effects. However noise somewhat covers up the past. The other recommendations still hold.

6.5 The Performance of each Player

Definition (Noise).

- Nose 1 = Chance, that a cooperation gets received as defection
- Nose 2 = Chance, that a defection gets received as cooperation

The performance plots shows the average reward of a cerctain player, depending on the two noise levels. The simulation was run twice, therefor, each player has 2 performance graphs.

For most players there are tables that show how many of their moves are cooperative, dependant of the Noise levels under which the simulation was run. In this tables the entries in the first row have no Noise1 and the lower the row is the higher is Noise1. Noise2 is zero on the left side and gets higher for entries more on the right side. So the Noise 1 goes from 0 to 0.15 from top to bottom, and Noise2 goes from 0 to 0.15 from left to right.

6.5.1 Cooperative Player

The cooperative player's performance in both simulations is shown in figure 1.

In a situation of no noise, this player performs strong against mostly friendly players, but gets exploited by aggressive players. In this situation it performs better than always defect, but similar to random.

Because this player is cooperative and not reactive Noise2 doesn't matter. On the other hand performance drastically decreases with Noise1. The seemingly inserted rejections make other players explore defective moves. Because the defective moves are not retaliated the other players might then stick with these defective moves. Players that start exploiting this player are Friedmann, Pavlov, CDowning and LookBack CDowning.

In a situation with noise this strategy is still strong with TFT mutants, because a cooperative interaction gets restored in the fastest possible way.

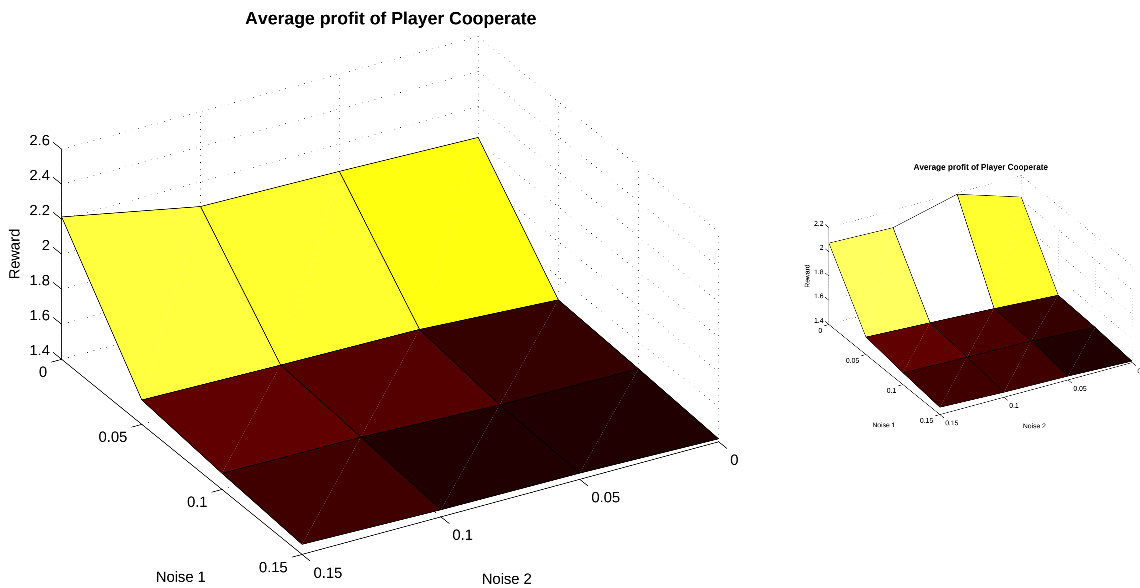


Figure 1: Reward plot of the Cooperative Player

Still with a drop of 0.7 to 0.8 in performance this player is one of the players that is the most susceptible to noise overall.

Traits of the player:

+ Can sustain cooperation with friendly players even in noise

- Exploitable

- Does not respond to the opponents move

6.5.2 Defective Player

The player's performance in both simulations is shown in figure 2.

The player is so unfriendly, that Noise2 help him. A move where he can exploit the opponent gives him 5 times the reward of mutual defections, therefore even a small number of exploiting moves helps him a lot. If we would run the simulation with a Noise close to 50% this player would become the most successful player. The performance increases in general against TFT mutants, but especially against players that try hard to avoid mutual defections, such as TF2T, reconciliation TFT, limited Reconciliation TFT. The strongest rise in performance comes against the player

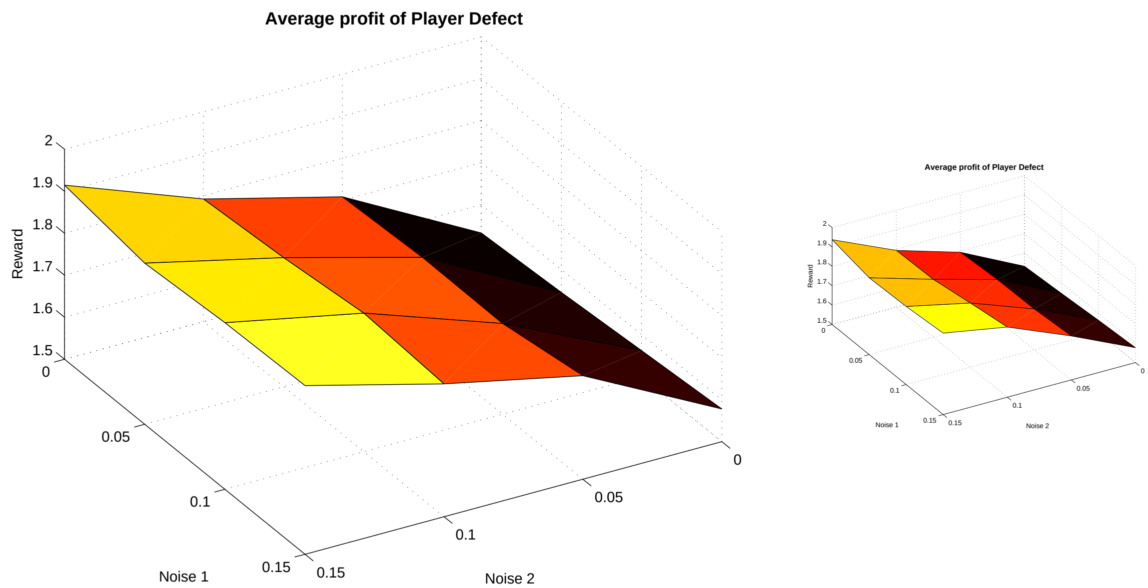


Figure 2: Reward plot of the Player Defect

Evolutionary.

Traits of the player:

+ Can exploit players that do not respond

+ Performance increase with Noise

- Ends up in mutual defections with most players

6.5.3 Random Player

The player's performance in both simulations is shown in figure 3.

The player does not get influenced that much by noise. There is a strange peak at zero noise, we were unable to find out why. If the noise makes his moves appear a little more cooperate there is a slight increase in performance, most likely due to more cooperative reactions from TFT mutants.

6.5.4 Tit For Tat

The player's performance in both simulations is shown in figure 4.

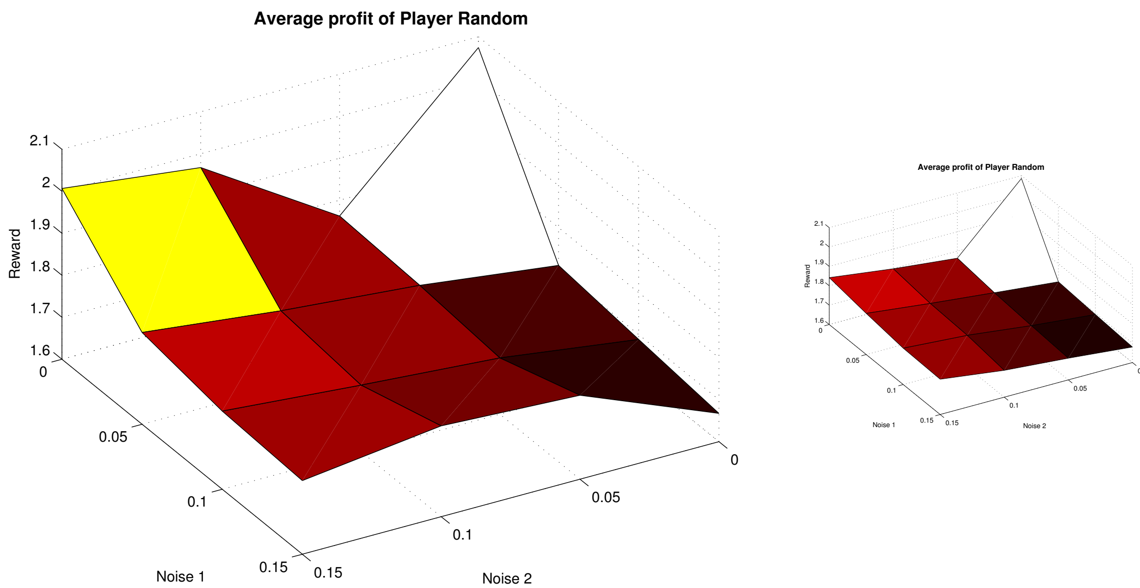


Figure 3: Reward plot of the Random Player

Tit for Tat (TFT) is a rather strong player. He won Axelrod's tournaments that were without noise. For Noise1=0 he is one of the strongest of all players. These graphs above show that this player is extremely susceptible to noise1. The reason is that with no noise 1 he will always cooperate with other TFT mutants because the first move was cooperative. With noise the first move maters less. There are basically three states he can enter with himself.

State 1: Mutual cooperation. Performance 3

State 2: Alternating defection and cooperation. Performance 2.5

State 3: Mutual defection. Performance 1

A Noise1 signal changes the state from state 1 to state 2 or state 2 to state 3. Noise2 works in the other direction. The two noises are basically transition probabilities. For Noise1 greater than zero and Noise2 equal to zero the TFT ends up stuck in state three with itself. For equal noises in both directions TFT should be 50% of the time in state 2 and 25% in the states 1 and 3. This would imply the performance: $0.25 * 3 + 0.5 * 2.5 + 0.25 * 1 = 2.25$

Table 3 shows the performance of TFT against itself. Because the players in our simulation could only perform mirrored decisions state 2 was impossible and the actual performance was somewhat lower.

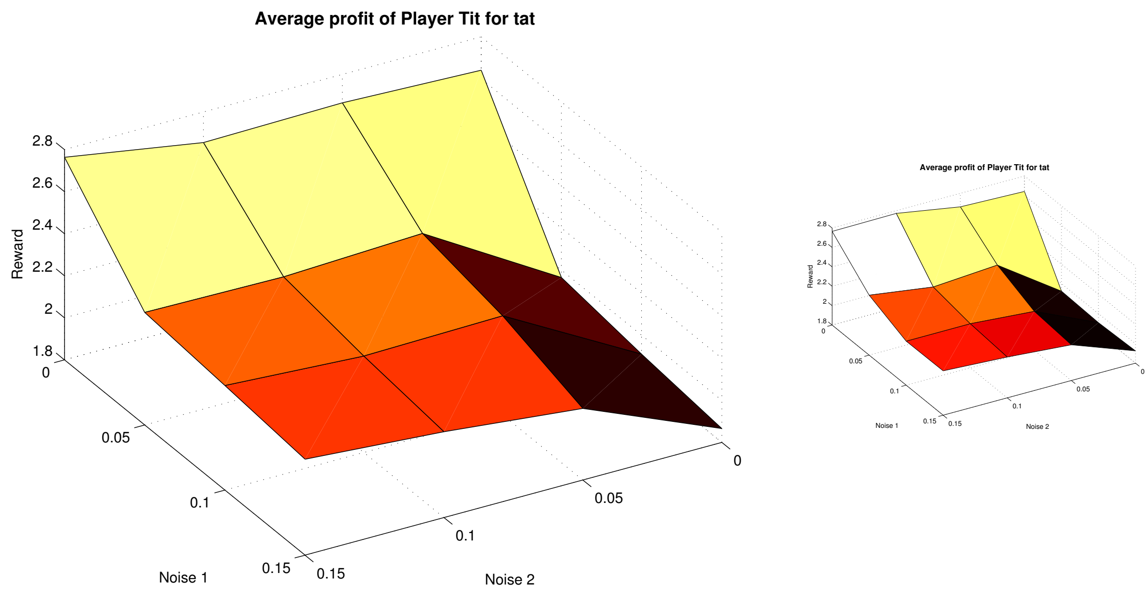


Figure 4: Reward plot of the Player Tit For Tat

Table 3: Performance of TFT playing against TFT

	Noise 2 = 0	Noise 2 = 0.05	Noise 2 = 0.1	Noise 2 = 0.15
Noise 1 = 0	3.0000	3.0000	3.0000	3.0000
Noise 1 = 0.05	1.0016	2.0127	2.3438	2..4745
Noise 1 = 0.10	1.0018	1.6184	1.9847	2.1967
Noise 1 = 0.15	1.0011	1.5170	1.7666	2.0086

Already at Noise1=0.05 the number of cooperative moves TFT performs drops from 40 to 80 percent. At higher Noise2 with no Noise1 the number of defections performed by TFT halves.

Traits of the player:

+ Responds fast

+ Not exploitable

+ Forgiving

- Only accepts an apology, but does not initiate it himself, so he can stuck in mutual defections!

Table 4: Cooperation of TFT depending on the Noise

	Noise 2 = 0	Noise 2 = 0.05	Noise 2 = 0.1	Noise 2 = 0.15
Noise 1 = 0	0.8075	0.8247	0.8351	0.8917
Noise 1 = 0.05	0.3958	0.5967	0.6192	0.6516
Noise 1 = 0.10	0.3418	0.5127	0.5428	0.5866
Noise 1 = 0.15	0.2886	0.4240	0.4836	0.5297

6.5.5 Friedmann

The player's performance in both simulations is shown in figure 5.

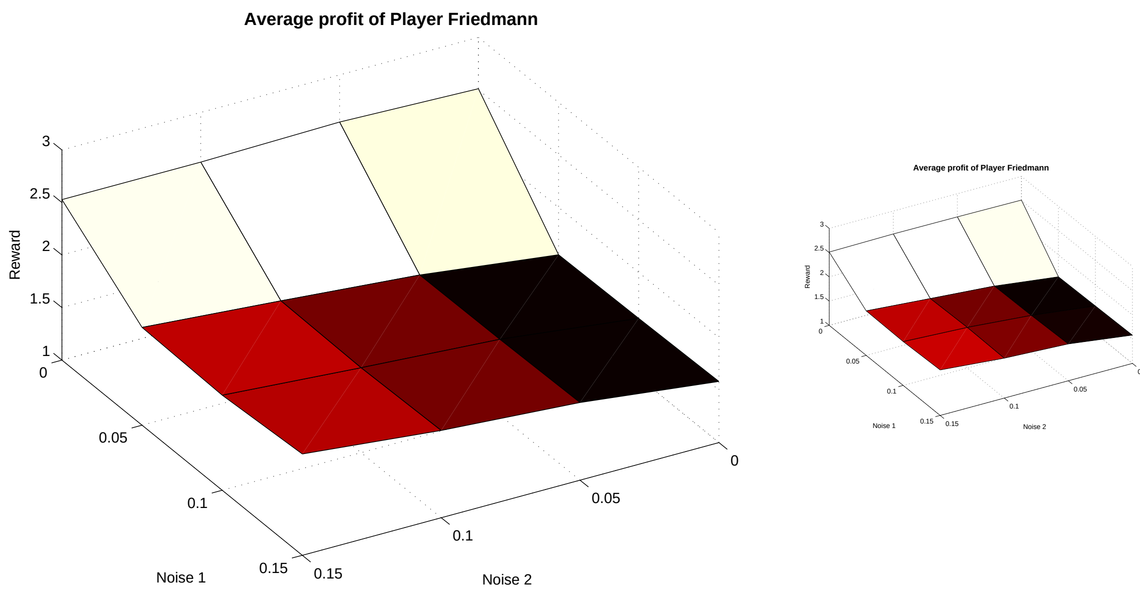


Figure 5: Reward plot of the Player Friedmann

In the case of perfect information this player can profit from mutual cooperation with many players. In this case he performs well. However for any Noise1 greater than zero the player will receive a rejection at some point and therefore act like the player "Defect" most of the time. Friedmann generally tries to retaliate so hard on a rejection that the other player will not even attempt one rejection. However Friedmann cannot capitalise on this deterrent effect, because the moment the opponent realises it is already too late and he cannot know it in advance.

Table 5: Cooperation of Friedmann dependant on the Noise

	Noise 2 = 0	Noise 2 = 0.05	Noise 2 = 0.1	Noise 2 = 0.15
Noise 1 = 0	0.6002	0.6000	0.6000	0.6000
Noise 1 = 0.05	0.0004	0.0001	0.0001	0.0001
Noise 1 = 0.10	0.0001	0.0001	0.0001	0.00016
Noise 1 = 0.15	0.0001	0.0001	0.0001	0.0001

As soon as Noise1 is greater than 0 the number of cooperative moves goes to zero. But already at zero noise Friedmann only has 60% cooperative moves, while TFT has 80%.

Traits of the player:

+ Stays in mutual cooperation with friendly players when Noise1 is zero

- Completely breaks down with Noise1 greater than zero

6.5.6 Pavlov

The player's performance in both simulations is shown in figure 6.

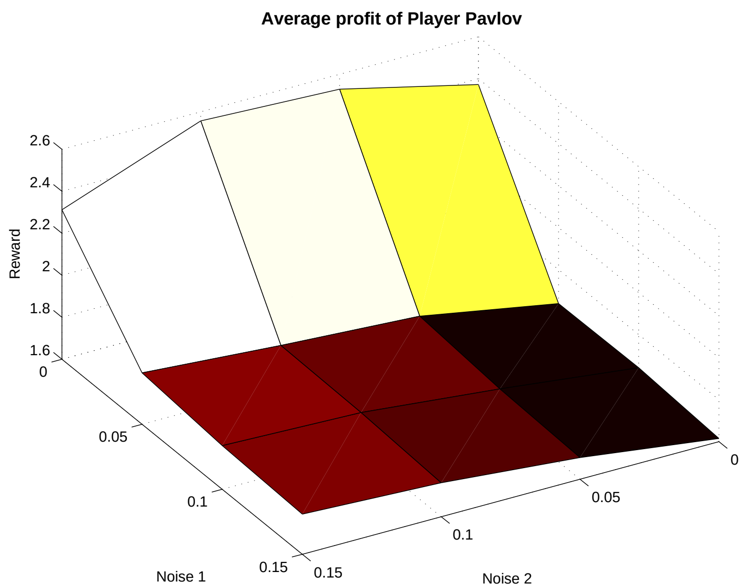
Pavlov performs rather well without Noise1. If Noise1 is greater then zero, some players realise that defections against Pavlov work as well as cooperations. Pure defect against Pavlov results in the rewards 5 1 5 1 5 1 for the opponent, while pure cooperation results in 3 3 3 3, given Pavlov is cooperative when the opponent started playing random. With zero noise the opponent gets an average reward of 3 for cooperation and defection but with noise the performance of cooperation decreases (because Pavlov retaliates). Players that largely start to defect against Pavlov are: Friedmann, Tit for average Tat, CDowning, LookBack CDowning and the Strategy switcher.

