UNIVERSITÀ DEGLI STUDI DI TORINO
FACOLTÀ DI ECONOMIA

MASTER'S DEGREE IN ECONOMICS (LM56)

THESIS IN SIMULATION MODELS FOR ECONOMICS

Agent-Based vs. Robotic Simulation: a Repeated Prisoner's Dilemma Experiment

Advisor: *Prof. Pietro Terna*

Opponent: *Prof. Sergio Margarita*

Candidate: *Bruno Grimaldi*

Academic Year

2011/2012

Abstract

At present, simulation models are a very important tool for researchers and decision-makers. This project aimed at analyzing and comparing two different kinds of methodology, namely a virtual and a robotic agent-based simulation. Can the real-world uncertainty and imprecision help the cause of modelers? What is the added value of engaging in a robotic simulation experiment? The analysis of the two methodologies has been carried out through a side-by-side creation and confrontation of a virtual and a robotic simulation model center around the Prisoner's Dilemma. The real-world model consists of two robots (agents) that meet each other repeatedly facing the decision to either cooperate or defect. The environment has been structured in such a way that their choices entail consequences in terms of distance to travel. The same model has been created in a virtual environment with NetLogo. The confrontation of the two simulations, showed that the physical simulation was able to give us many insights on aspects otherwise neglected. It allowed us to better calibrate the virtual model to the real environment by highlighting errors of analogy. Furthermore, the imprecision of real world led us to some key insight about the Prisoner's Dilemma as well. Moreover, the educational aspect of engaging in a physical simulation should not be underestimated. Programming robotic agents allows to obtain real feedbacks, without layers of abstraction to obfuscate the dynamics of the interactions between the agents and the "world". We can state that the robotic simulation provides a valuable contribution to the virtual simulation under several aspects and in our case offers another important perspective on the Prisoner's Dilemma.

Contents

1

Introduction

1.1 General introduction

The idea behind my thesis project comes from three factors that started to converge during my second year of Master's Degree. The first one was the opportunity to learn about simulation for economics, agent-base modeling, complexity and emergent behavior in the course Simulation Models for Economics. In addition, during this course I had the opportunity to get familiar with more than one programming language. The one we studied in depth was NetLogo, which is a programming language particularly suitable for simulation in the social sciences and the exploration of emergent phenomena.

The second factor that contributed to the idea behind my thesis project was the discovery of Lego Mindstorms NXT 2.0, a programmable robotics kit released by Lego in July 2006.

I discovered Lego NXT in summer 2011, right after finishing my course in Simulation Models for Economics and I immediately developed an interest in it. I have always been a Lego fan and I decided to buy one because I thought it would have been a perfect tool to enhance my programming skills, while having a real life feedback of what I was doing. In fact, the modularity of Lego kit allows to start from very simple robots that can be used in a turtle-like manner, to very complex one that can solve very hard tasks, for instance the Rubik Cube. Additionally, on the Internet, there is a vast and active community of developers and Lego NXT fans that share new programs, new robots and precious advices every day.

Another feature of Lego NXT that influenced my decision to purchase one, was the possibility of programming the NXT Intelligent Brick (this is how the computer is called) with a large variety of unofficial languages such as NXC, NBC, leJOS NXJ, RobotC, Robolab, NXT-Phyton and not just the official one provided by Lego.

The third factor that contributed to the idea behind my thesis project was the reading of the paper *Behavioral AI Experiments and Economics* presented by Birk and Wiernik (1996) during the 21th European Conference on Artificial Intelligence. The paper is about the promising possibilities of cooperation between behavioral AI and economics and describes an experiment in which robotic agents live in an ecosystem together with competitors. The authors basically created an agent-based simulation whose environment was the real world and I found this idea very fascinating.

These three elements really made me interested in exploring how autonomous robotic agents could help the current development of simulation models for economics. *Can the real-world uncertainty and*

imprecision help the cause of modelers? What is the added value of engaging in a robotic simulation experiment?

The desire to answer these questions pushed me to begin my project towards this direction. To achieve my goal, I needed to create a simulation model feasible in NetLogo and above all in the real world. I found in the Lego NXT the perfect tool to create inexpensive robotic agents that do not need too much computational power. As we know, agent-based simulations do not require agents to follow complex procedures. On the contrary, they usually follow very simple rules.

The economic field I wanted to explore with my simulation experiments is game theory, more precisely the case of the Prisoner's Dilemma game. The Prisoner's Dilemma is a simple example of what happens in very common situations in which there is a confrontation between two forces that can either decide to cooperate or to fight, for instance in price wars, advertising campaigns or in arms race.

The simulation model consists of two players that meet each other repeatedly through Prisoner's Dilemma like interactions. Each player is endowed with five strategies (Cooperate, Defect, Random, Tit For Tat, Win Stay Lose Shift) whose intensity of utilization can be set through sliders before running the model, allowing the decision-maker to create many initial scenarios similar to real problem that he/she is facing. In order to do so, I created an environment in which the two robots face the decision to either cooperate or defect and this entails consequences in terms of distance to travel. Then, I replicated the same model in NetLogo for a comparison to seek differences, gains or drawbacks of the two methodologies.

Listed below are the chapters of my dissertation with the subdivision of the arguments. Here I will provide only a brief general description.

Chapter 1 is a review concerning some of the most relevant concepts of game theory. The terminology of games is introduced, as well as the typologies of games, also with some examples. Furthermore, the original formulation of the Prisoner's Dilemma is presented and explained in detail. Finally, the game is put in context with the economic theory. This review is critical to quickly understand the analysis undertaken in the following chapters.

Chapter 2 begins with a discussion on models and their importance in decision-making processes. Then a detailed explanation of agent-based models (ABMs) has been undertaken and an investigation on how simple rule can lead to emergent phenomena has been carried out. A brief section has been dedicated to how ABMs can represent the real world. The description of the Environment-Rules-Agents (ERA) scheme, a general scheme that can be employed in building agent-based simulations, concludes the chapter.

Chapter 3 is devoted to the introduction and explanation of a sociological model, called EGO, developed by the sociologist Luciano Gallino. Both the psycho-sociological and the AI version of the model are discussed. Finally, the theory of social actor is put in context with my thesis project.

Chapter 4 deals with robotics and AI used for economic purposes. First, I discussed about the reason that brought my attention to these particular fields. Second, two articles are illustrated and commentated: the first is *Behavioral AI Experiments and Economics* by Birk and Wiernik (1996); and the second is *Robots that Have Emotions* Parisi and Petrosino (2010). Both articles treat topics very relevant to my thesis, namely robotic simulation oriented to economic areas. Furthermore, a discussion about the pedagogical aspects of the *bifocal* (virtual + physical simulation) approach is undertaken.

Chapter 5 is the main part of my work. Here, the model that I have created for my thesis is explained in detail. The structure of the model is carefully presented for both the virtual and the robotic simulation. The central part of the chapter is devoted to an easy example of ABMs that represents a Repeated Prisoner's Dilemma model. The purpose of this section is to understand the functioning of NetLogo in the first place, and see how it can be used for agent-based simulations. After introducing the operation of NetLogo, I provided a detailed description of the code of the program I have developed for my thesis.

Chapter 6 is devoted to the simulations. In this section, I have undertaken a plan of simulations to discover what is the winning mix of strategies that player A can use when confronting player B in repeated Prisoner's Dilemma interactions. Then, I introduced the hardware and the software of the robotic simulation and I have undertaken a real simulation through the mean of two Lego NXT robots. After, assessing the results obtained with the two methodologies, a confrontation between the virtual and the real experiment is carried out.