Without Noise 1 the number of Cooperative moves is around 80% while it is around 50% with Noise 1. It is interessting that with Noise2 but no Noise1 Pavlov is less cooperative than TFT.

Traits of the player:

+ Forgives fast

+ Initializes cooperation out of mutual defection



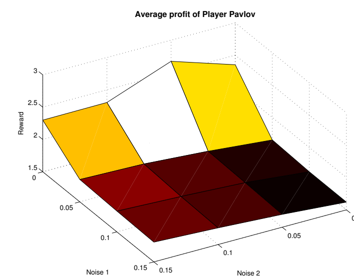


Figure 6: Reward plot of the Player pavlov

Table 6: Cooperation of Pavlov depending on the Noise

	Noise 2 = 0	Noise 2 = 0.05	Noise 2 = 0.1	Noise 2 = 0.15
Noise 1 = 0	0.8239	0.8488	0.8541	0.8037
Noise 1 = 0.05	0.4885	0.5433	0.5697	0.5860
Noise 1 = 0.10	0.4939	0.5211	0.5338	0.5455
Noise 1 = 0.15	0.5063	0.5187	0.5251	0.5319

+ Not exploitable without noise

- Too many cooperative moves against defecting players

- Not retaliating (pure defect and pure cooperation perform equally well)

6.5.7 Tit For 2 Tat

The player's performance in both simulations is shown in the two figures below:

As a TFT mutant it has a similar performance. The difference is that this player is more forgiving. The bad thing is that this makes him exploitable. The better thing is that it is more robust to Noise1 as it does not react to single defections. The player still ends up in mutual defections with itself if Noise2 is zero and Noise1 greater than

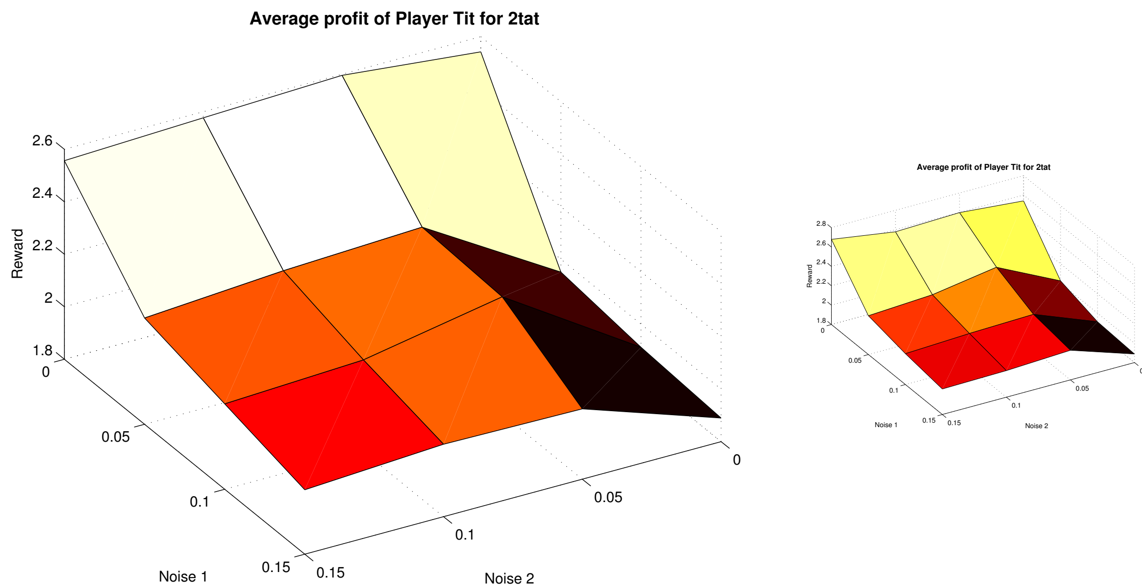


Figure 7: Reward plot of the Player Tit For 2 Tat

zero, but for both Noises greater than zero the player plays much stronger against itself. At zero noise the performance of TF2T is worse than the one of TFT, but with Noise1 their performances are similar. The effect that players like "Evolutionary" will exploit TF2T seems to balance out with the better resistance to noise.

Table 7: Cooperation of TF2T depending on the Noise

	Noise 2 = 0	Noise 2 = 0.05	Noise 2 = 0.1	Noise 2 = 0.15
Noise 1 = 0	0.8271	0.8796	0.8880	0.8947
Noise 1 = 0.05	0.5366	0.7450	0.7712	0.7926
Noise 1 = 0.10	0.5132	0.7349	0.7375	0.7622
Noise 1 = 0.15	0.4864	0.6503	0.6977	0.72749

The cooperation stays much higher than for TFT especially if Noise2 also is not zero.

Traits different to TFT:

+ More Forgiving

- More Exploitable

6.5.8 Joss

The player's performance in both simulations is shown in the two figures below:

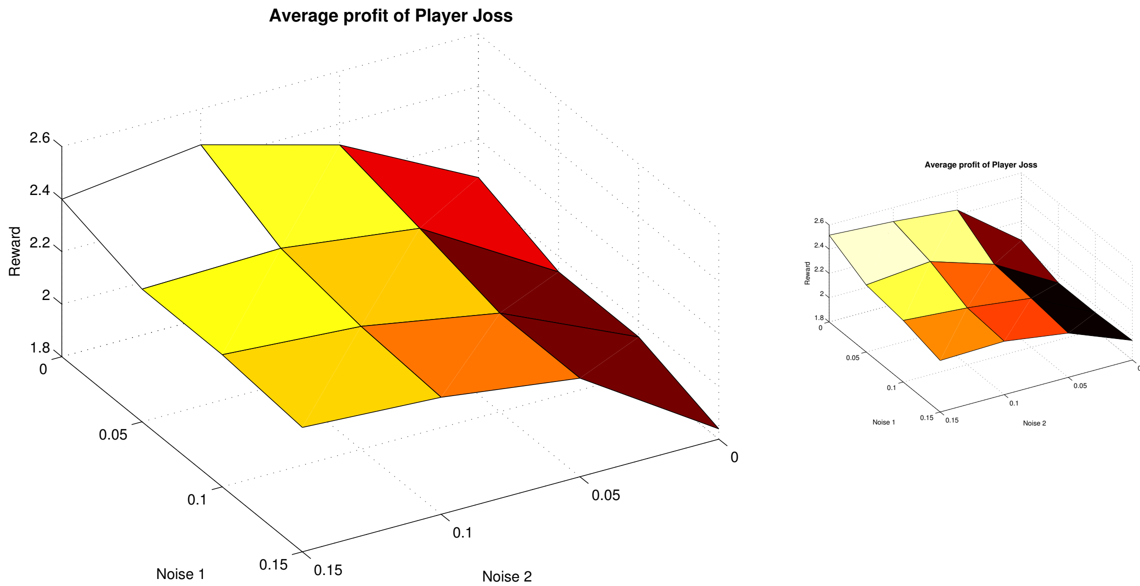


Figure 8: Reward plot of the Player Joss

While this is a TFT mutant, its performance dependence on noise looks totally different. The player generally performs poorly. At some noises this player is able to exploit TF2T (Noise2 greater than 0 and Noise1 0) however most of the time the retaliation of the defections outweighs their gain. TFT is very susceptible to Noise1 and Joss makes himself look like under Noise1 to his opponent. It is interesting that the performance of this player looks like the performance of "Defect" just shifted about 0.5 upwards.

Table 8: Cooperation of Joss depending on the Noise

	Noise 2 = 0	Noise 2 = 0.05	Noise 2 = 0.1	Noise 2 = 0.15
Noise 1 = 0	0.3983	0.5300	0.6102	0.6172
Noise 1 = 0.05	0.3983	0.5300	0.6102	0.6172
Noise 1 = 0.10	0.2968	0.4026	0.4625	0.5006
Noise 1 = 0.15	0.2404	0.3720	0.4225	0.4591

With higher Noise2 some cooperation can be achieved, but generally the player is mostly defecting.

Traits different to TFT:

- initates defections

6.5.9 Diekmann

The player's performance in both simulations is shown in the two figures below:

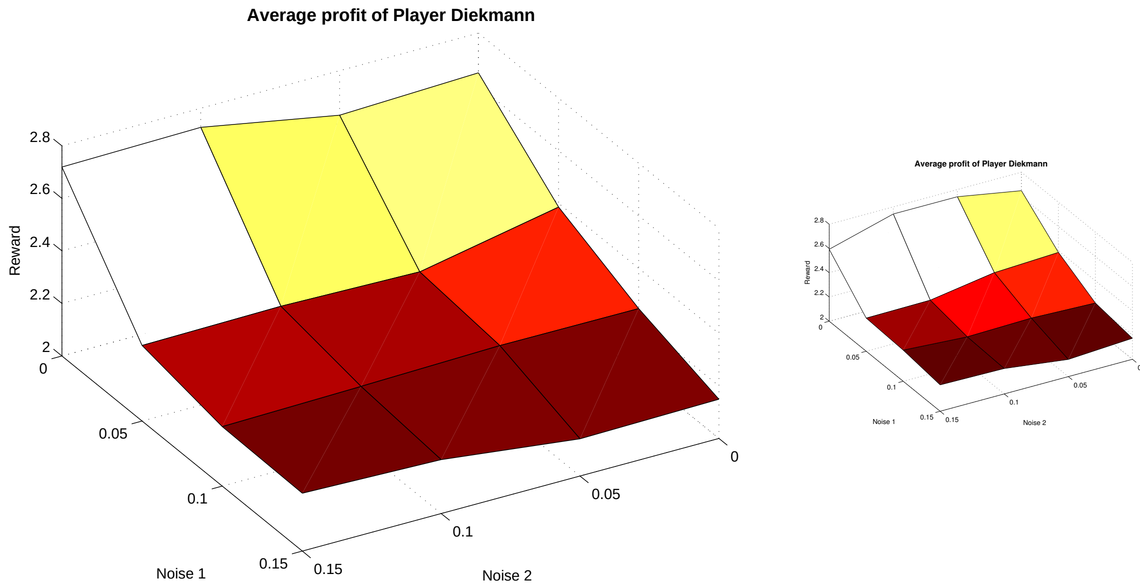


Figure 9: Reward plot of the Player Diekmann

This is a TFT mutant, that performs on the same level as TFT at no noise, and is therefore one of the strongest players if there is no noise. While his performance also drops with Noise1 the effect is much less severe. For Noise1=0.05 and Noise2=0, TFT drops about 0.9, while Diekmann only drops 0.5. This player actually initiates cooperation and gets not stuck in mutual defection with TFT mutants. The weakness of this player is that he is exploitable by defective moves every 10 moves. However on our simulation we haven't seen a player exploiting this weakness. Evolutionary could exploit it if the period in which the algorithm updates would be a multiple of the period in which Diekmann inserts cooperative moves.

Table 9: Cooperation of Diekmann depending on the noise

	Noise 2 = 0	Noise 2 = 0.05	Noise 2 = 0.1	Noise 2 = 0.15
Noise 1 = 0	0.8635	0.9034	0.9113	0.8741
Noise 1 = 0.05	0.7157	0.7263	0.7200	0.7359
Noise 1 = 0.10	0.6230	0.6500	0.6655	0.6946
Noise 1 = 0.15	0.5760	0.5895	0.6283	0.6496

The cooperation drops not nearly as much with the Noise as this is the case for TFT, he even is much more cooperative at no noise. At low noise2 values he is more cooperative than TF2T, but when both noises get high TF2T is more cooperative.

Traits different to TFT:

+ initates cooperation

- More Exploitable

6.5.10 Tit For Average Tat

The player's performance in both simulations is shown in the two figures below:

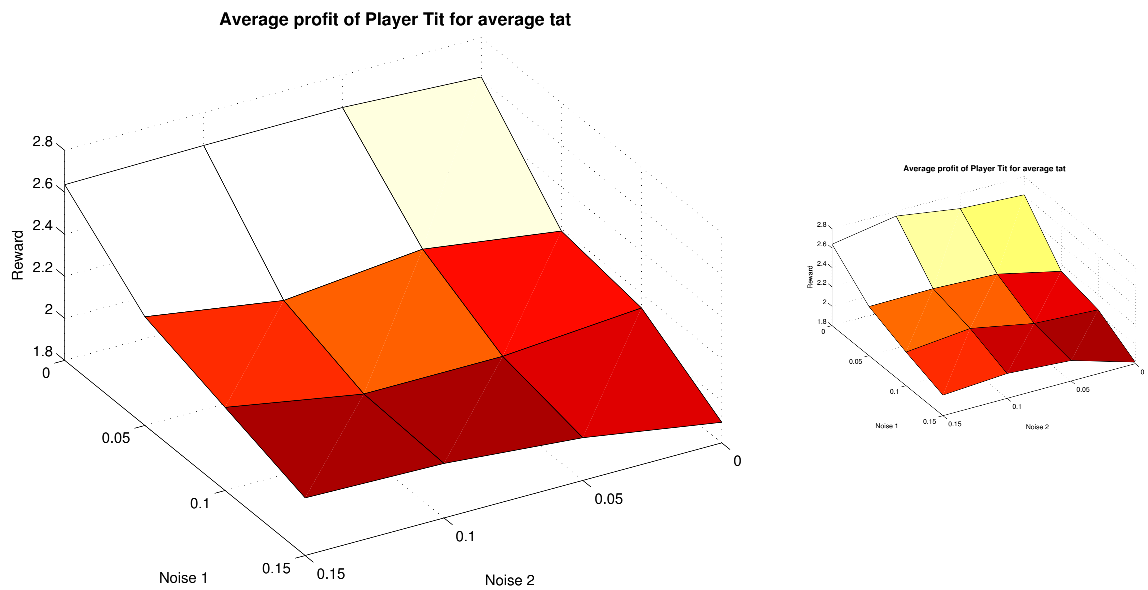


Figure 10: Reward plot of the Player Tit For Average Tat

This TFT mutant looks very similar to Diekmann and other friendlier TFT mutants. The fact that it reacts to the players average move during the last turns allows him to ignore some of the noise. The problem is that if mutual defection appears it is just as hard to get out of it as it was to get into it. Maybe this players performance would have decreased if the simulation was run even longer. In general this player performs about as well as Reconciliation TFT and a little bit worse than Diekmann. Theoretically this player is exploitable.

The cooperation drops surprisingly fast with Noise1, but still not as fast as for TFT. The number of cooperative moves is still more similar to TFT than Diekmann. This is surprising, that the performances look so close, while the underlying moves

Table 10: Cooperation of Tit For Average Tat depending on the noise

	Noise 2 = 0	Noise 2 = 0.05	Noise 2 = 0.1	Noise 2 = 0.15
Noise 1 = 0	0.8056	0.8404	0.8456	0.8547
Noise 1 = 0.05	0.5316	0.6170	0.6140	0.6863
Noise 1 = 0.10	0.4712	0.4860	0.5100	0.5985
Noise 1 = 0.15	0.3419	0.4134	0.4660	0.5098

are so different. Diekmanns very nice approach seems to be just as efficient as TFAT's more retaliating method.

Traits of the player:

+ ignores single moves

- Exploitable

6.5.11 Reconciliation Tit for Tat

The player's performance in both simulations is shown in the two figures below:

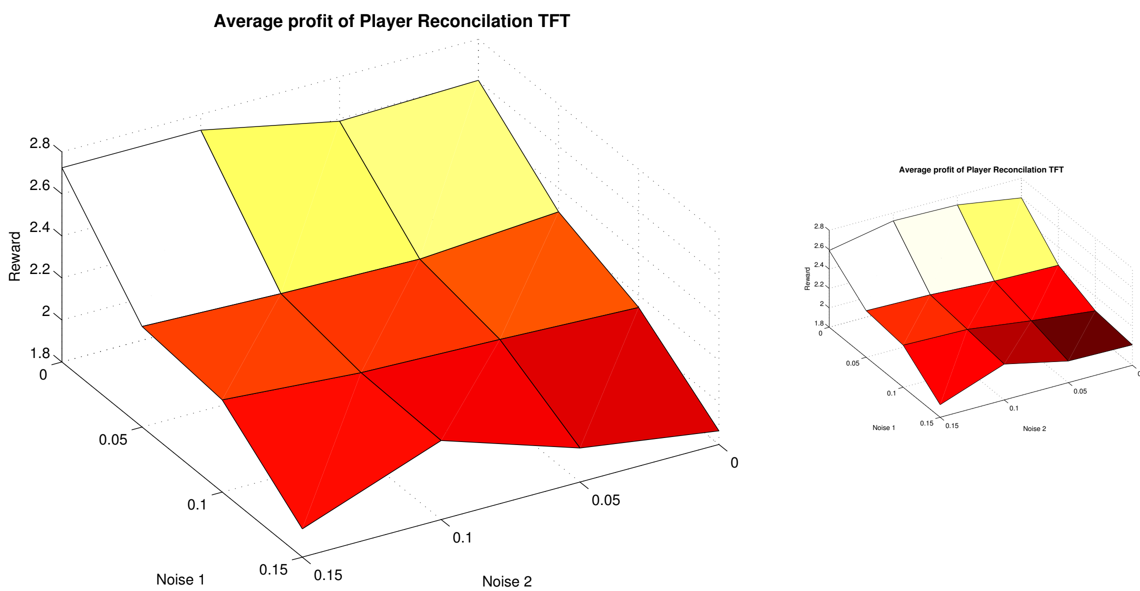


Figure 11: Reward plot of the Player Reconciliation Tit For Tat

Like other more forgiving TFT mutants he has similar performance to TFT without noise, and drops less with noise1. The disadvantage of this player is that while it is not exploitable it still performs worse against defective players, when its reconciliation attempts are shut down over and over again.

Table 11: Cooperation of Reconcilation Tit For Tat depending on the noise

	Noise 2 = 0	Noise 2 = 0.05	Noise 2 = 0.1	Noise 2 = 0.15
Noise 1 = 0	0.8728	0.8972	0.9033	0.8723
Noise 1 = 0.05	0.7169	0.7218	0.7397	0.7450
Noise 1 = 0.10	0.6861	0.7010	0.7175	0.7217
Noise 1 = 0.15	0.6930	0.6955	0.7161	0.67528

The number of cooperations is very high even with high noise1, at noise1=0.15 it's cooperation is higher than that of most TFT mutants. Generally the number of cooperative moves is more similar to Diekmann than to TFAT.