2

Game Theory and the Prisoner's Dilemma

2.1 What is game theory?

Game theory is a branch of mathematics that studies strategic decision making. Thanks to its characteristics, it can be applied to all kind of situation in which there is a confrontation between two contestants and in which the two contestants must make decisions most favorable to their interest, without knowing what decisions the other contestant will make. Therefore, it can be said that game theory is an interactive game: in this situations, each player constantly tries to anticipate the most

probable moves of the opponent in order to develop the best counter-move possible. The aim of game theory is to develop a conceptual framework to allow players to correctly define moves and counter-moves (Deulofeu, 2011).

2.2 Principles of game theory

The reader must understand that even though game theory uses terminology of games (players, games, strategies, balanced game, the value of a play, etc..), every situation that I am going to present has nothing to do with an actual game. We can imagine it as confrontation, initially between two persons or two groups, in which there are some rules that determine the possible moves. Players make decisions simultaneously, which does not allow them to know the decision of the opponent, and both players obtain a payoff. From now on I will use the terms (Pindyck & Rubinfeld, 2006):

- *players* to refer to the participants of the situation;
- *game* to refer to a situation in which players make strategic decisions that take into account each other's actions and responses;
- *strategies* to refer to the rule or plan of action for playing a game:
    - *Pure strategy* to refer to a strategy in which a player makes a specific choice or takes a specific action;
    - *Mixed strategy* to refer to a strategy in which a player makes random choice among two or more possible actions, based on a set of chosen probabilities;
    - *Dominant strategy* to refer to a strategy that is optimal no matter what an opponent does;

- *payoffs* to refer to the gains or the losses associated with a possible outcome;

2.3 Type of games

In game theory we can have zero-sum and non-zero-sum games. The difference between the two is that in the first type the wins of one player always correspond to losses of the other players (this is why they are called zero-sum: one wins exactly the amount one's opponents lose), whereas the second type even if the aim of the game is still to win, the confrontation is not total. On the one hand, the wins of one players do not necessarily imply losses for the other player. If one player adopting certain strategies can win, so can the other. On the other hand, there are situations in which cooperation can bring benefits to both players. This necessarily implies the introduction of elements such as communication, trust, but also threats (for instance to force the other players to keep a commitment). In this case we can talk about non-zero-sum games and we can distinguish between cooperative and non-cooperative games. Cooperative games are games in which participants can negotiate binding contracts that allow them to plan joint strategies, whereas non-cooperative games are games in which negotiation and enforcement of binding contract are not possible. An example of the former could be a buyer and a seller that negotiate the price of a good and an example of the latter could be given by two competing firms, that assuming the other's behavior, independently determine pricing and advertising strategy to gain market share. Usually the non-cooperative games are more interesting to analyze. The most important aspect in strategy designing is to understand your opponent's point of view, and (assuming your opponent is rational) deducing how he or she is likely to respond to your actions (Pindyck & Rubinfeld, 2006). We

have to keep in mind that game theory is focused around decision making and from now on this aspect will become more important since that by focusing on the Prisoner's Dilemma we will create a tension between cooperation or defection. What can be, under these circumstances, the decisions of each player? What emerges is a so called *dilemma*: both players can cooperate or defect and it is not clear which decision will bring greater benefits, since the outcome also depends on the decisions of the other player. In general, cooperation will benefit both players and it is the best outcome, whereas defection will be devastating. If we had just these two outcomes the dilemma would not exist; but if one of the players wants to cooperate and the other one want to defect, the benefits would be for the defector and they would be greater from those obtain with cooperation: then the dilemma becomes evident.

The complexity of these games regards the fact that there is a mixture of mathematical aspect and physiological and moral aspects. The solution of these games will not only be hard because of the mathematical aspect but also because the outcomes depend on the decision of the opponents. Certainly, the interest surrounding non-cooperative games is much higher than cooperative game, since they more often occur in the real world, where rarely situation of conflict don't mix cooperation and confrontation.

We can consider that the set of situations for two persons, analyzed and solved in the game theory, can be ordered in two opposite groups: in one there are zero-sum games, fully competitive and in the other those fully cooperative. Both of these types are, at least in theory, easy to solve (usually we have situations in which both players have the same goal and the way to achieve that goal is to combine efforts, for instance the pilot of an airplane and the airport controller).