Traits different to TFT:

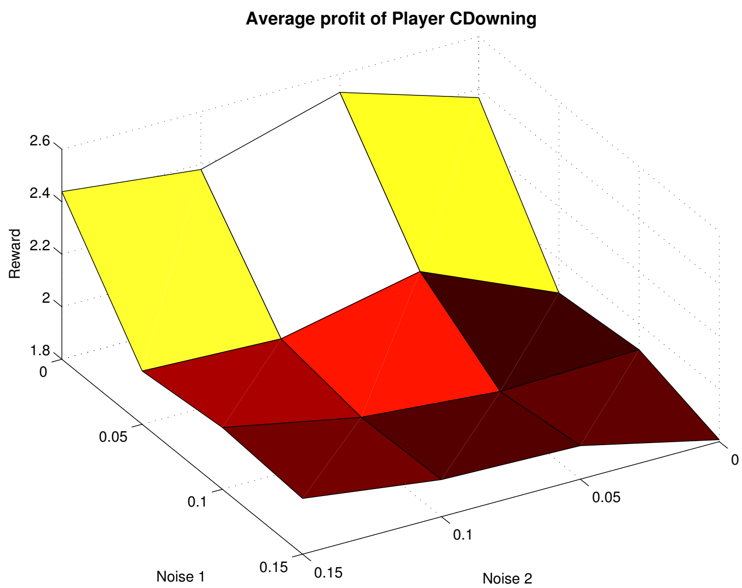
+ Initiates Cooperation

6.5.12 CDowning and DDowning

The two players' performance in both simulations is shown in the two figures below:

If Noise1 is zero, then CDownig performs stronger than DDowning. DDowning performs generally poorly with high Noise2 and no Noise1 there is a slight performance gain. At no noise DDowning outperforms CDowning against "Cooperate" and "Watcher", but is much worse against all TFT mutants and Friedmann. Noise2 improves the performance of DDowning against the TFT mutants a little. In the individual parings it seems that DDowning can end up in either mutual rejection or mutual cooperation. The decision where they end up seems to be random. For example at Noise2=0.15 mutual cooperation appears in TFT and TF2T, but not in Diekmann and Joss, while at Noise2=0.1 it is the other way around.

DDowning seems to reject almost all the time, while CDowning rejects most of the time if Noise1 is greater than 0. A problem of Downing is that it compares decisions that hapen at the same time, but at this time the opponent does not know Downings decision, so his decision can only be dependant on Downings past decisions.



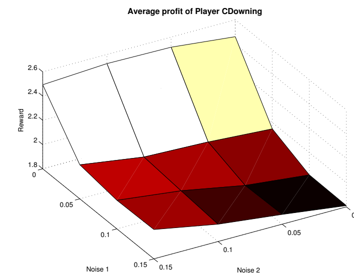


Figure 12: Reward plot of the Player CDowning

Table 12: Cooperations of CDowning depending on the Noise

	Noise 2 = 0	Noise 2 = 0.05	Noise 2 = 0.1	Noise 2 = 0.15
Noise 1 = 0	0.7507	0.6013	0.7660	0.6504
Noise 1 = 0.05	0.1210	0.1711	0.0829	0.0507
Noise 1 = 0.10	0.1169	0.0507	0.0508	0.0508
Noise 1 = 0.15	0.0507	0.0745	0.0504	0.0505

6.5.13 Tit For Tat with Reputation

The player's performance in both simulations is shown in the two figures below:

There were too many unfriendly players, that any player could have been friendly enough so that this player would have looked over a defection. The only player that would cooperate enough is "Cooperate". Against this player the number of cooperations was higher then the one of TFT, but because "Cooperate" is an exploitable player, this actually hurt this strategy. In an environement where most players are cooperating this player could theoretically be exploited.

The values in the upper table are very similar to the results of Tit For Tat

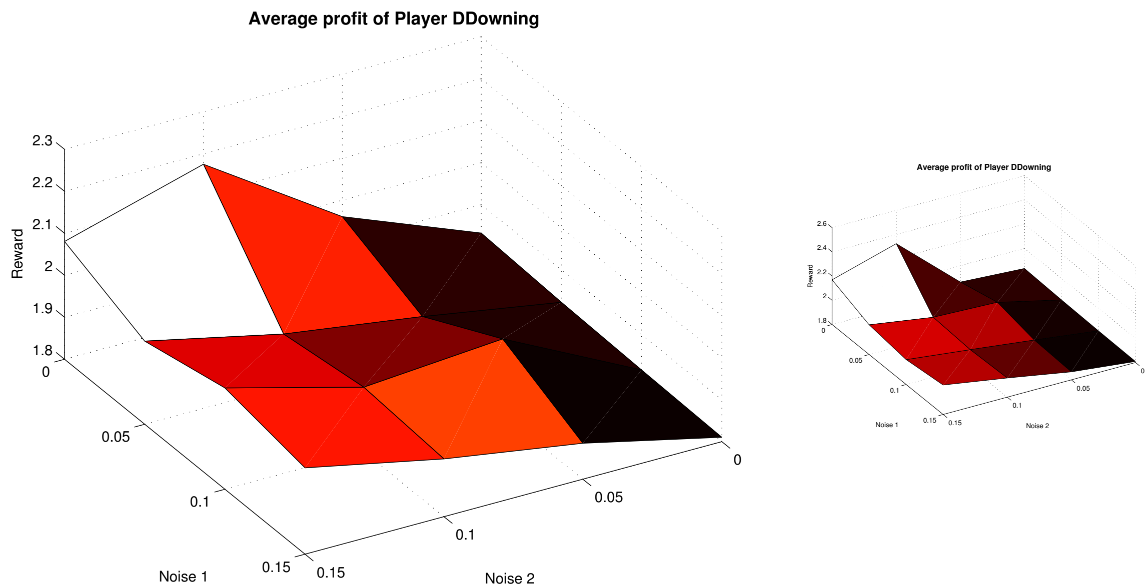


Figure 13: Reward plot of the Player DDowning

Table 13: Cooperations of CDowning depending on the Noise

	Noise 2 = 0	Noise 2 = 0.05	Noise 2 = 0.1	Noise 2 = 0.15
Noise 1 = 0	0.0500	0.1499	0.2997	0.3500
Noise 1 = 0.05	0.0500	0.0501	0.0517	0.0502
Noise 1 = 0.10	0.0500	0.1110	0.0503	0.0502
Noise 1 = 0.15	0.0501	0.0501	0.0501	0.0502

6.5.14 Strategy Switcher

The player's performance in both simulations is shown in the two figures below:

This player was by far the strongest player before the Downing players were added to the simulation. In the final simulation he is strong in fields, where the Noise is large as is performance is not impacted by noise. The disadvantage comes from trying out different strategies in the beginning. In the case of no noise this means that Friedmann will always defect. In the case of Downing mutants this seems to result in defections from the Downing players. Before these Downing players were added this player outperformed TFt by a huge margin, even at zero noise. The strength of this player comes from his ability to cooperate with TFT mutants and exploit exploitable players. This player might even be stronger if better strategies

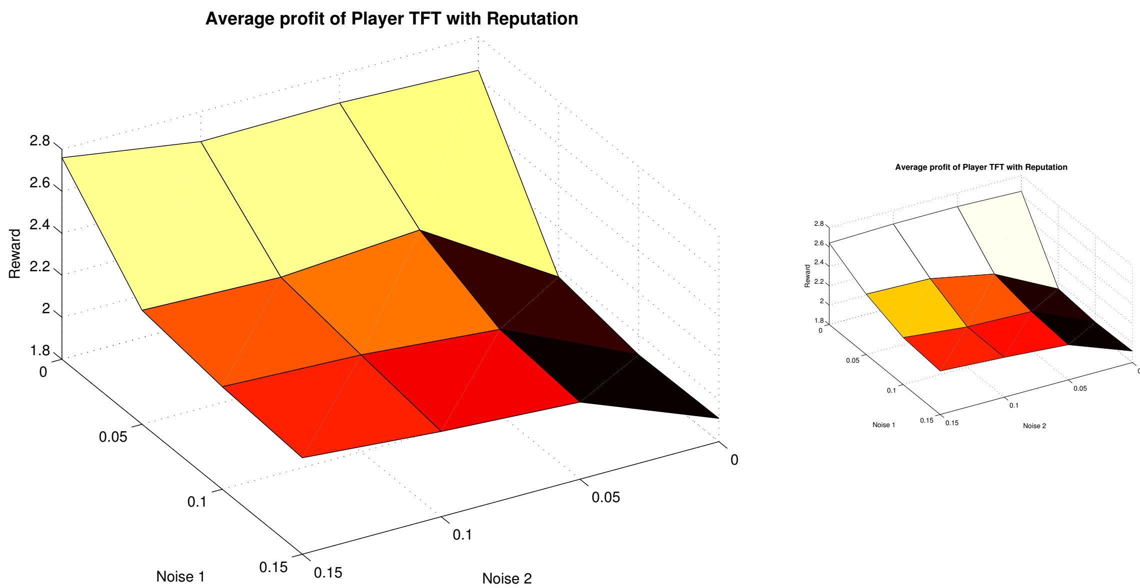


Figure 14: Reward plot of the Player Tit For Tat with Reputation

Table 14: Cooperations of Tit For Tat with Reputation depending on the Noise

	Noise 2 = 0	Noise 2 = 0.05	Noise 2 = 0.1	Noise 2 = 0.15
Noise 1 = 0	0.8059	0.8218	0.8375	0.8440
Noise 1 = 0.05	0.8059	0.8218	0.8375	0.84402
Noise 1 = 0.10	0.3476	0.4939	0.5341	0.5942
Noise 1 = 0.15	0.3202	0.4402	0.4906	0.5349

were given to his arsenal. Currently he had "Cooperate", "Defect", TFT, TF2T and Pavlov as choices. A more efficient choice might have been Limited Reconciliation TFT and "Defect". Currently it also contains exploitable strategies (TF2T), so the player could be exploited.

In long simulations the player can benefit from having the optimal strategy, while in short simulations he spends a large amount of the time trying out strategies that might not be very strong.

The number of cooperations is rather low, but also does not change very much with the noise. The number of cooperations is most likely low, because this player actively looks if an opponent is exploitable.

Traits of the player:

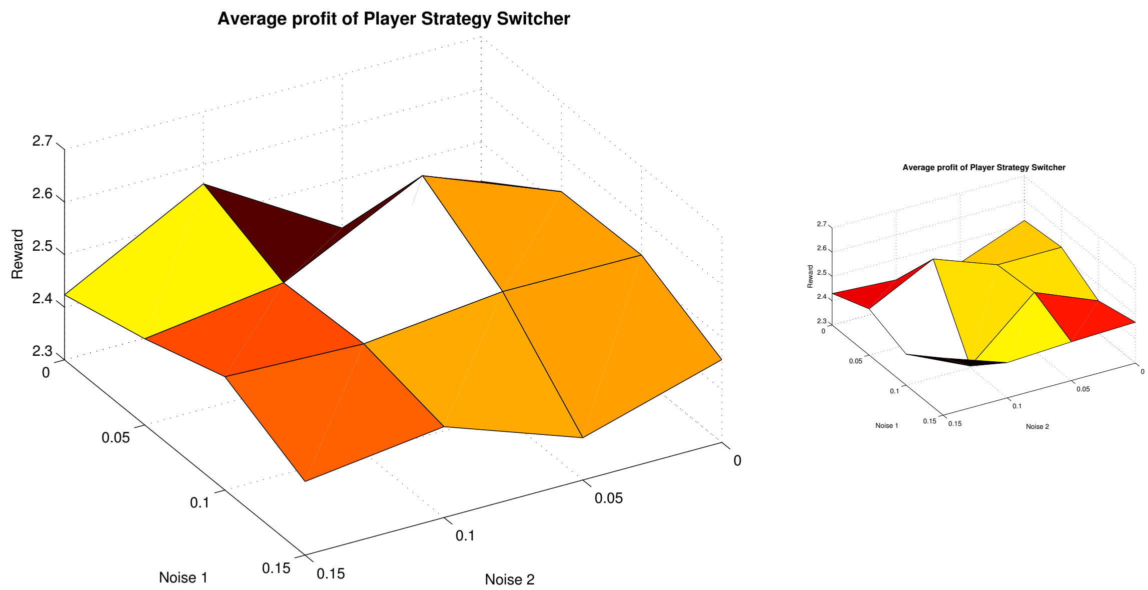


Figure 15: Reward plot of the Player Strategy Switcher

Table 15: Cooperations of Strategy Switcher depending on the Noise

	Noise 2 = 0	Noise 2 = 0.05	Noise 2 = 0.1	Noise 2 = 0.15
Noise 1 = 0	0.5164	0.5430	0.4803	0.4659
Noise 1 = 0.05	0.5164	0.5430	0.4803	0.4659
Noise 1 = 0.10	0.5098	0.5520	0.4497	0.4888
Noise 1 = 0.15	0.5097	0.5318	0.4675	0.4467

+ Can exploit others

+ Noise doesn't impact performance

+ Very adaptive

+ Strong in long simulations

- Exploitable himself

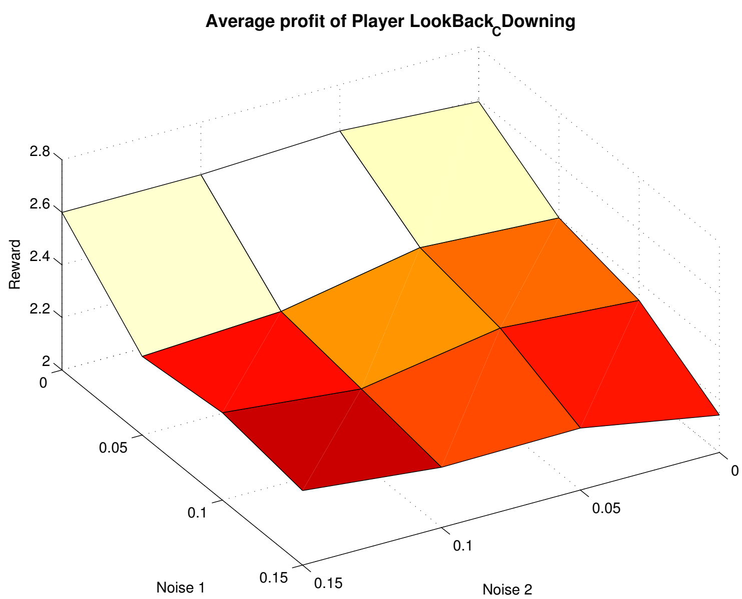
- Exploring defective moves can backfire (Friedmann)

- Is hard to read initially, this might trigger defections

- Weak in short simulations

6.5.15 Lookback CDowning and Lookback DDowning

The two players' performance in both simulations is shown in the two figures below:



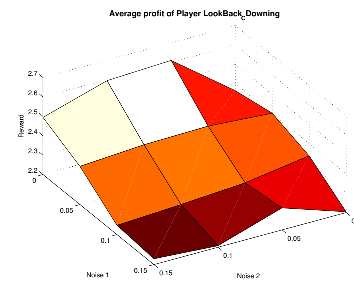


Figure 16: Reward plot of the Player Lookback CDowning

Generally both Lookback Downings are stronger than the not Lookback Downings. If Noise1 is zero, the CDowning mutant has again a higher performance. Comparing the second last move with the last move of the opponent seems to be the better way to correlate your actions with your opponent's actions. DDowning seems to be very resilient to Noise.

At zero noise CDowning performs well with most players, except "Evolutionary", Strategy Switcher, and some Downing mutants. With noise the performance against Friedmann and some Downing mutants that went well before drops. The overall performance however stays higher than TFT. The Look-back Downing behaves similar like Lookback CDowning behaves with noise.

The number of Cooperations is in general much higher then for the Variants that do not look-back. For Noise1 greater than zero about 40% cooperative moves remain, while CDowning and DDowning fall down to 5-10% cooperative moves. The average rewards for these players are also 0.2-0.5 higher then their not look-back counterparts.

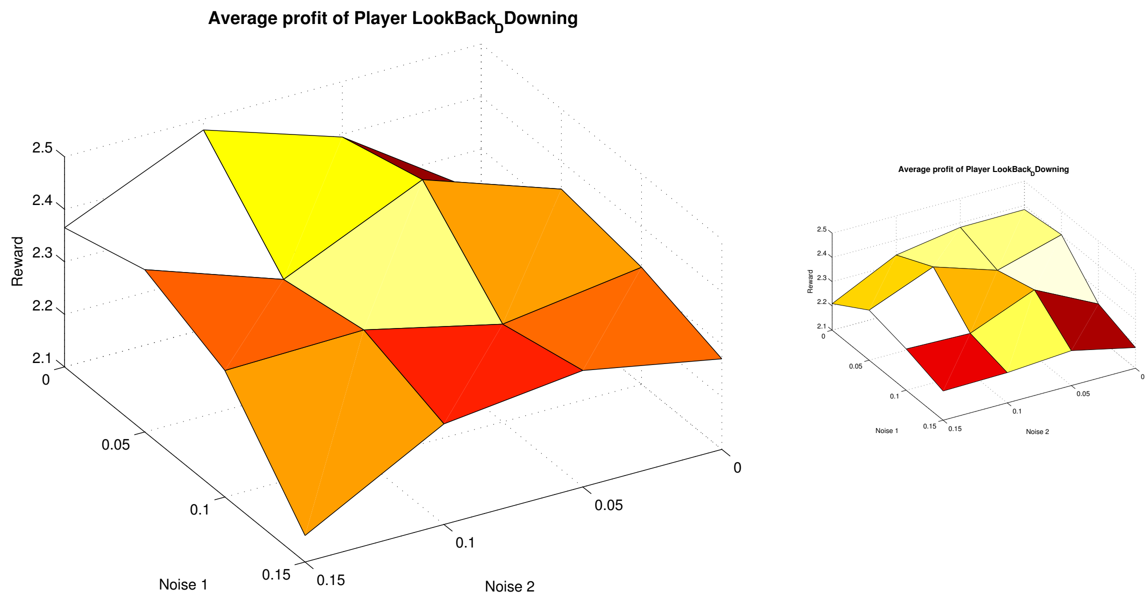


Figure 17: Reward plot of the Player Lookback DDowning

Table 16: Cooperations of the Lookback CDowning depending on the Noise

	Noise 2 = 0	Noise 2 = 0.05	Noise 2 = 0.1	Noise 2 = 0.15
Noise 1 = 0	0.7507	0.6511	0.6509	0.7010
Noise 1 = 0.05	0.4385	0.4405	0.4376	0.4329
Noise 1 = 0.10	0.4297	0.3865	0.3787	0.3696
Noise 1 = 0.15	0.3814	0.4120	0.3682	0.3271

6.5.16 Watcher

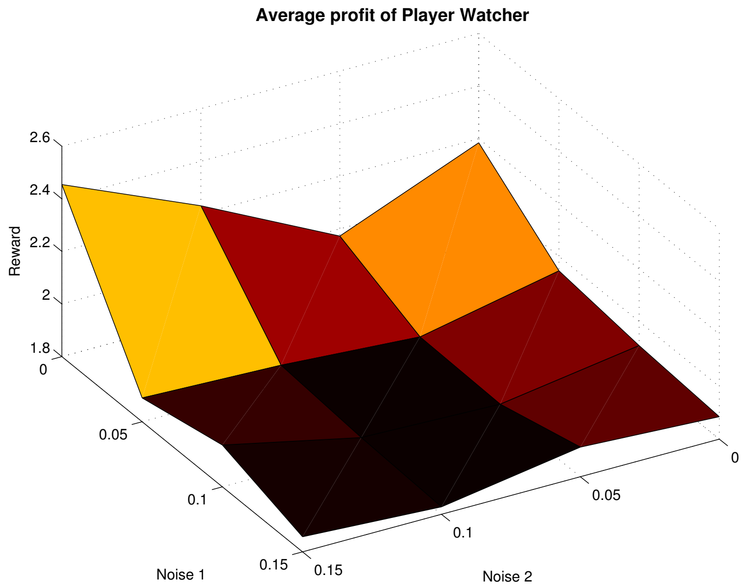
The player's performance in both simulations is shown in the two figures below:

This player generally does not perform very strong. It does not respond and can therefore be exploited. It also does not take the local situation into account. It will betray Friedmann even in no Noise, because in short term this is successful. After that the player copies strategies that were cooperative the whole time, while Friedmann is defecting. There is point of higher performance, where Noise2 is 0.15 and Noise 1 is zero. For some reason it performs strong against the Lookback-Downing algorithms.

Generally the number of cooperations is rather low, but there are no drastic jumps.

Table 17: Cooperations of the Lookback DDowning depending on the Noise

	Noise 2 = 0	Noise 2 = 0.05	Noise 2 = 0.1	Noise 2 = 0.15
Noise 1 = 0	0.4037	0.4563	0.4071	0.4762
Noise 1 = 0.05	0.4386	0.3912	0.4303	0.3848
Noise 1 = 0.10	0.3864	0.4025	0.3784	0.3827
Noise 1 = 0.15	0.3868	0.3262	0.3840	0.3769



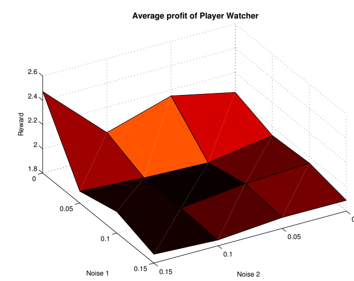


Figure 18: Reward plot of the Player Watcher

6.5.17 Evolutionary

The player's performance in both simulations is shown in the two figures below:

This player performs rather poorly at all noises. Less Noise is better for him, because more reliable information allows the player to adjust. Noise generally promotes bad strategies, while they are sorted out when there is little noise. The problem is that this players does add rejections and therefore triggers others rejections. The player himself has a very slow reaction time. Changing his strategy by mutations takes hundreds of turns. This is a stimescale the opponents cannot see. To the opponents this player does not look responsive. The player himself can only see the opponents reaction if it is within the segment legnth that the player tries to optimize. The interesting thing is that this player is able to exploit TF2T, he will add

Table 18: Cooperations of the Watcher depending on the Noise

	Noise 2 = 0	Noise 2 = 0.05	Noise 2 = 0.1	Noise 2 = 0.15
Noise 1 = 0	0.5399	0.4704	0.4298	0.4020
Noise 1 = 0.05	0.4569	0.3854	0.3544	0.3552
Noise 1 = 0.10	0.4297	0.3691	0.3522	0.3576
Noise 1 = 0.15	0.4125	0.3510	0.3428	0.3504

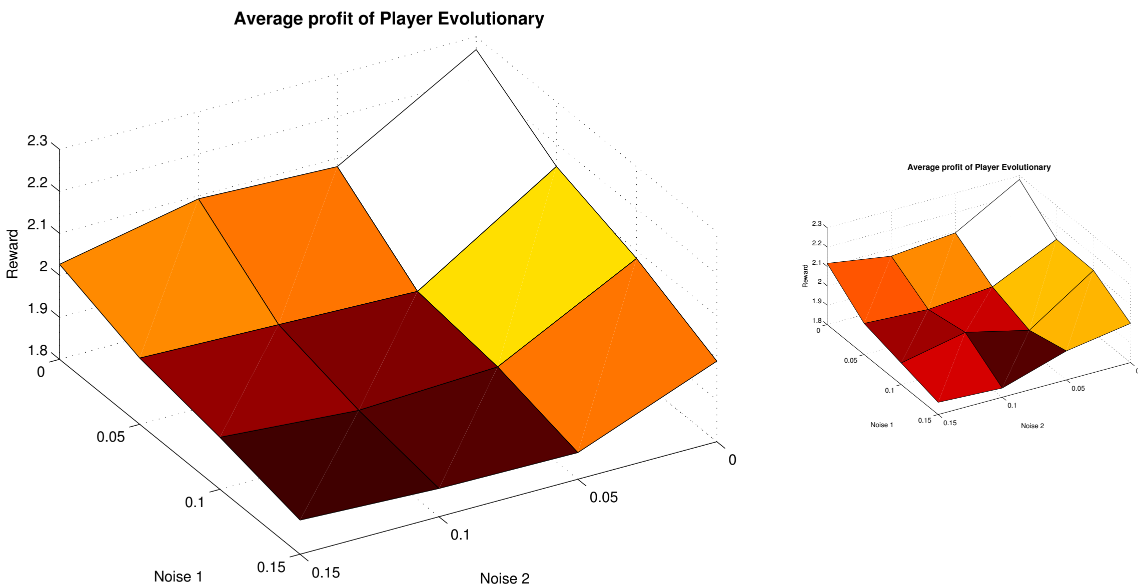


Figure 19: Reward plot of the Player Evolutionary

defections with mostly one sometimes two cooperative steps between them. It has a performance of 3.65 against TF2T at zero noise.

The decisions seem to be mostly an even mix of defections and cooperations, with a slight preference of defections.

Table 19: Cooperations of the Evolutionary depending on the Noise

	Noise 2 = 0	Noise 2 = 0.05	Noise 2 = 0.1	Noise 2 = 0.15
Noise 1 = 0	0.4330	0.4941	0.5151	0.4878
Noise 1 = 0.05	0.3859	0.4745	0.4860	0.4855
Noise 1 = 0.10	0.3987	0.4750	0.4919	0.4914
Noise 1 = 0.15	0.3755	0.4793	0.4733	0.4896

6.5.18 Limited Reconciliation Tit For tat

The player's performance in both simulations is shown in the two figures below:

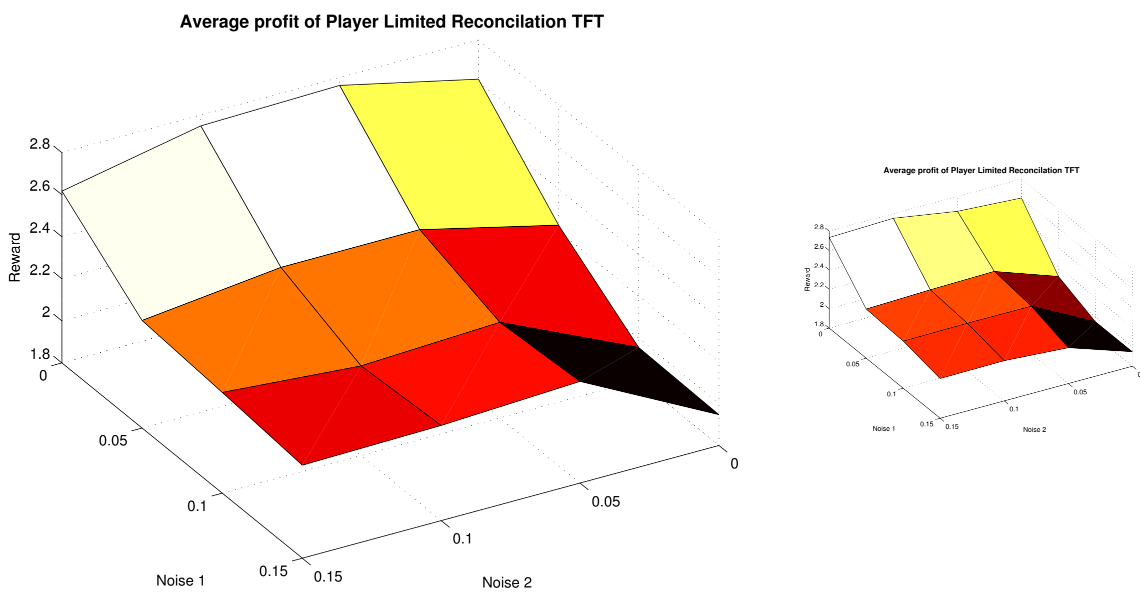


Figure 20: Reward plot of the Player Limited Reconcilation Tit For tat

At noise1 equal to zero the performance is similar to TFT. It does not perform as well as Reconciliation TFT against Joss, therefore it might be wiser to make the time between the reconciliation attempts larger every time, instead of limiting it to 3. Compared to TFT its performance does not break down that much with noise1, this is a property it shares with Diekmann, TFAT and Reconciliation TFT. The fact that the number of reconciliation attempts is limited makes the player stronger against defecting players like Friedmann (under noise1) and "Defect".

It is not as cooperative as the friendliest TFT mutants Diekmann and RTFT, but more friendly than standard TFT and TFAT.

Table 20: Cooperations of the Evolutionary depending on the Noise

	Noise 2 = 0	Noise 2 = 0.05	Noise 2 = 0.1	Noise 2 = 0.15
Noise 1 = 0	0.8064	0.8386	0.8942	0.8976
Noise 1 = 0.05	0.5084	0.6577	0.6834	0.7043
Noise 1 = 0.10	0.4254	0.6300	0.6568	0.6822
Noise 1 = 0.15	0.4208	0.5956	0.6383	0.6648

The table below shows the performance of LTFT minus the performance of TFT averaged over all matchups and both simulations:

Table 21: Comparison of the two players Limited Tit Fot Tat and Tit For Tat

	Noise 2 = 0	Noise 2 = 0.05	Noise 2 = 0.1	Noise 2 = 0.15
Noise 1 = 0	-0.0317	0.0414	0.0375	-0.0892
Noise 1 = 0.05	0.2235	-0.0006	0.0310	-0.0679
Noise 1 = 0.10	0.0438	0.0340	0.0019	0.0066
Noise 1 = 0.15	0.0514	0.0720	0.0189	-0.0312

At higher values for Noise1, LTFT outperforms TFT, while at high Noise2 values the reconciliation attempts are not needed, because the noise itself achieves that.

6.6 Impact of Noise on the whole system

Below are the figures that show how the average Reward and the average Cooperation behave in the system. Total Cooperation against Noise:

Average Reward against Noise

Noise 1 seems to destroy cooperation really fast, while Noise2 increases it a little. Generally the average reward is the highest if the number of cooperative moves is the highest. The Average Rewards dependence on the noise looks very similar to the values a typical TFT mutant has. This may be related to the fact, that there were 8 TFT mutants in the simulation.

6.7 Comparison of the players

Best strategy at each Noise:

At Noise1 equal zero the TFT mutants win, but for every noise1 larger than zero the strategy switcher wins, due to the small impact noise has on his performance.

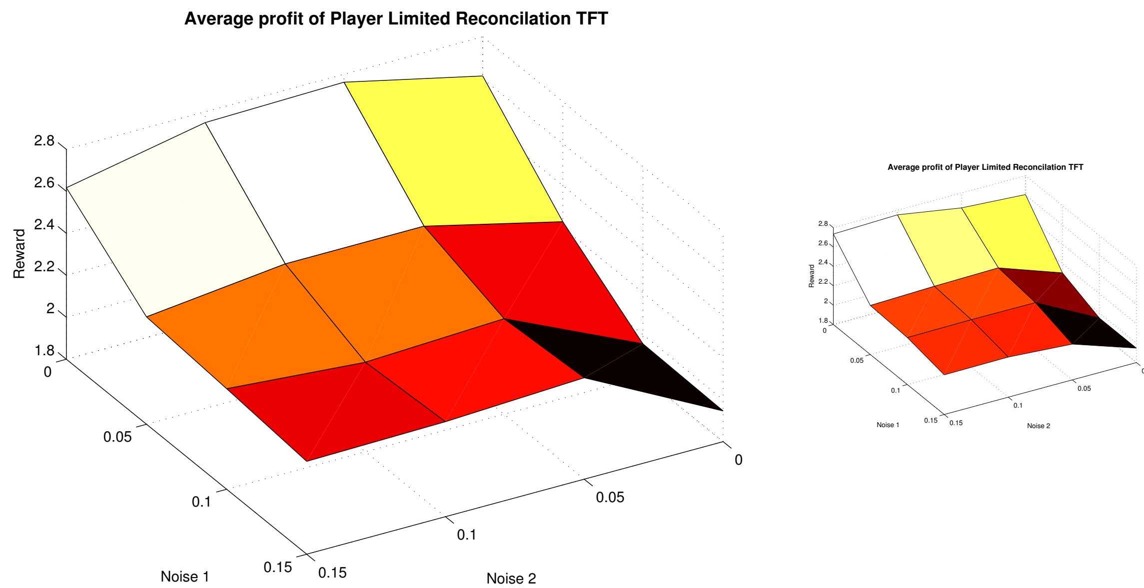


Figure 21: Reward plot of the Player Limited Reconcilation Tit For tat

In the following graphs the players performances are compared at zero noise, Noise1=0 and Noise2=0.15, Noise1=0.1 and Noise1=0 and both noises 0.1. The results are from the first simulation run.

Comparison of all players at zero noise:

With no noise the best strategies are either TFT mutants, or variants of CDowning. Defective strategies perform poorly. This is in accordance to what Axelrod said: Be nice.

Noise1=0, Noise2=0.1

The ranking of the different strategies does not change much. Generally everybody is profiting of the noise. Noise1=0.1, Noise2=0

The performance of the TFT mutants drastically decreases. Only Diekmann stays somewhat high. The strongest strategies are now the Lookback Downing and the strategy switcher. The strategy switcher before had the problem that the defections he tried out to exploit non-responding players cost him a lot. Now everybody sees defections due to the noise and it doesnt matter that much anymore.

Noise1=0.1, Noise2=0.1

The TFT mutants perform somewhat better than without Noise2, but the strategy switcher is still much stronger than the competition.