In this thesis, we are going to analyze the opposite case. The two players have opposite and shared interests at the same time. For instance, we can think of a seller of an apartment and a possible buyer: both want to close the negotiation (cooperation), but at the same time they disagree on price (conflict/defection); the same situation can be found in the negotiation for the merge of two firms.

To demonstrate the difference between zero-sum and non-zero-sum games we propose an example taken from *The Prisoner's Dilemma and Dominant Strategies* by Deulofeu. Imagine a situation in which two competing firms (A and B) want to promote their products and both of them receive the same offer from the TV channel: they can make the advertisement in the afternoon (40% of the audience watch TV at this time slot) or at night (with 60% of TV viewer connected); there is no overlapping of TV viewer between the two time slots. If both firms choose the same time slot, each firm will be able to sell its products at 30% of the audience of the same time slots, whereas if they choose to broadcast the TV commercial at different times, each of the firm will catch 50% of its own time slot. What is the best strategy for each firm? Is it best for the two firms to communicate or to hide the decision?

We can express the game in form of payoff matrix in which each value stands for the percentage that each firm will obtain. We thus have to put two values in each cell of the payoff matrix since the gain of one firm does not necessarily correspond to a loss of the other one, but each firm has its own gains. In Table 1.1, the first value in the each cell represents the benefits for firm A and the second value the benefits for firm B.

		Firm B	
		Afternoon Commercial	Night Commercial
Firm A	Afternoon Commercial	12, 12	20, 30
	Night Commercial	30, 20	18, 18

Table 2.1 –Payoff matrix of firm A and B

If A and B decide to broadcast the commercial in the afternoon, each firm will obtain 12% of the total audience (30% of 40%), whereas if they chose to broadcast the TV commercial at different times, their results will be symmetric: if A advertises in the afternoon and B at night, A will obtain the 20% (half of 40%) and B the 30% (half of 60%) of the total TV viewers. If they invert their strategies they will get opposite results. Given the symmetry of the matrices and keeping in mind that the strategies of A correspond to rows and the strategies of B correspond to columns, the analysis of the two is similar. Treating this situation as a zero-sum game we would search for a mixed strategy that give a value of the game for player A. The strategy would be as follows: broadcast the TV commercial in the afternoon with probability 3/5 and broadcast the TV commercial at night with probability 2/5. Firm A thus obtains an average gain for game of 19.2. In the same way, given the symmetry, player B would do something similar: on 5 games, firm B will broadcast randomly, 2 times the commercial in the afternoon and 3 times the commercial at night. Firm B obtains the same average gain for the game as firm A. Only analyzing the results in details, we can see that both players can strive to gain more without affecting the other player's

gains. The previous result, obtained with a mixed strategy, is thus not optimal. The above matrix is structured in way so that the reasoning of zero-sum game does not work (reduce as much as possible the gains of the opponent which is equivalent to increase as much as possible my gains). Suppose that firm A, instead of playing a mixed strategy, plays always the strategy of broadcasting the advertising at night, whereas firm B plays the optimal mixed strategy. Firm A would gain, in average, 30*2/5+18*3/5=22.8, while firm B would continue to gain 19.2. Firm B keeps gaining the same amount, A instead increased its gains, which is something that could not happen in a zero-sum game. Clearly, firm B desires play the same strategy as firm A, hoping that A will play a mixed strategy. What happen if both firms play the pure strategy of broadcasting the TV commercial at night? The two firms will gain 18% and they would reduce their gains by the same amount. It looks like a dead end, since each firm can increase its gains without affecting the other one, but if they both want to gain more, the situation becomes even worse. Clearly, we can have another solution. Imagine that the two players can communicate and they decide not to choose the same time slot for the commercial. They could agree to play alternately the two strategies so that the average payoff would be 25% (A alternates gains of 20 and 30 and B alternates gains of 30 and 20). This could be the best solution and it is also a situation of equilibrium (Deulofeu, 2011).