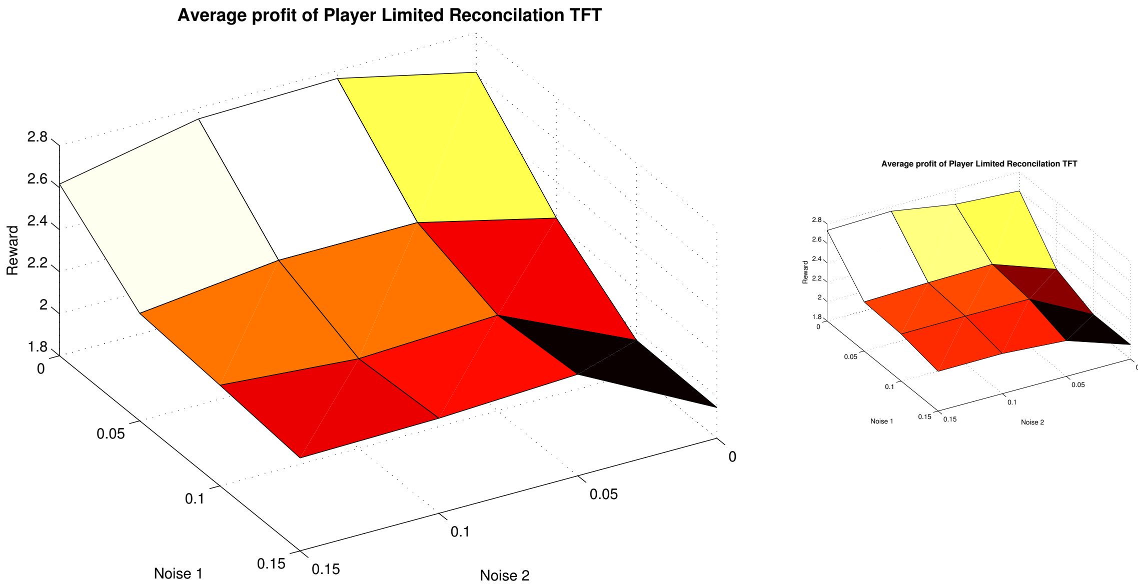


Figure 22: Reward plot of the Player Limited Reconcilation Tit For tat

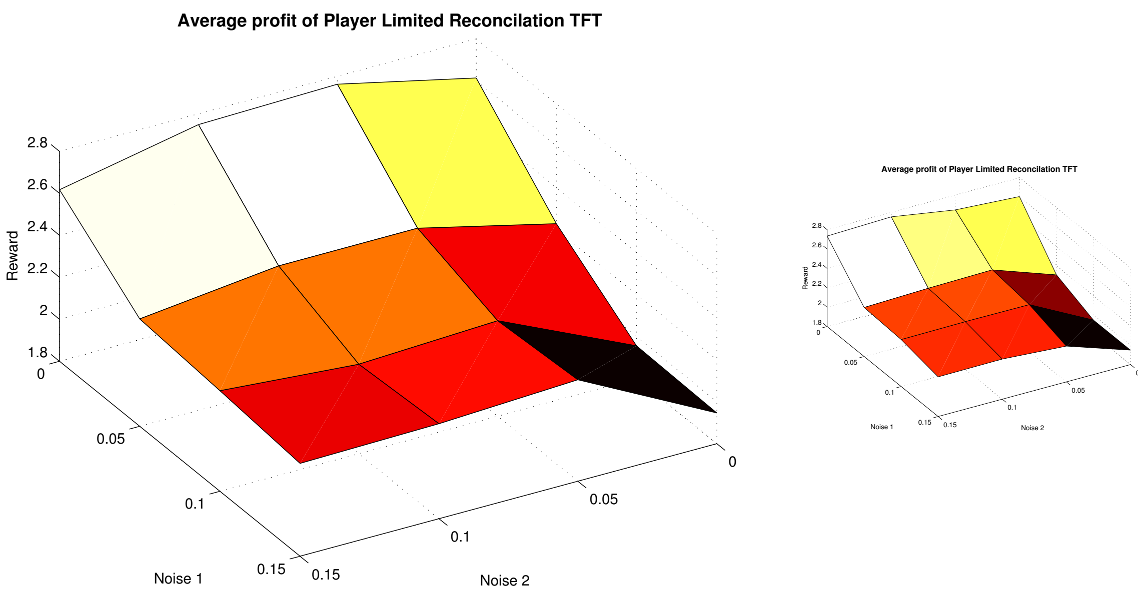


Figure 23: Reward versus Noise with the best players

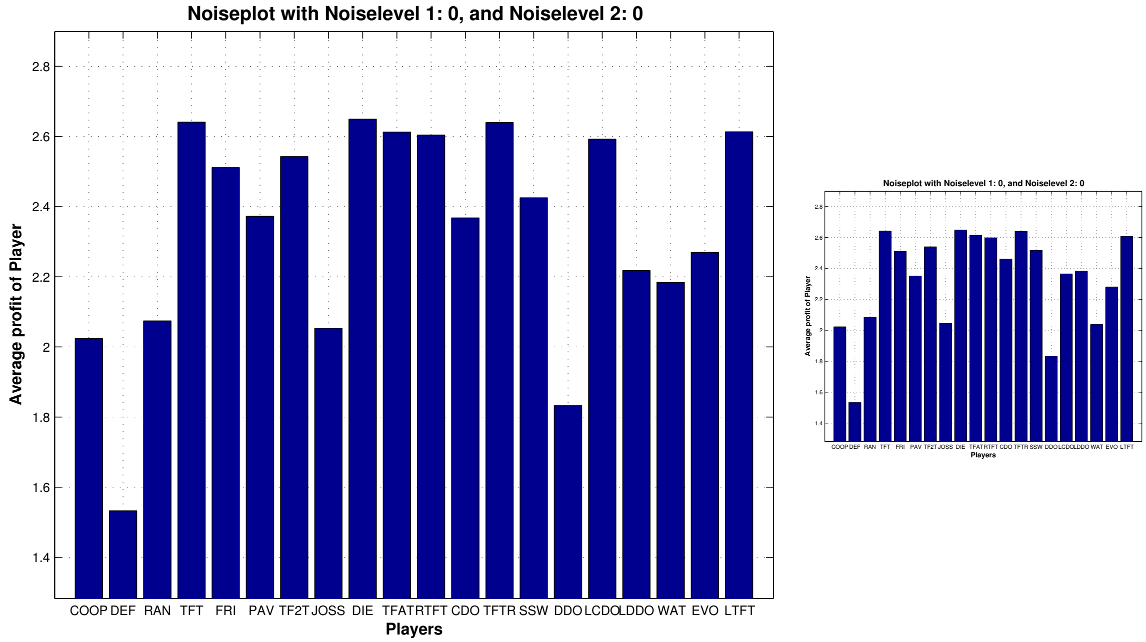


Figure 24: Reward of players with both noises equals to zero

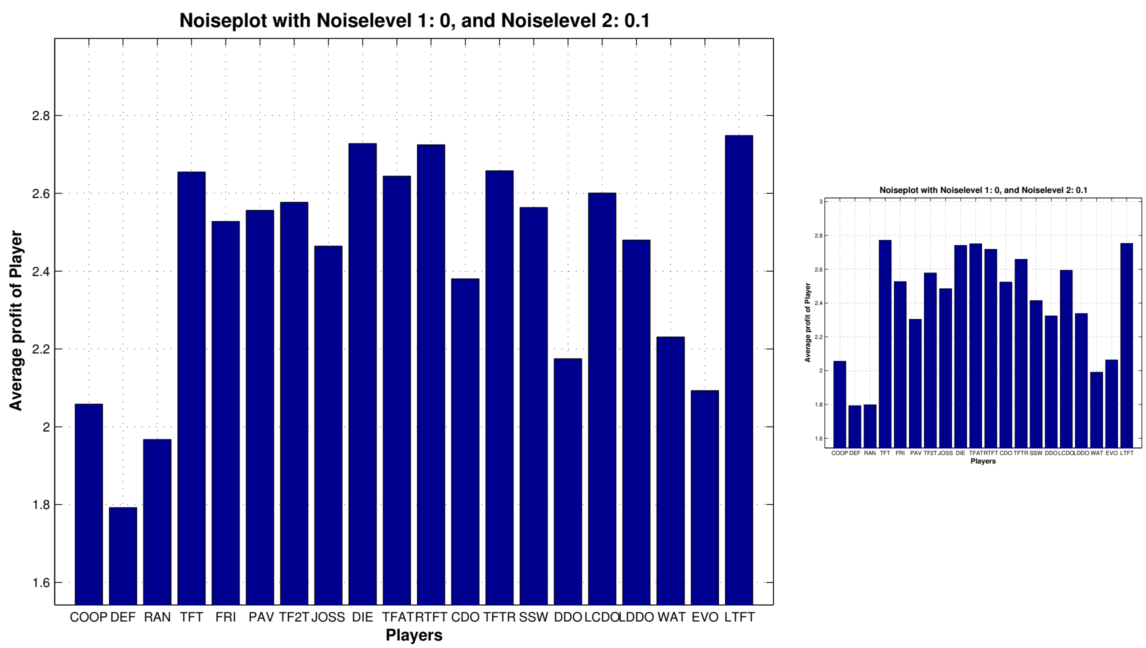


Figure 25: Reward of players with both noises equals to zero

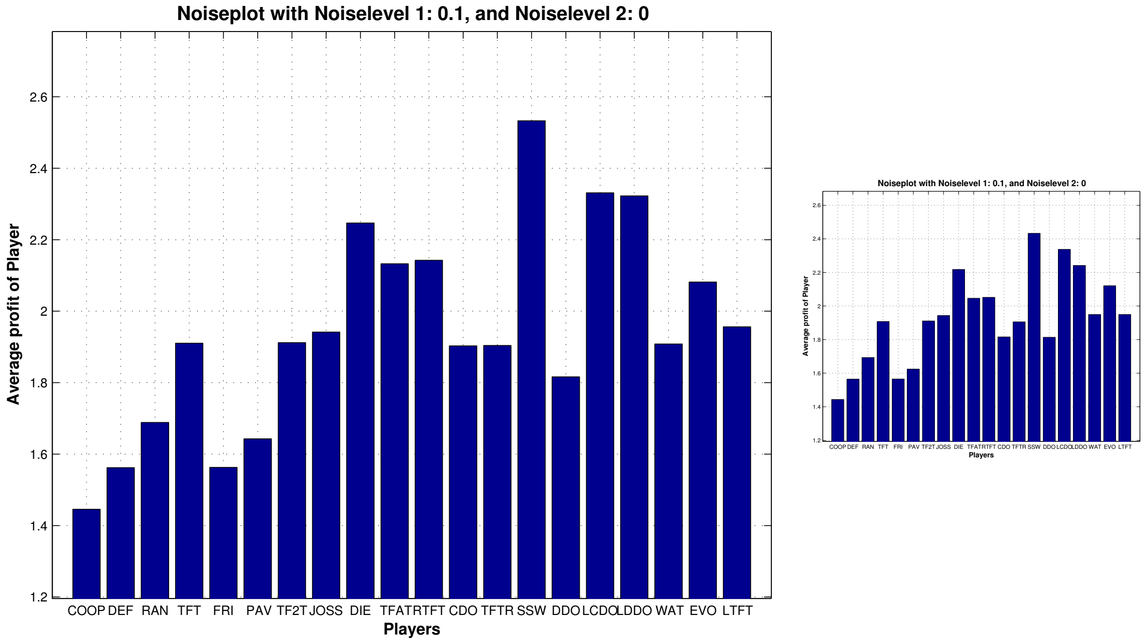


Figure 26: Reward of players with both noises equals to zero

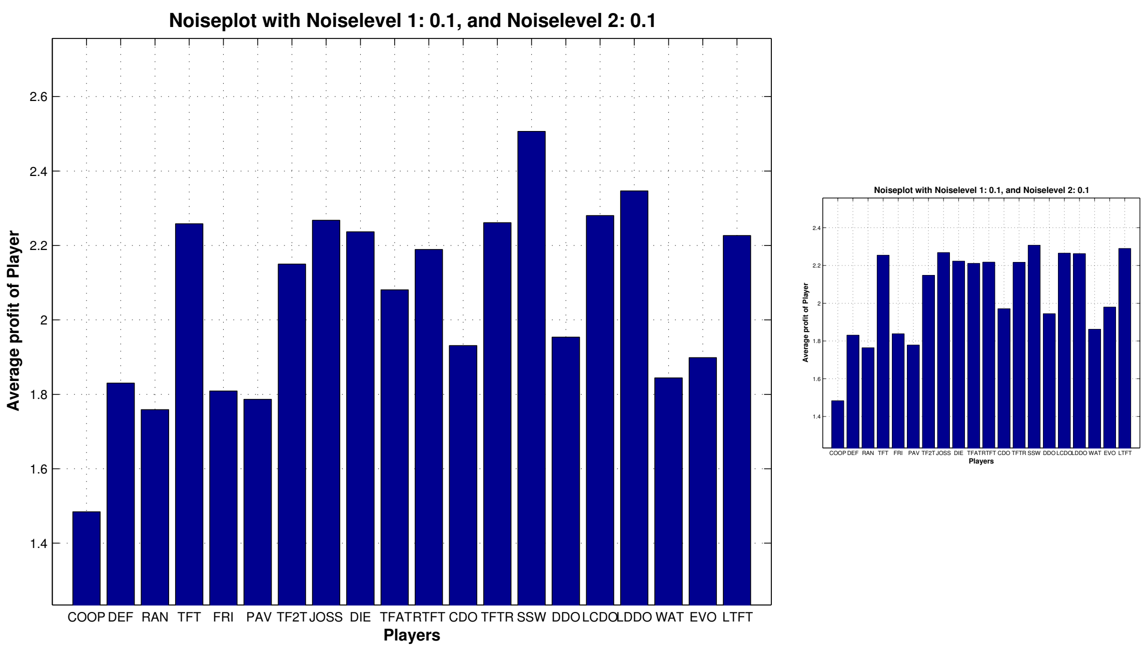


Figure 27: Reward of players with both noises equals to zero

7 Summary and Outlook

The aim of this simulation was to investigate two things. The first was the impact of noise on the tournament. It turned out that Noise that lets Defections appear as cooperations is beneficial for most players. The opposite, when cooperative moves are perceived as defections, has a much larger impact. The performance of most friendly players drastically drops. It is especially harsh for players that relied on the effect of the first move being cooperative and have no mechanism to restore cooperation once it is lost.

The second investigated topic was how learning players would perform. Of the three learning mechanisms, copying others, Evolution and Strategy switching, Strategy switching performed the strongest. The strategy was also stronger than most not learning strategies. The other two approaches failed, because they were not responsive.

Outlook: The performance of the players was heavily impacted by the nature of the other players participating in the simulation. It would be interesting to run the simulation with more players. Another possible investigation could be to find out if the average performance increases with a noise that covers up defections forever, or if there is a turning point after which the performance decreases again.

8 References

[1] Donninger, C., "Is it always efficient to be nice?". Wien: Physica-Verlag Heidelverg (1986)

[2] Kuhn, S., "Prisoner's Dilemma". URL: http://plato.stanford.edu/entries/prisoner-dilemma (accessed october 15, 2011)

[3] Axelrod, R., "Die Evolution der Kooperation". Munich: Oldenbourg Wissenschaftsverlag GmbH, German Edition (2000)

[4] Nowak, M., "Five Rules for the Evolution of Cooperation". Science 314, page 1560 (2006)

[5] Sandholm, T. et al, "Multiagent Reinforcement Learning in the Iterated Prisoner's Dilemma". Biosystems Vol 37, pages 147-166 (1995)

[6] Queck, H. et al, "Adaptation of Iterated Prisoner's Dilemma Strategies by Evolution and Learning". Computational Intelligence and Games, pages: 40-47 (2007)

[7] Wu, J. et al, "How to Cope with Noise in the Iterated Prisoner's Dilemma". Journal of Conflict Resolution, Vol. 39, pages 183-189 (1995)

A Submitted Researchplan

A.1 General Introduction

Tournament like simulation of the prisoner's dilemma with repeated inter- actions. Random errors are introduced in the information about the player's recent behavior. We want to observe the different outcome of the traditional players if noise is introduced. Further we want to try to implement new players with learning strategies.
We believe that this makes the simulation more realistic.
Extension of Axelrod's Tournaments.

A.2 Fundamental Questions

Can a dispute based on miscommunication be overcome?
Can treason be hidden behind pretended miscommunication?
Does miscommunication discourage cooperation?
How much miscommunication can cooperation survive?
Do learning strategies have an advantage over the other ones?
How do the traditional players act and how does the final result change, if noise is introduced?
Independent variables: length of simulation, reliability of communication, rewards
Dependent variables: correlation between cooperation and success, frequency of cooperation, successful strategies

A.3 Expected Results

Miscommunication works against cooperating strategies.
Programs that reconcile are more successful.
The reward of the learning players is less influenced by the noise.

A.4 References

- On Evolving Robust Strategies for Iterated Prisoner's Dilemma, P. J. DARWEN and X. YAO, 16. November 1993

- Multiagent Reinforcement Learning in the Iterated Prisoner's Dilemma, T. W. SANDHOLM and R. H. CRITES

- Adaptation of Iterated Prisoner's Dilemma Strategies by Evolution and Learning, H. Y. QUEK and C. K. GOH, 2007

- How to Cope with Noise in the Iterated Prisoner's Dilemma, J. WU and R. AXELROD, JOURNAL OF CONFLIC RTESOLUTION, Vol. 39 No. 1, March1995 183-189

- Five Rules for the Evolution of Cooperation, M. A. NOWAK, Science 314, 1560 (2006)

A.4.1 Research Methods

Agent-Based Model

A.5 Other

The type(s) of the learning strategies we will decide later, after reading some of the literature.

B Matlabcode

B.1 Master.m

Listing 1: Master.m

```matlab
clear all;close all; home                   % Initalisation
% [a, MSGID] = lastwarn();
% warning('off', MSGID)
%Note: Most standard players are taken from the lecture http://www.socio.ethz.ch/education/fs11/igt/notes/Evolution_von_Kooperation_2011.pdf

tic                                         % start time measurement
N = 20000                                   % Number of turns
maxplayers = 20;                            % Maximum number of players
K = zeros(maxplayers,maxplayers,N );        % Contains the information about the players true decisions: 1=Cooperate   2=Betray
K2 = zeros(maxplayers,maxplayers,N );       % Contains the information about the players decision disturbed by noise
minNoise1 = 0                              % The chance that cooperation gets recieved as betrayal goes from the value minNoise1 to maxNoise1
maxNoise1 = 0.15
minNoise2 = 0                               % The chance that betrayal gets recieved as cooperation goes from the value minNoise2 to maxNoise2
maxNoise2 = 0.15
NoiseInc=0.05;                              % Noise increment with each simulation
maxX=(maxNoise1-minNoise1)/NoiseInc+1+10^-15;  % number of points of the x-axis, the last addition is to prevent floating point errors
maxY=(maxNoise2-minNoise2)/NoiseInc+1+10^-15;  % number of points of the y-axis
player = 'player';                           % Name of the player functions
Rewardmatrix=zeros(maxplayers,maxplayers,maxX,maxY); % Matrix that tracks how many points the players get out of each encounter
Reward=zeros(2,1);                          % Rewards that the players get in an encounter
Points=zeros(maxplayers);                   % Total amount of points of a player
AverageCoop=zeros(maxX,maxY,maxplayers,maxplayers); % The average cooperation at a given noise for a given matchup
SysAvRew=zeros(maxX,maxY);                  % The average reward in the whole system for a given noise

Winner=0;                                   % most succesful player

list = playerlist(player, maxplayers);

Noise(1,1:maxX)=(minNoise1:NoiseInc:maxNoise1);
Noise(2,1:maxY)=(minNoise2:NoiseInc:maxNoise2);
```

```
for x=1:maxX
    Noise1=(x-1)*NoiseInc+minNoise1;
    for y=1:maxY
        Noise2=(y-1)*NoiseInc+minNoise2;
        %create the players
        for i=1:maxplayers
            if list(i)==1
                i2=int2str(i);
                eval(['P' i2 '=player' i2 '(' num2str(maxplayers) ');']);
                Names{i}=eval(['P' i2 '.name']);
                Shorts{i}=eval(['P' i2 '.short']);
            end
        end
        for i=1:N % loop trough all turns
            for j=1:maxplayers % loop trough all players
                for k=1:j      % let each player interact with all other players
                    j2=int2str(j);
                    k2=int2str(k);
                    if list(j)==1 && list(k)==1
                        K(j,k,i)=eval(['P' j2 '.decide(K2,k,i)']);  % player j decides how to behave to player k
                        K(k,j,i)=eval(['P' k2 '.decide(K2,j,i)']);  % player k decides how to behave to player j
                        Reward=win([K(j,k,i) K(k,j,i)]); % Rewards are calculated
                        if (j == k)
                            Reward=Reward/2; % otherwise the interaction with itself get counted double
                        end
                        Points(j) = Points(j) + Reward(1);  % Points get updated
                        Points(k) = Points(k) + Reward(2);
                        Rewardmatrix(j,k,x,y) = Rewardmatrix(j,k,x,y) + Reward(1);
                        Rewardmatrix(k,j,x,y) = Rewardmatrix(k,j,x,y) + Reward(2);
                        % noise/miscommunication
                        if (K(j,k,i) == 1) %player j cooperates
                            if (rand > Noise1) %transmission correct
                                K2(j,k,i) = 1;
                            else % miscommunication
                                K2(j,k,i) = 2;
                            end
                        else %player j betrays
                            if (rand > Noise2) %transmission correct
                                K2(j,k,i) = 2;
                            else % miscommunication
                                K2(j,k,i) = 1;
                            end
                        end
```

```
                        if (K(k,j,i) == 1) %player k cooperates
                            if (rand > Noise1) %transmission correct
                                K2(k,j,i) = 1;
                            else % miscommunication
                                K2(k,j,i) = 2;
                            end
                        else % player k betrays
                            if (rand > Noise2) %transmission correct
                                K2(k,j,i) = 2;
                            else % miscommunication
                                K2(k,j,i) = 1;
                            end
                        end
                    end
                end
            end
        end
        %delete players
        for i=1:maxplayers
            if list(i)==1
                i2=int2str(i);
                eval(['clear P' i2]);
            end
        end
        %Output Cooperation and average Reward in the system
        for i=1:maxplayers
            for j=1:maxplayers
                AverageCoop(x,y,i,j)=2-mean(mean(K(i,j,:)));
            end
            SysAvRew(x,y)=mean(mean(Rewardmatrix(:,:,x,y)))/N;
        end
        %output progress
        Noise1
        Noise2
        toc
    end
end

save simulation2 Rewardmatrix N Names Noise AverageCoop Shorts;
toc % end time measurement
```

B.2 win.m

Listing 2: win.m

```
function [ Reward ] = win( K )
%GEWINN Gewinnberechnung

    if K(1)==1 && K(2)==1
            Reward(1)=3;
            Reward(2)=3;

    elseif K(1)==2 && K(2)==1

             Reward(1)=5;
             Reward(2)=0;

    elseif K(1)==1 && K(2)==2

             Reward(1)=0;
             Reward(2)=5;

    elseif K(1)==2 && K(2)==2

             Reward(1)=1;
             Reward(2)=1;

    else
        disp('Unknown decisions were made!!!')

    end
```

B.3 show_data.m

Listing 3: show_data.m

```
%##################################################################################
% Content of file:
% 1.cell: initialization
% 2.cell: plot reward of all players with given noiselevels
% 3.cell: plot cooperation of all players with given noiselevels
% 4.cell: plot statistics for a given player (reward vs given noiselevels)
% with surface plot
% 5.cell: plot statistics for a given player (cooperation vs given
% noiselevels) with surfac plot
% 6.cell: reward vs noise with name of the best player in plot
% 7.cell: total cooperation/reward normed with surface plot
% 8.cell: 2 given Players against each other

% Use of file:
```

```
% 1. set filename of the simulation file in the 1. cell
% 2. set desired noiselevel in the 2.cell
% 3. set desired noiselevel in the 3.cell
% 4. set desired positions of players and desired noiselevels in the 4.cell
% 5. set desired positions of players and desired noiselevels in the 5.cell
% 6. set playersInRange true in the 6.cell to write the players in range
% in a textfile
% 7. set the filename for the players in range in the 6.cell
% 8. set the range in the 6.cell
% 9. set the filename in the 7.cell for the file with the 2 matrices
%10. set the desired players to face each other
%11. run the whole file

% hint: just run one cell, if only this result is desired
% warning: the more things you want to plot, the more plots you got

%##########################################################################

%% Initialize and get data of the simulation file

% Clear used Variables:
clear filename vars rewardMatrix numberOfPlayers numberOfTurns listOfPlayers ...
    noise averageCoop lengthOfNoise i j k


% Inputs:
filename = 'simulation.mat';                  % name of the simulation-file
nummberOfSimulation = 1;                      % number of simulation
% Calc:
vars=load(filename);                          % load variables of the simulation-file
figureCounter = 1;                            % open new figure for each plot

rewardMatrix = vars.Rewardmatrix;             % store the rewardmatrix
numberOfTurns = vars.N;                       % store the numbers of turns
listOfPlayers = vars.Names;                   % store the list of players
noise = vars.Noise;                           % store the noisematrix
averageCoop = vars.AverageCoop;               % store average cooperation
short = vars.Shorts;                          % store short names of players

numberOfPlayers = length(listOfPlayers);      % get the numbers of players
lengthOfNoise = size(noise,2);

% Convert rewardmatrix
for i = 1:numberOfPlayers                  % generate empty matrices for every player
    eval(['R' int2str(i) '=zeros(lengthOfNoise,lengthOfNoise,numberOfPlayers);']);
    eval(['C' int2str(i) '=zeros(lengthOfNoise,lengthOfNoise,numberOfPlayers);']);
```

```
end

for i = 1:numberOfPlayers                  % rewardematrix Ri(noise1,noise2,oponent)
    for k = 1:lengthOfNoise
        for j = 1:lengthOfNoise
            eval(['R' int2str(i) '(' int2str(k) ',' int2str(j) ',:)='...
                'rewardMatrix(' int2str(i) ',:,' int2str(k) ',' int2str(j) ');']);
        end
    end

end

for i = 1:numberOfPlayers                     % coopmatrix Ci(noise1,noise2,oponent)
    for k = 1:lengthOfNoise
        for j = 1:lengthOfNoise
           eval(['C' int2str(i) '(' int2str(k) ',' int2str(j) ',:)=averageCoop'...
                '(' int2str(k) ',' int2str(j) ',' int2str(i) ',:);']);
        end
    end

end


%% Plot Reward of all players with given noiselevels

% Clear used Variables:
clear noiseLevel tempRewardMatrix rewardVectors i k h lengthN

% Inputs:
noiseLevel = [1;...                    % Noise Level 1 (player --> opponent)
              1];                      % Noise Level 2 (opponent --> player)


% Calc:
tempRewardMatrix = zeros(numberOfPlayers);  % temporary rewardmatrix
lengthN = size(noiseLevel,2);      % number of diffrent noise level constellations
rewardVectors = zeros(lengthN,numberOfPlayers);% the reward vector for each
                                               % noise level constellation is saved


h = figure(figureCounter);                                % initialize figure

set(h,'NumberTitle','off')
set(h,'Position',[10 100 1000 600])         % position and size of figure
set(h,'Name',['Reward of all Players at given Noiselevels ' int2str(1)])
                                            % set title of figure
```

```
for i = 1:lengthN                          % iterate over all noise lever constellations
    for k = 1:numberOfPlayers                  % iterate over all players
        eval(['tempRewardMatrix(' int2str(k) ',:)=R' int2str(k) ...
        '(noiseLevel(1,i),noiseLevel(2,i),:);']); % save reward of each player in
                                                  % temporary rewardmatrix
    end
    rewardVectors(i,:)=sum(tempRewardMatrix')/(numberOfTurns*numberOfPlayers);
                             % rewardvector of each noise level constellation is saved

    subplot(lengthN,1,i)                       % plotting options
    bar(rewardVectors(i,:));
    grid on;
    set(gca,'XTick',1:1:numberOfPlayers)
    set(gca,'XTickLabel',short,'FontSize',8)
    set(gca,'XLim',[0 numberOfPlayers+1])
    set(gca,'YLim',[max((min(rewardVectors(i,:))-0.25),0) min((max(...
        rewardVectors(i,:))+0.25),5)])
    title(['Noiseplot with Noiselevel 1: ',num2str(noise(1,noiseLevel(1,i))),...
        ', and Noiselevel 2: ', num2str(noise(2,noiseLevel(2,i)))],'FontWeight'...
        ,'bold','FontSize',12);
    xlabel('Players','FontWeight','bold','FontSize',10)
    ylabel('Average profit of Player','FontWeight','bold','FontSize',10)

end
saveas(h,['pics\simulation' num2str(nummberOfSimulation) '\'get(h,'Name') '.eps'])
figureCounter = figureCounter + 1;  % update figurecounter

%% Plot Cooperation of all players with given noiselevels

% Clear used Variables:
clear noiseLevel tempCoopMatrix coopectors i k h lengthN

% Inputs:
noiseLevel = [1 2;...                      % Noise Level 1 (player --> opponent)
              1 2];                        % Noise Level 2 (opponent −−> player)


% Calc:
tempCoopMatrix = zeros(numberOfPlayers);  % temporary rewardmatrix
lengthN = size(noiseLevel,2);       % number of diffrent noise level constellations
coopVectors = zeros(lengthN,numberOfPlayers); % the cooperation vector for each
                                              % noise level constellation is saved


h = figure(figureCounter);                                 % initialize figure
```

```
set(h,'NumberTitle','off')
set(h,'Position',[10 100 900 600])         % position and size of figure
set(h,'Name','Cooperation of all Players at given Noiselevels')  % set title of
                                                 % figure

for i = 1:lengthN                         % iterate over all noise lever constellations
    for k = 1:numberOfPlayers             % iterate over all players
        eval(['tempCoopMatrix(' int2str(k) ',:)=C' int2str(k) ...
            '(noiseLevel(1,i),noiseLevel(2,i),:);']); % save cooperation of each
                                                % player in temporary cooperation matrix
    end
    coopVectors(i,:)=mean(tempCoopMatrix,2); % coopvector of each noise level
                                             % constellation is saved


    subplot(lengthN,1,i)                     % plotting options
    bar(coopVectors(i,:));
    grid on;
    set(gca,'XTick',1:1:numberOfPlayers)
    set(gca,'XTickLabel',short,'FontSize',8)
    set(gca,'XLim',[0 numberOfPlayers+1])
    set(gca,'YLim',[max((min(coopVectors(i,:))-0.05),0) min((...
        max(coopVectors(i,:))+0.05),1)])
    title(['Noiseplot with Noiselevel 1: ',num2str(noise(1,noiseLevel(1,i))),...
        ', and Noiselevel 2: ', num2str(noise(2,noiseLevel(2,i)))],'FontWeight'...
        ,'bold','FontSize',12);
    xlabel('Players','FontWeight','bold','FontSize',10)
    ylabel('Cooperation of Player in %','FontWeight','bold','FontSize',10)
end
saveas(h,['pics\simulation' num2str(nummberOfSimulation) '\'get(h,'Name') '.eps'])
figureCounter = figureCounter + 1;  % update figurecounter

%% Plot statistics for a given player (Reward vs given Noiselevels)

% Clear used Variables:
clear position noiseLevel lengthN givenPlayers tempRewardMatrix k ...
    tempRewardVector i h

% Inputs:
position = [1];                 % Numbers of the players (hint: type listOfPlayers
                                % to see which player has which number)
noiseLevel = [1  ;...          % Noise Level 1
              1];              % Noise Level 2

% Calc:
givenPlayers = length(position);    % number of given players
```

```
lengthN = size(noiseLevel,2);        % number of given noise level constellations
tempSurf = zeros(lengthOfNoise);

tempRewardMatrix = zeros(givenPlayers,numberOfPlayers,lengthN);
                                                    % temporary reward matrix
tempRewardVector = zeros(1,numberOfPlayers);

for i = 1:givenPlayers               % fill tempRewardMatrix(given Player, all
                                              % opponents, given noise level)
    for k = 1:numberOfPlayers
        for l = 1:lengthN
            tempRewardMatrix(i,k,l) = eval(['R' int2str(position(i))...
                '(noiseLevel(1,l),noiseLevel(2,l),k);']);
        end
    end
end

for i = 1:givenPlayers               % iterate over all given players
    h = figure(i+figureCounter);                        % initialize figure
    set(h,'NumberTitle','off')
    set(h,'Position',[10 100 800 720])         % position and size of figure
    set(h,'Name',['Reward of Player ' listOfPlayers{position(i)} ...
        ' against all Players at given Noiselevels'])  % set title of figure
    for k = 1:lengthN                % iterate over each noiselevel constellation
        tempRewardVector = tempRewardMatrix(i,:,k)/numberOfTurns;
                                  % take right vector out of the tempRewardMatrix

        subplot(lengthN,1,k)         % plotting options
        bar(tempRewardVector);
        grid ON;
        set(gca,'XTick',1:1:numberOfPlayers)
        set(gca,'XTickLabel',short,'FontSize',8)
        set(gca,'XLim',[0 numberOfPlayers+1])
        set(gca,'YLim',[max((min(tempRewardVector)-0.25),0)...
            min((max(tempRewardVector)+0.25),5)])
        title(['Noiseplot with Noiselevel 1: ',num2str(noise(1,noiseLevel...
            (1,k))),' and Noiselevel 2: ', num2str(noise(2,noiseLevel(2,k))),...
            ' for Player ' listOfPlayers{position(i)}, ''],'FontWeight','bold'...
            ,'FontSize',12);
        xlabel('Opponents','FontWeight','bold','FontSize',10)
        ylabel(['Average profit of Player ' listOfPlayers{position(i)} ''],...
            'FontWeight','bold','FontSize',8)
    end
%saveas(h,['pics\simulation'num2str(nummberOfSimulation) '\'get(h,'Name') '.eps'])
end
```

```
for i = 1:givenPlayers                    % iterate over all given players

    tempSurf = sum(eval(['R' int2str(position(i))]),3)/...
        (numberOfPlayers*numberOfTurns);

    h = figure(figureCounter+givenPlayers+i);                  % initialize figure
    set(h,'NumberTitle','off')
    set(h,'Position',[10 100 700 700])          % position and size of figure
    set(h,'Name',['Reward vs Noise of Player ' listOfPlayers{position(i)} ''])
                                                              % set title of figure


    surf(tempSurf)
    title(['Average profit of Player ' listOfPlayers{position(i)} ''],...
        'FontWeight','bold','FontSize',12);

    % set a colormap for the figure.
    colormap(hot);

    % set the view angle.
    view(150,47);

    % labels
    set(gca,'XTick',1:1:lengthOfNoise)
    set(gca,'YTick',1:1:lengthOfNoise)
    set(gca,'XTickLabel',noise(2,:))
    set(gca,'YTickLabel',noise(1,:))


    xlabel('Noise 2');
    ylabel('Noise 1');
    zlabel('Reward');
saveas(h,['pics\simulation' num2str(nummberOfSimulation) '\'get(h,'Name') '.eps'])
end

figureCounter = figureCounter + 2* givenPlayers; % update figurecounter

%% Plot statistics for a given player (Cooperation vs given Noiselevels)

% Clear used Variables:
clear position noiseLevel lengthN givenPlayers tempCoopMatrix k tempCoopVector i h

% Inputs:
position = [1,2,3,4,5];                    % Numbers of the players (hint: type
                                   % listOfPlayers to see which player has which number)
noiseLevel = [1  ;...          % Noise Level 1
```

```
            1];             % Noise Level 2

% Calc:
givenPlayers = length(position);     % number of given players
lengthN = size(noiseLevel,2);        % number of given noise level constellations

tempCoopMatrix = zeros(givenPlayers,numberOfPlayers,lengthN);
                                                % temporary cooperation matrix
tempCoopVector = zeros(1,numberOfPlayers);
tempSurf = zeros(lengthOfNoise);

for i = 1:givenPlayers               % fill tempCoopMatrix(given Player, all
                                     % opponents, given noise level)
    for k = 1:numberOfPlayers
        for l = 1:lengthN
            tempCoopMatrix(i,k,l) = eval(['C' int2str(position(i)) ...
                '(noiseLevel(1,l),noiseLevel(2,l),k);']);
        end
    end
end

for i = 1:givenPlayers               % iterate over all given players
    h = figure(i+figureCounter);                        % initialize figure
    set(h,'NumberTitle','off')
    set(h,'Position',[10 500 1000 900])        % position and size of figure
    set(h,'Name',['Cooperation of Player ' listOfPlayers{position(i)} ...
        ' against all Players at given Noiselevels'])  % set title of figure
    for k = 1:lengthN                % iterate over each noiselevel constellation
        tempCoopVector = tempCoopMatrix(i,:,k); % take right vector
                                                % out of the tempCoopMatrix

        subplot(lengthN,1,k)         % plotting options
        bar(tempCoopVector);
        grid ON;
        set(gca,'XTick',1:1:numberOfPlayers)
        set(gca,'XTickLabel',short,'FontSize',8)
        set(gca,'XLim',[0 numberOfPlayers+1])
        set(gca,'YLim',[max((min(tempCoopVector)-0.05),0)...
            min((max(tempCoopVector)+0.05),1)])
        title(['Noiseplot with Noiselevel 1: ',num2str(noise(1,noiseLevel...
            (1,k))),' and Noiselevel 2: ', num2str(noise(2,noiseLevel(2,k)))...
            ,' for Player ' listOfPlayers{position(i)}, ''],'FontWeight','bold'...
            ,'FontSize',12);
        xlabel('Opponents','FontWeight','bold','FontSize',10)
        ylabel(['Cooperation of Player ' listOfPlayers{position(i)} ''],...
            'FontWeight','bold','FontSize',8)
```

```
    end
saveas(h,['pics\simulation' num2str(nummberOfSimulation) '\'get(h,'Name') '.eps'])
end
for i = 1:givenPlayers                  % iterate over all given players

    tempSurf = mean(eval(['C' int2str(position(i))]),3);

    h = figure(figureCounter+givenPlayers+i);                    % initialize figure
    set(h,'NumberTitle','off')
    set(h,'Position',[10 500 800 800])         % position and size of figure
    set(h,'Name',['Cooperation vs Noise of Player ' listOfPlayers{position(i)}])
                                                    % set title of figure


    surf(tempSurf)
    title(['Average cooperation of Player ' listOfPlayers{position(i)} ''],...
        'FontWeight','bold','FontSize',12);

    % set a colormap for the figure.
    colormap(hot);

    % set the view angle.
    view(225,35);

    % labels
    set(gca,'XTick',0:1:lengthOfNoise)
    set(gca,'YTick',0:1:lengthOfNoise)
    set(gca,'XTickLabel',noise(1,:))
    set(gca,'YTickLabel',noise(2,:))


    xlabel('Noise 1');
    ylabel('Noise 2');
    zlabel('Cooperation');
saveas(h,['pics\simulation' num2str(nummberOfSimulation) '\'get(h,'Name') '.eps'])
end

figureCounter = figureCounter + 2* givenPlayers; % update figurecounter

%% Reward vs Noise with name of the best player

% Clear used Variables:
clear positions h tempRewardMatrix value position player noiseLevel ...
    tempPositions endPositions endReward playersInRange range filename file

% Inputs:
```

```
playersInRange = true;  % true: calculate players in range, false, don't calculate
                        % players in range
range = 0.05;            % how close have other players be, to be mentioned
filename = 'range.txt'; % file, where players in range are saved

% Calc:
positions = zeros(lengthOfNoise^2,numberOfPlayers);            % vector for player
                                                  % with maximum reward for given noise
tempRewardMatrix = zeros(lengthOfNoise^2,numberOfPlayers);     % create new
                                                            % temporary reward matrix

for i = 1:lengthOfNoise                           % iterate over all noise combinations
    for k = 1:lengthOfNoise
        for l = 1:numberOfPlayers                             % iterate over all players
            for m = 1:numberOfPlayers                        % iterate over all opponents
            tempRewardMatrix(k+(i-1)*lengthOfNoise,l) = tempRewardMatrix(k+...
                (i-1)*lengthOfNoise,l) + eval(['R' int2str(l) '(' int2str(i)...
                ',' int2str(k) ',' int2str(m)   ');']);
                                              % add temporary rewardmatrix (1,player)
            end
        end
    end
end

[value, position] = max(tempRewardMatrix'/(numberOfTurns*numberOfPlayers));
                                                                        % take maximas

for i = 1:lengthOfNoise^2                        % fill positionmatrix
    positions(i,position(i)) =  value(i);
end

[noiseLevel ,player] = find(positions);
tempPositions = sortrows([noiseLevel player],1);

for i = 1:lengthOfNoise% get positions matrix and reward matrix ready for plotting
    for k = 1:lengthOfNoise
        endPositions(i,k) = tempPositions(k+(i-1)*lengthOfNoise,2);
        endReward(i,k) = positions(k+(i-1)*lengthOfNoise,tempPositions(...
            k+(i-1)*lengthOfNoise,2));
    end
end

h = figure(figureCounter+1);                                 % initialize figure
set(h,'NumberTitle','off')
set(h,'Position',[10 500 800 800])          % position and size of figure
set(h,'Name','Reward vs Noise with best Player named')  % set title of figure
```

```
colormap(winter)
imagesc(0:1:lengthOfNoise-1,0:1:lengthOfNoise-1,endReward)
set(gca,'XTick',0:1:lengthOfNoise)
set(gca,'YTick',0:1:lengthOfNoise)
set(gca,'XTickLabel',noise(1,:))
set(gca,'YTickLabel',noise(2,:))

for i = 1:lengthOfNoise
    for k = 1:lengthOfNoise
        text(k-1,i-1,...
        [listOfPlayers{endPositions(i,k)}],...
        'HorizontalAlignment','center','VerticalAlignment','bottom',...
        'FontWeight','bold','FontSize',12);
        text(k-1,i-1,...
        [num2str(endReward(i,k))],...
        'HorizontalAlignment','center','VerticalAlignment','top');


   end
end
saveas(h,['pics\simulation' num2str(nummberOfSimulation) '\'get(h,'Name') '.eps'])



if(playersInRange)                          % caluclate players in range:
    result=zeros(lengthOfNoise^2,numberOfPlayers); % empty matrix for position
                                                   % of players
    tempRewardMatrix = tempRewardMatrix./(numberOfPlayers*numberOfTurns);
                                                   % norm tempRewardMatrix
    lowerValue = endReward .* (1-range);           % calculate lower value
    for i = 1:lengthOfNoise                        % iterate over all noise levels
        for k = 1:lengthOfNoise
            clear tempResult
            tempResult = find(tempRewardMatrix(k+(i-1)*lengthOfNoise,:)>=...
                lowerValue(i,k));  % find players in range
            result(k+(i-1)*lengthOfNoise,1:length(tempResult)) = tempResult;
        end
    end


file = fopen(filename,'w');  % open file with given filename
fprintf(file, 'Players in a %1.2f range for each noise level \n\n',range);
                                                                  % print header

for i = 1:lengthOfNoise                                           % print file
```

```
    for k = 1:lengthOfNoise
        fprintf(file, 'Noise level 1: %1.2f, Noise level 2: %1.2f',...
            noise(1,i),noise(2,k));
        fprintf(file,', highest reward: %1.4f, in range (>%1.4f):\n',...
            endReward(i,k),lowerValue(i,k));
        for l=find(result(k+(i-1)*lengthOfNoise,:))
            fprintf(file, '%s (%1.4f)\n',listOfPlayers{result(k+(i-1)*...
                lengthOfNoise,l)},tempRewardMatrix(k+(i-1)*lengthOfNoise,...
                result(k+(i-1)*lengthOfNoise,l)));
        end
        fprintf(file, '\n');
    end
end

fclose(file);                                       % close file

end
figureCounter = figureCounter + 2;                  % update figurecounter


%% Total Cooperation/Reward normed

% Clear used Variables:
clear i k totalReward totalCoop tempTotalCoop filename file

% Inputs:
filename = 'totalresult.txt';                  % filename of file for total results

% Calc:

totalReward = zeros(lengthOfNoise);  % create total reward matrix
totalCoop = zeros(lengthOfNoise);         % create total cooperation matrix

for k=1:numberOfPlayers                        % iterate over all players
    for i=1:numberOfPlayers         % calculate total reward matrix
        totalReward(:,:)=totalReward(:,:)+eval(['R' int2str(k) '(:,:,' ...
            int2str(i) ')'  ';'])/(numberOfPlayers*numberOfTurns*numberOfPlayers);

    end
    for i=1:lengthOfNoise           % calculate temporary total cooperation matrix
        for j=1:lengthOfNoise
            tempTotalCoop(i,j,k)=mean(eval(['C' int2str(k) '(i,j,:)'  ';']));
        end
    end
end

```

```
for l=1:lengthOfNoise                   % calculate total cooperation matrix
    for j=1:lengthOfNoise
        totalCoop(l,j)=mean(tempTotalCoop(l,j,:));
    end
end

h = figure(figureCounter+1);                            % initialize figure
set(h,'NumberTitle','off')
set(h,'Position',[10 500 800 800])          % position and size of figure
set(h,'Name','Total Reward vs Noise')  % set title of figure


surf(totalReward)


% set a colormap for the figure.
colormap(hot);

% set the view angle.
view(225,35);

% labels
set(gca,'XTick',0:1:lengthOfNoise)
set(gca,'YTick',0:1:lengthOfNoise)
set(gca,'XTickLabel',noise(1,:))
set(gca,'YTickLabel',noise(2,:))


xlabel('Noise 1');
ylabel('Noise 2');
zlabel('Reward');
saveas(h,['pics\simulation' num2str(nummberOfSimulation) '\'get(h,'Name') '.eps'])
h = figure(figureCounter+2);                            % initialize figure
set(h,'NumberTitle','off')
set(h,'Position',[10 500 800 800])          % position and size of figure
set(h,'Name','Total Cooperation vs Noise')  % set title of figure


surf(totalCoop)


% set a colormap for the figure.
colormap(jet);

% set the view angle.
view(225,35);
```

```
% labels
set(gca,'XTick',0:1:lengthOfNoise)
set(gca,'YTick',0:1:lengthOfNoise)
set(gca,'XTickLabel',noise(1,:))
set(gca,'YTickLabel',noise(2,:))


xlabel('Noise 1');
ylabel('Noise 2');
zlabel('Cooperation');
saveas(h,['pics\simulation' num2str(nummberOfSimulation) '\'get(h,'Name') '.eps'])
file = fopen(filename,'w');     % open file with given filename
fprintf(file, 'Total Rewardmatrix: \n\nNoise ',range);
                                                  % print header for rewardmatrix

fprintf(file, '| %1.2f ',noise(1,:));              % print reward matrix
fprintf(file, '\n -----|');
for k=1:lengthOfNoise
    for i = 1:lengthOfNoise
        fprintf(file, '-------');
    end
    fprintf(file, '\n %1.2f ',noise(2,k));
    fprintf(file, '| %1.2f ',totalReward(k,:));
    fprintf(file, '\n -----|');
end
for i = 1:lengthOfNoise
    fprintf(file, '-------');
end

fprintf(file, '\n\nTotal Cooperatiomatrix: \n\nNoise ',range);
                                                   % print header for coopmatrix

fprintf(file, '| %1.2f   ',noise(1,:));                   % print coopmatrix
fprintf(file, '\n -----|');
for k=1:lengthOfNoise
    for i = 1:lengthOfNoise
        fprintf(file, '---------');
    end
    fprintf(file, '\n %1.2f ',noise(2,k));
    fprintf(file, '| %1.4f ',totalCoop(k,:));
    fprintf(file, '\n -----|');
end
for i = 1:lengthOfNoise
    fprintf(file, '---------');
end
```

```
fclose(file);                                                    % close file

figureCounter=figureCounter+2;

%% 2 given Players against each other

% Clear used Variables:
clear players shortTemp tempRewardMatrix l k i

% Inputs:
player = [1 ;...                    % player 1
          1 ] ;                     % player 2


% Calc:
players = size(player,2);           % number of faceoffs
tempRewardMatrix = zeros(lengthOfNoise^2,2,players);
                                    % create temporary rewardmatrix(noiselevel,2,faceoff)


for l = 1:players                   % create rewardmatrix
    for i = 1:lengthOfNoise
        for k = 1:lengthOfNoise
            tempRewardMatrix(k+(i-1)*lengthOfNoise,1,l) = eval(['R' int2str...
                (player(1,l)) '(' int2str(i) ',' int2str(k) ',' int2str...
                (player(2,l)) ');'])/numberOfTurns;
            tempRewardMatrix(k+(i-1)*lengthOfNoise,2,l) = eval(['R' int2str...
                (player(2,l)) '(' int2str(i) ',' int2str(k) ',' int2str...
                (player(1,l)) ');'])/numberOfTurns;
        end
    end
end

for l = 1:players                   % iterate over faceoffs
    h = figure(l+figureCounter);                              % initialize figure
    set(h,'NumberTitle','off')
    set(h,'Position',[10 500 1600 900])         % position and size of figure
    set(h,'Name',['' listOfPlayers{player(1,l)} ' against ' listOfPlayers...
        {player(2,l)} ' and vice versa'])  % set title of figure
    for i = 1:lengthOfNoise
        for k = 1:lengthOfNoise
            subplot(lengthOfNoise, lengthOfNoise, k+(i-1)*lengthOfNoise)
            bar(tempRewardMatrix(k+(i-1)*lengthOfNoise,:,l))
            grid ON;
            set(gca,'XTick',1:1:2)
```

```
            shortTemp{1} = short{player(1,l)};
            shortTemp{2} = short{player(2,l)};
            set(gca,'XTickLabel',shortTemp,'FontSize',8)
            set(gca,'XLim',[0 3])
            set(gca,'YLim',[max((min(tempRewardMatrix(k+(i-1)*...
                lengthOfNoise,:,l))-0.25),0) min((max(tempRewardMatrix(...
                k+(i-1)*lengthOfNoise,:,l))+0.25),5)])
            title(['Noiselv 1: ',num2str(noise(1,k)),' and Noiselv 2: ',...
                num2str(noise(1,i)),],'FontWeight','bold','FontSize',12);
            xlabel('Opponents','FontWeight','bold','FontSize',10)
            ylabel(['Reward'],'FontWeight','bold','FontSize',8)
        end
    end
saveas(h,['pics\simulation' num2str(nummberOfSimulation) '\'get(h,'Name') '.eps'])
end
```

B.4 playerlist.m

Listing 4: playerlist.m

```
function [ Liste ] = playerlist(player, maxplayers)
%PLAYERLIST: imports the players
%Input: "player": A string, which is equal to the Name of the Players
%Input: "maxplayers" the maximum of allowed players

% Output:
% If a "playerxx" exists, the value xx of the Vector "Liste" becomes 1
% If a "playerxx" doesn't exist, the value xx becomes 0


Liste=1;

for i=1:maxplayers
    i2=int2str(i);
    Pruefbed = strcat(player, i2);
    if exist(Pruefbed)==2
        Liste(i)=1;
    else
        Liste(i)=0;
    end
end
```

B.5 player1.m

Listing 5: player1.m

```
classdef player1
properties
    name = 'Cooperate';
    short = 'COOP';
end
methods
    function P1 = player1(np)
    end
    function decision=decide(obj,K,op,turn)
        decision=1;
    end
end
end
```

B.6 player2.m

Listing 6: player2.m

```
classdef player2
properties
    name='Defect';
    short='DEF';
end
methods
    function P2 = player2(np)
    end
    function decision=decide(obj,K,op,turn)
        decision=2;
    end
end
end
```

B.7 player3.m

Listing 7: player3.m

```
classdef player3
properties
    name='Random';
    short='RAN';
end
methods
    function P3 = player3(np)
```

```
    end
    function decision=decide(obj,K,op,turn)
        if (rand>0.5)
            decision=1;
        else
            decision=2;
        end
    end
end
end
```

B.8 player4.m

Listing 8: player4.m

```
classdef player4
properties
    name='Tit for tat';
    short='TFT';
end
methods
    function P4 = player4(np)
    end
    function decision=decide(obj,K,op,turn)
        if (turn == 1)
            decision = 1; %cooperate in turn 1
        elseif (K(op,4,turn-1) == 1)
            decision = 1;
        else
            decision = 2;
        end
    end
end
end
```

B.9 player5.m

Listing 9: player5.m

```
classdef player5
properties
    name='Friedmann';
    short='FRI';
end
methods
```

```
    function P5 = player5(np)
    end
    function decision=decide(obj,K,op,turn)
    if (turn == 1)
        decision = 1; %cooperate in turn 1
    elseif (max(K(op,5,:)) == 2) % was betrayed once
        decision = 2;
    else
        decision = 1;
    end
    end
end
end
```

B.10 player6.m

Listing 10: player6.m

```
classdef player6
properties
    name='Pavlov';
    short='PAV';
end
methods
    function P6 = player6(np)
    end
    function decision=decide(obj,K,op,turn)
    if (turn == 1)
        decision = 1; %cooperate in turn 1
    elseif (K(op,6,turn-1) == 1) % he cooperates, that means the stretagy is
      continued
        decision = K(6,op,turn-1);
    else                        % He betrayed therefore the strategy is changed
        if (K(6,op,turn-1) == 1)
            decision = 2;
        else
            decision = 1;
        end
    end
    end
end
end
```

B.11 player7.m

Listing 11: player7.m

```
classdef player7
properties
    name='Tit for 2tat';
    short='TF2T';
end
methods
    function P7 = player7(np)
    end
    function decision=decide(obj,K,op,turn)
    if (turn == 1)
        decision = 1; %cooperate in turn 1
    elseif (turn ==2 )
        decision = 1;
    elseif (K(op,7,turn-1) == 1 || K(op,7,turn-2) == 1)
        decision = 1;
    else
        decision = 2;
    end
    end
    end
end
```

B.12 player8.m

Listing 12: player8.m

```
classdef player8
properties
    name='Joss';
    short='JOSS';
    r=0.1; %random rejection chance
    playernumber=8;
end
methods
    function P8 = player8(np)
    end
    function decision=decide(obj,K,op,turn)
    if (turn == 1)
        decision = 1; % cooperate in turn 1
        if (rand < obj.r) % insert random defections
            decision=2;
        end
    else
        if (K(op,obj.playernumber,turn-1) == 1)
```

```
            if (rand < obj.r) % insert random defections
                decision=2;
            else
                decision=1;
            end
        else
            decision = 2;
        end
    end
    end
end
end
```

B.13 player9.m

Listing 13: player9.m

```
classdef player9
properties
    name='Diekmann'; %source:www.socio.ethz.ch/vlib/pesb/pesb9.pdf
    short='DIE';
end
methods
    function P9 = player9(np)
    end
    function decision=decide(obj,K,op,turn)
        if (turn == 1)
            decision = 1; % cooperate in turn 1
        else
            if (K(op,9,turn-1) == 1)
                decision = 1;
            elseif (mod(turn,10)==0) %insert two cooperative moves every ten moves
                decision=1;
            elseif (mod(turn,10)==1) %insert two cooperative moves every ten moves
                decision=1;
            else
                decision = 2;
            end
        end
    end
end
end
```

B.14 player10.m

Listing 14: player10.m

```
classdef player10
    %created by Meier David
properties
    name='Tit for average tat';
    short='TFAT';
    mem=5; %how many moves does the player remember
    playernumber=10; %the number of the player
    erase=50; %erase memory after this amount of turns
end
methods
    function P10 = player10(np)
    end
    function decision=decide(obj,K,op,turn)
        if (mod(turn,obj.erase)<obj.mem+2) %play tft in the first rounds
            if (mod(turn,obj.erase) == 1)
                decision = 1; %cooperate in turn 1
            elseif (K(op,obj.playernumber,turn-1) == 1)
                decision = 1;
            else
                decision = 2;
            end
        else
            if (sum(K(op,obj.playernumber,turn-obj.mem:turn-1))/obj.mem<=1.5) %
              averaged decision over 10 turns is cooperative
                decision=1;
            else
                decision=2;
            end
        end
    end
end
end
```

B.15 player11.m

Listing 15: player11.m

```
classdef player11  < handle
    %created by Samuel Andermatt (the idea is rather straightforward, so in
    %case somebody has had this idea before I apologize)
    %The idea is that you basically go for TFT, but try to avoid to enter a
    %state where players reject each other over and over again.
    %Therefore you will try to reconcile, as soon as the rejections on both
    %sides caused enough damage to the other player to avoid beeing
```

```
    %exploitable
properties
    name='Reconcilation TFT';
    short='RTFT';
    playernumber=11;
    k=zeros(1); %this is a number that allows the object to switch between a
      recooonciling state and a TFT state. 0 means TFt, 1 means reconcile.
    memory=20; %determines how many turns you go back at max
end
methods
    function P11 = player11(np)
        P11.k=zeros(np,1);
    end
    function decision=decide(obj,K,op,turn)
        if (turn == 1)
            decision = 1; %cooperate in turn 1
        elseif (obj.k(op)==1) %player is in reconciling state
            decision = 1;
            obj.k(op)=0; %Go back to TFT state
        elseif (K(op,obj.playernumber,turn-1) == 1)  %cooperation is always met
          with cooperation
            decision = 1;
        else
            %calculate if the conditions are met to enter reconciling state
            winCoop=0; %the winnings the opponent had if he cooperated
            winReject=0; %the winnings he made by rejecting
            %calculate the winnings B can get by exploiting the
            %reconcilation attempt, a reconcilation attempt is two
            %consecutive cooperative steps
            C=win([2 1]);
            RecT=2*C(1);
            for i=turn-1:-1:turn-obj.memory-1
                if(i<1)
                    continue; %there are no turns before the first turn
                end
                A=win([1 1]); %winnings for cooperation
                winCoop=winCoop+A(1); %the points he would have won by cooperating
                B=win([K(op,obj.playernumber,i) K(obj.playernumber,op,i)]); %the
                  points won by rejecting
                winReject=winReject+B(1); %the points the opponent actually won
                  trhough rejection
                if (winCoop>winReject+RecT) %Cooperation would have bben better for
                   the opponent
                    obj.k(op)=1;
                    decision=1;
                    break;
```

```
                end
            end
            if (obj.k(op)==0) %Criteria for reconcilation have not been met
                decision = 2;
            end
        end
    end
end
end
```

B.16 player12.m

Listing 16: player12.m

```
classdef player12 < handle
properties
    name='CDowning';
    short='CDO';
    n_c_cd=0;            % number of cases with oponent: c, downing: c
    n_c_dd=0;            % number of cases with oponent: c, downing: d
    n_cd=0;              % number of cases with downing: c
    n_dd=0;              % number of cases with downing: d
    playernumber = 12;

end
methods
    function P12 = player12(np)
        P12.n_c_cd=zeros(np,1);
        P12.n_c_dd=zeros(np,1);
        P12.n_cd=zeros(np,1);
        P12.n_dd=zeros(np,1);
    end

    function decision=decide(obj,K2,op,turn)
        if (turn == 1)
            decision = 1;
        else
            [obj.n_c_cd, obj.n_c_dd, obj.n_cd, obj.n_dd] = update_rounds(obj, obj.
              n_c_cd, obj.n_c_dd, obj.n_cd, obj.n_dd, K2, op, turn);
            p_c_cd=obj.n_c_cd(op)/(turn-1);
            p_c_dd=obj.n_c_dd(op)/(turn-1);
            p_cd=obj.n_cd(op)/(turn-1);
            p_dd=obj.n_dd(op)/(turn-1);

            if (p_cd == 0)
```

```
                p1=0.5;
            else
                p1=p_c_cd/p_cd;
            end
            if (p_dd == 0)
                p2 = 0.5;
            else
                p2=p_c_dd/p_dd;
            end

            E1 = p1*3 + (1-p1) * 0;
            E2 = p2*5 + (1-p2) * 1;

            if(E2>E1)
                decision = 2;
            else
                decision = 1;
            end

        end
    end

    function [n_c_cd_new, n_c_dd_new, n_cd_new, n_dd_new] = update_rounds(obj,
      n_c_cd_old, n_c_dd_old, n_cd_old,n_dd_old, K, op, turn)
        n_c_cd_new = n_c_cd_old;
        n_c_dd_new = n_c_dd_old;
        n_cd_new = n_cd_old;
        n_dd_new = n_dd_old;

        if (K(op,obj.playernumber,turn-1) == 1)
            if(K(obj.playernumber,op,turn-1) == 1)
                n_c_cd_new(op) = n_c_cd_old(op) + 1;
            else
                n_c_dd_new(op) = n_c_dd_old(op) + 1;
            end
        end
        if (K(obj.playernumber,op,turn-1) == 1)
            n_cd_new(op) = n_cd_old(op) + 1;
        else
            n_dd_new(op) = n_dd_old(op) + 1;
        end
    end
end
end
```

B.17 player13.m

Listing 17: player13.m

```
classdef player13
    %created by samuel andermatt
    %this is a player that takes the decisions to other players into
    %account (works with signaling)
properties
    name='TFT with Reputation';
    short='TFTR';
    playernumber=13;
    threshold=0.85; %number of cooperations that have to be made with other players
       on average to ensure cooperation
end
methods
    function P13 = player13(np)
    end
    function decision=decide(obj,K,op,turn)
        if (turn == 1)
            decision = 1; %cooperate in turn 1
        elseif (K(op,obj.playernumber,turn-1) == 1)
            decision = 1;
        elseif (mean(K(op,:,turn-1))-1 < (1-obj.threshold)) %average of oponents
          decision is higher than threshold
            decision = 1;
        else
            decision = 2;
        end
    end
end
end
```

B.18 player14.m

Listing 18: player14.m

```
classdef player14 < handle
    %created by Samuel Andermatt
    %this is an attempt to create a simple learning player
properties
    name='Strategy Switcher';
    short='SSW';
    playernumber=14;
    s=zeros(1,1); %current strategy
    lastS=zeros(1,1); %strategy of last turn
```

```
    strchange=20; %decides how many turns you wait until you change your strategy
    ts=zeros(5,1); %turnes spent in each strategy
    ps=zeros(5,1); %performance of each strategy
end
methods
    function P14 = player14(np)
        P14.s=zeros(np,1)+1; %start with strategy 1
        P14.lastS=zeros(np,1)+1;
        P14.ts=zeros(5,np);
        P14.ps=zeros(5,np);
    end
    function decision=decide(P14,K,op,turn)
        if (turn==1) %cooperate in turn 1
            decision=1;
        elseif (P14.s(op) == 1) %strategy one is active
            %TFT
            if (K(op,P14.playernumber,turn-1) == 1)
                decision = 1;
            else
                decision = 2;
            end
        elseif (P14.s(op) == 2)
            %TF2T
            if (K(op,P14.playernumber,turn-1) == 1 || K(op,P14.playernumber,turn-2) == 1)
                decision = 1;
            else
                decision = 2;
            end
        elseif (P14.s(op) == 3)
            decision=2; %always defect
        elseif (P14.s(op) == 4)
            decision=1; %always cooperate
        else
            %pavlov
            if (K(op,6,turn-1) == 1) % he cooperates, that means the stretagy is continued
                decision = K(P14.playernumber,op,turn-1);
            else                      % He betrayed therefore the strategy is changed
                if (K(6,op,turn-1) == 1)
                    decision = 2;
                else
                    decision = 1;
                end
            end
```

```
        end

        %update ts and ps
        P14.ts(P14.s(op),op)=P14.ts(P14.s(op),op)+1; %one term more spent in
          strategy s
        if (turn>1)
            W=win([K(P14.playernumber,op,turn-1) K(op,P14.playernumber,turn-1)]); %
              winnings from last turn
            P14.ps(P14.lastS(op),op)=((P14.ts(P14.lastS(op),op)-1)*P14.ps(P14.lastS
              (op),op)+W(1))/P14.ts(P14.lastS(op),op); %average performance of
              strategy P14.lastS
        end

        %choose new strategy

        %evaluation phase

        P14.lastS(op)=P14.s(op); %the strategy from last turn is no longer needed,
          therefore it is updated here

        %in the first 100 turns experience is gained with all strategies
        if (turn == P14.strchange)
            P14.s(op)=2; %change to strategy 2
        elseif (turn == 2*P14.strchange)
            P14.s(op)=3;
        elseif (turn == 3*P14.strchange)
            P14.s(op)=4;
        elseif (turn == 4*P14.strchange)
            P14.s(op)=5;
            %Now 20 turns have been played with each strategy, the player
            %will now only play the most sucessful ones.
        elseif (turn >= 5*P14.strchange && mod(turn,P14.strchange) == 0) %initial
          testing phase ended, change strategies every 20 turns
            %this is a simple way to choose a strategy, he simply chooses
            %the one performing best
            [maxPF,maxInd]=max(P14.ps(:,op)); %maxInd is the index of the best
              performing strategy
            P14.s(op)=maxInd;
        else
        end
    end
end
end
```

B.19 player15.m

Listing 19: player15.m

```
classdef player15 < handle
properties
    name='DDowning';
    short='DDO';
    n_c_cd=0;           % number of cases with oponent: c, downing: c
    n_c_dd=0;           % number of cases with oponent: c, downing: d
    n_cd=0;             % number of cases with downing: c
    n_dd=0;             % number of cases with downing: d
    playernumber = 15;

end
methods
    function P15 = player15(np)
        P15.n_c_cd=zeros(np,1);
        P15.n_c_dd=zeros(np,1);
        P15.n_cd=zeros(np,1);
        P15.n_dd=zeros(np,1);
    end

    function decision=decide(obj,K2,op,turn)
        if (turn == 1)
            decision = 2;
        else
            [obj.n_c_cd, obj.n_c_dd, obj.n_cd, obj.n_dd] = update_rounds(obj, obj.
              n_c_cd, obj.n_c_dd, obj.n_cd, obj.n_dd, K2, op, turn);
            p_c_cd=obj.n_c_cd(op)/(turn-1);
            p_c_dd=obj.n_c_dd(op)/(turn-1);
            p_cd=obj.n_cd(op)/(turn-1);
            p_dd=obj.n_dd(op)/(turn-1);

            if (p_cd == 0)
                p1=0.5;
            else
                p1=p_c_cd/p_cd;
            end
            if (p_dd == 0)
                p2 = 0.5;
            else
                p2=p_c_dd/p_dd;
            end

            E1 = p1*3 + (1-p1) * 0;
            E2 = p2*5 + (1-p2) * 1;
```

```
            if(E2>E1)
                decision = 2;
            else
                decision = 1;
            end

        end
    end

    function [n_c_cd_new, n_c_dd_new, n_cd_new, n_dd_new] = update_rounds(obj,
      n_c_cd_old, n_c_dd_old, n_cd_old,n_dd_old, K, op, turn)
        n_c_cd_new = n_c_cd_old;
        n_c_dd_new = n_c_dd_old;
        n_cd_new = n_cd_old;
        n_dd_new = n_dd_old;
        if (K(op,obj.playernumber,turn-1) == 1)
            if(K(obj.playernumber,op,turn-1) == 1)
                n_c_cd_new(op) = n_c_cd_old(op) + 1;
            else
                n_c_dd_new(op) = n_c_dd_old(op) + 1;
            end
        end
        if (K(obj.playernumber,op,turn-1) == 1)
            n_cd_new(op) = n_cd_old(op) + 1;
        else
            n_dd_new(op) = n_dd_old(op) + 1;
        end
    end
end
end
```

B.20 player16.m

Listing 20: player16.m

```
classdef player16 < handle
properties
    name='LookBack_CDowning';
    short='LCDO';
    n_c_cd=0;           % number of cases with oponent: c, downing: c
    n_c_dd=0;           % number of cases with oponent: c, downing: d
    n_cd=0;             % number of cases with downing: c
    n_dd=0;             % number of cases with downing: d
    playernumber = 16;
```

```
end
methods
    function P16 = player16(np)
        P16.n_c_cd=zeros(np,1);
        P16.n_c_dd=zeros(np,1);
        P16.n_cd=zeros(np,1);
        P16.n_dd=zeros(np,1);
    end

    function decision=decide(obj,K2,op,turn)
        if (turn == 1 || turn == 2)
            decision = 1;
        else
            [obj.n_c_cd, obj.n_c_dd, obj.n_cd, obj.n_dd] = update_rounds(obj, obj.
              n_c_cd, obj.n_c_dd, obj.n_cd, obj.n_dd, K2, op, turn);
            p_c_cd=obj.n_c_cd(op)/(turn-1);
            p_c_dd=obj.n_c_dd(op)/(turn-1);
            p_cd=obj.n_cd(op)/(turn-1);
            p_dd=obj.n_dd(op)/(turn-1);

            if (p_cd == 0)
                p1=0.5;
            else
                p1=p_c_cd/p_cd;
            end
            if (p_dd == 0)
                p2 = 0.5;
            else
                p2=p_c_dd/p_dd;
            end

            E1 = p1*3 + (1-p1) * 0;
            E2 = p2*5 + (1-p2) * 1;

            if(E2>E1)
                decision = 2;
            else
                decision = 1;
            end

        end
    end

    function [n_c_cd_new, n_c_dd_new, n_cd_new, n_dd_new] = update_rounds(obj,
      n_c_cd_old, n_c_dd_old, n_cd_old,n_dd_old, K, op, turn)
```

```
            n_c_cd_new = n_c_cd_old;
            n_c_dd_new = n_c_dd_old;
            n_cd_new = n_cd_old;
            n_dd_new = n_dd_old;
            if (K(op,obj.playernumber,turn-1) == 1)
                if(K(obj.playernumber,op,turn-2) == 1)
                    n_c_cd_new(op) = n_c_cd_old(op) + 1;
                else
                    n_c_dd_new(op) = n_c_dd_old(op) + 1;
                end
            end
            if (K(obj.playernumber,op,turn-2) == 1)
                n_cd_new(op) = n_cd_old(op) + 1;
            else
                n_dd_new(op) = n_dd_old(op) + 1;
            end
        end
end
end
```

B.21 player17.m

Listing 21: player17.m

```
classdef player17 < handle
properties
    name='LookBack_DDowning';
    short='LDDO';
    n_c_cd=0;              % number of cases with oponent: c, downing: c
    n_c_dd=0;              % number of cases with oponent: c, downing: d
    n_cd=0;                % number of cases with downing: c
    n_dd=0;                % number of cases with downing: d
    playernumber = 17;

end
methods
    function P17 = player17(np)
        P17.n_c_cd=zeros(np,1);
        P17.n_c_dd=zeros(np,1);
        P17.n_cd=zeros(np,1);
        P17.n_dd=zeros(np,1);
    end

    function decision=decide(obj,K2,op,turn)
        if (turn == 1 || turn == 2)
```

```
            decision = 2;
        else
            [obj.n_c_cd, obj.n_c_dd, obj.n_cd, obj.n_dd] = update_rounds(obj, obj.
              n_c_cd, obj.n_c_dd, obj.n_cd, obj.n_dd, K2, op, turn);
            p_c_cd=obj.n_c_cd(op)/(turn-1);
            p_c_dd=obj.n_c_dd(op)/(turn-1);
            p_cd=obj.n_cd(op)/(turn-1);
            p_dd=obj.n_dd(op)/(turn-1);

            if (p_cd == 0)
                p1=0.5;
            else
                p1=p_c_cd/p_cd;
            end
            if (p_dd == 0)
                p2 = 0.5;
            else
                p2=p_c_dd/p_dd;
            end

            E1 = p1*3 + (1-p1) * 0;
            E2 = p2*5 + (1-p2) * 1;

            if(E2>E1)
                decision = 2;
            else
                decision = 1;
            end

        end
    end

    function [n_c_cd_new, n_c_dd_new, n_cd_new, n_dd_new] = update_rounds(obj,
      n_c_cd_old, n_c_dd_old, n_cd_old,n_dd_old, K, op, turn)
        n_c_cd_new = n_c_cd_old;
        n_c_dd_new = n_c_dd_old;
        n_cd_new = n_cd_old;
        n_dd_new = n_dd_old;
        if (K(op,obj.playernumber,turn-1) == 1)
            if(K(obj.playernumber,op,turn-2) == 1)
                n_c_cd_new(op) = n_c_cd_old(op) + 1;
            else
                n_c_dd_new(op) = n_c_dd_old(op) + 1;
            end
        end
        if (K(obj.playernumber,op,turn-2) == 1)
```

```
            n_cd_new(op) = n_cd_old(op) + 1;
        else
            n_dd_new(op) = n_dd_old(op) + 1;
        end
    end
end
end
```

B.22 player18.m

Listing 22: player18.m

```
classdef player18 < handle
    %created by Samuel Andermatt
    %this is an attempt to create a simple learning player
properties
    name='Watcher';
    short='WAT';
    playernumber=18;
    memory=6; %decides how many turns the player looks back to determine the most
      succesful strategy
    strategy; %decides which players strategy is chosen
end
methods
    function P18 = player18(np)
        P18.strategy=zeros(np,1);
    end
    function decision=decide(obj,K,op,turn)
        if (turn==1)
            decision=1; %cooperate in turn 1
        elseif (turn<obj.memory+2)
            decision=K(op,obj.playernumber,turn-1); %TFT for the first turns
        elseif (mod(turn-2,obj.memory)~=0)
            decision=K(obj.strategy(op),op,turn-obj.memory-1); %take the most
              succesful strategy against your opponent
        else
            %determine which strategy is best against your opponent
            np=length(K(:,1)); %number of players
            performance=zeros(np,1);
            for i=1:np
                for j=1:obj.memory
                    p=win([K(i,op,turn-obj.memory-1) K(op,i,turn-obj.memory-1)]); %
                      the winings player i made vs this opponent
                    performance(i)=performance(i)+p(1);
                end
```

```
            end
            [maxPF,maxInd]=max(performance); %determine which player performed best vs this opponent
            obj.strategy(op)=maxInd;
            decision=K(obj.strategy(op),op,turn-obj.memory-1);
        end
    end
end
end
```

B.23 player19.m

Listing 23: player19.m

```
classdef player19 < handle
    %At this point I apologize if I missuse terms differently used in
    %evolutionary algorithms. I am not familiar with this field.
properties
    name='Evolutionary';
    short='EVO';
    playernumber=19;
    stratlen=10; %length of the strategy that has to be optimized
    subsegs=1; %decides in how many segments each strategy is split
    childnum=2; %number of mutated children
    mut=0.1; %mutation rate
    transition=1000; %once the transition turn is reached the mutability is changed, this is because initially larger changes in the strategy are needed
    mut2=0.075; %the mutability after the transition
    transition2=5000; %second transition into the most stable phase
    mut3=0.05;
    child=zeros(1,1,1); %this array stores the children strategies
    parent=zeros(1,1); %the parent strategy
    seglen=1;
end
methods
    function P19 = player19(np)
        P19.child=zeros(np,P19.childnum,P19.stratlen);
        P19.seglen=P19.stratlen/P19.subsegs; %decides how long a segment is
        P19.parent=zeros(np,P19.stratlen); %the parent strategy
    end
    function decision=decide(obj,K,op,turn)
        %this part creates the sequance to start of
        if (turn == 1)
            decision = 1; %cooperate in turn 1
        elseif(turn<obj.stratlen+1)
```

```
         decision = K(op,obj.playernumber,turn-1); %use TFT to generate the first sequence
         obj.parent(op,turn-1)=decision;
    else
        if (turn==obj.transition) %transition into the second regime
            obj.mut=obj.mut2;
        end
        if (turn==obj.transition2) %transition into the second regime
            obj.mut=obj.mut3;
        end
        if (turn==obj.stratlen+1) %the last parent entry has to be made in a seperate space
            obj.parent(op,turn-1)=K(obj.playernumber,op,turn-1);
        end
        %the next part creates the first mutations
        if (turn==obj.stratlen+1)
            for i=1:obj.stratlen
                for j=1:obj.childnum
                    if rand>obj.mut %ad a mutation
                        obj.child(op,j,i)=K(obj.playernumber,op,i);
                    else %add a mutation
                        if (K(obj.playernumber,op,i)==1)
                            obj.child(op,j,i)=2;
                        else
                            obj.child(op,j,i)=1;
                        end
                    end
                end
            end
        end
        %from now on the main algorithm can run
        if(mod(turn,obj.stratlen*(1+obj.childnum)+1)==1)
            %create new children
            %calculate performance
            perf=zeros(obj.childnum,obj.subsegs); %this array will store the performance of all strategies
            parperf=zeros(obj.subsegs,1);
            for i=0:obj.childnum
                for j=1:obj.subsegs
                    for k=0:obj.seglen
                        turn2=turn-(obj.childnum-i+1)*obj.stratlen+(j-1)*obj.seglen+k-1;
                        w=win([K(obj.playernumber,op,turn2) K(op,obj.playernumber,turn2)]); %calculates the winnings
                        if(i==0)
```

```
                            parperf(j)=parperf(j)+w(1); %updates the parents performance
                        else
                            perf(i,j)=perf(i,j)+w(1); %updates the childrens performance
                        end
                    end
                end
            end
            for i=1:obj.subsegs
                [maxperf,perfInd]=max(perf(:,i)); %calculates the performance of the best child, and which child performed strongest
                if (maxperf > parperf(i)) %child performes better
                    for j=1:obj.seglen
                        turn2=(i-1)*(obj.seglen)+j; %turn in the strategy that will be changed
                        obj.parent(op,turn2)=obj.child(op,perfInd,turn2); %exchange the segment of the parent with the more succesful segment
                    end
                else %parent is strongest
                end
            end
            %create and mutate children
            for i=1:obj.childnum
                obj.child(op,i,:)=obj.parent(op,:);
                %mutate
                for j=1:obj.stratlen
                    if (rand<obj.mut) %add mutation
                        if (obj.child(op,i,j)==1)
                            obj.child(op,i,j)=2;
                        else
                            obj.child(op,i,j)=1;
                        end
                    else
                    end
                end
            end
        end
        %choose the next move
        if (mod(turn,obj.stratlen*(1+obj.childnum)+1)==0)
            decision=K(op,obj.playernumber,turn-1); %add a TFT step until you evaluate the performance of each child
        else
            %perform the appropriate child strategy
```

```
                    x=mod(turn,obj.stratlen*(1+obj.childnum)+1); %decides in which turn
                       we are in the cicle
                    x2=floor((x-1)/obj.stratlen); %decides which strategy will be
                      played, 0 is the original strategy
                    if(x2==0) %the parent strategy is played
                        decision=obj.parent(op,x);
                    else
                        x3=mod(x-1,obj.stratlen)+1; %decides in which turn we are
                          during the current stategy
                        decision=obj.child(op,x2,x3);
                    end
                end
            end
        end
    end
end
```

B.24 player20.m

Listing 24: player20.m

```
classdef player20 < handle
    %created by Samuel Andermatt (the idea is rather straightforward, so in
    %case somebody has had this idea before I apologize)
    %The idea is that you basically go for TFT, but try to avoid to enter a
    %state where players reject each other over and over again.
    %Therefore you will try to reconcile, as soon as the rejections on both
    %sides caused enough damage to the other player to avoid beeing
    %exploitable
properties
    name='Limited Reconcilation TFT';
    short='LTFT';
    playernumber=20;
    k=zeros(1); %this is a number that allows the object to switch between a
      recooncilng state and a TFT state. 0 means TFt, 1 means reconcile.
    recnum=zeros(1); %how often reconciliation was attempted
    maxRec=3; %how often reconciliation is attempted, if cooperation appears, then
      the number is reseted
end
methods
    function P20 = player20(np)
        P20.k=zeros(np,1);
        P20.recnum=zeros(np,1);
    end
    function decision=decide(obj,K,op,turn)
```

```
    if (turn == 1)
        decision = 1; %cooperate in turn 1
    elseif (obj.k(op)==1) %player is in reconciling state
        obj.k(op)=0; %Go back to TFT state
        decision = 1;
    elseif (K(op,obj.playernumber,turn-1) == 1)  %cooperation is always met
      with cooperation
        decision = 1;
        if (turn>2) %this test cannot be made after the first step, because it
          goes two steps back
            if (K(op,obj.playernumber,turn-1) == 1 && K(op,obj.playernumber,
              turn-2) == 1) %a peaceful state is reached, the reconciliation
              number is reseted
                obj.recnum(op)=0;
            end
        end
    else
        %calculate if the conditions are met to enter reconciling state
        memory=20; %determines how many turns you go back
        winCoop=0; %the winnings the opponent had if he cooperated
        winReject=0; %the winnings he made by rejecting
        %calculate the winnings B can get by exploiting the
        %reconcilation attempt, a reconcilation attempt is two
        %consecutive cooperative steps
        C=win([2 1]);
        RecT=2*C(1);
        for i=turn-1:-1:turn-memory-1
            if(i<1)
                continue; %there are no turns before the first turn
            end
            A=win([1 1]); %winnings for cooperation
            winCoop=winCoop+A(1); %the points he would have won by cooperating
            B=win([K(op,obj.playernumber,i) K(obj.playernumber,op,i)]); %the
              points won by rejecting
            winReject=winReject+B(1); %the points the opponent actually won
              trhough rejection
            if (winCoop>winReject+RecT&&obj.recnum(op)<3) %Cooperation would
              have bben better for the opponent
                obj.k(op)=1;
                obj.recnum(op)=obj.recnum(op)+1;
                decision=1;
                break;
            end
        end
        if (obj.k(op)==0) %Criteria for reconcilation have not been met
            decision = 2;
```

```
                  end
              end
          end
end
end
```