The above example showed that when we play a non-zero-sum game, sometime it is possible to use strategies of cooperation to improve the results. The problem arises when the extra gains are not fairly distributed among players. In other words, the problem consists in the repartition of the surplus. Also, the idea of “fair” distribution is not always evident, and sometimes it can exist more than one rational solution. In the rationality there can be more than one rational solution.

2.4 Prisoner's Dilemma

The game know as Prisoner's Dilemma (a name assigned to a specific type of non-zero-sum games proposed by Albert. W. Tucker in 1950) is one of the most famous of the game theory. It is a simple example of what happens in very common situations in which there is a confrontation between two forces that can either decide to cooperate or to fight, for instance in price wars, advertising campaigns or in an arms race.

Even if the name of the dilemma refers to a prisoner and it can be formulated as a game between two criminals that do not know whether to declare them innocent or guilty, we can expose it in term of more interesting situations. For instance a war between two countries can be formulated as follow.

Two enemies countries, C1 and C2, have to decide their policy on arms. Each country has two possible strategies:

1. deny any form of cooperation, that is to arm as to prepare for a possible war;
2. cooperate, that is to disarm or at least to prohibit certain weapons.

The four possible results are (A, A), (A, B), (B, A) and (B, B), in which the first coordinate is the strategy of C1 and the second is the strategy of C2.

The four outcomes can be represented by the following table:

		Country C2	
		Option A	Option B
Country C1	Option A	A, A (arms race)	A, B (only C2 gets weapons)
	Option B	B, A (only C1 gets weapons)	B, B (disarm)

Table 2.2 – Payoff matrix of arms race

We could assign a payoff to every outcome in the matrix, keeping in mind that in this case the payments are going to be different for each player. In each cell we will have a pair of numbers, the first one corresponds to the gains for C1 and the second one to the gains of C2. Doing so we can build a payoff matrix as follows:

		Country C2	
		Option A	Option B
Country C1	Option A	2, 2	5, 0
	Option B	0, 5	4, 4

Table 2.3 – Payoff matrix of arms race with numbers

If we interpret the numbers as gains, the dilemma becomes clear. Every country wants to get weapons. As a matter of facts, if C2 chooses option A, C1 will gain 2 if it also gets weapons, otherwise it gets 0. If C2 chooses

B, C1 would gain 5 if it gets weapons and 4 if it does not. Symmetrically the same happens for C2, so an arms race is the solution that entails greater benefits. We can say that (A, A) is the equilibrium solution of this non-cooperative game.

Clearly, each country wishes that the other country disarms (because it generates higher benefits) and, furthermore, the highest global benefit is reached when both countries disarm. If the two nations do not cooperate, the highest global benefit (4, 4) cannot be realized, but if just one them cooperates, it will bear a huge risk (since it does not know what the other will do). Consequently, trust becomes an essential element of the game. Without trust the best global outcome is totally unstable because every player will try to protect himself from a possible defection of the opponent (Deulofeu, 2011).

Even if the formulation of the Prisoner's Dilemma refers to game theory, the issue that stands behind it was already considered earlier in time. Thomas Hobbes, English philosopher and author of the book *Leviathan,* analyzed a situation very similar to the one we can find in the Prisoner's Dilemma. He claimed that before government existed, the society used to live in a state of anarchy in which only the most competitive individual would find space. The only way to enhance cooperation was the creation of laws and their enforcement. Hobbes argued that there is a need for a central authority (the government), since leaving the society in the hands of individual would lead to a catastrophe (Holcombe, 2004).

After this brief introduction and some general concept of game theory, we can now move to the actual Prisoner's Dilemma game.

Nowack described the Prisoner's Dilemma in his book *SuperCooperators* as follow.

Imagine that you and your accomplice are both held prisoner, having been captured by the police and charged with a serious crime. The prosecutor interrogates you separately and offers each of you a deal. This offer lies at the heart of the Dilemma and goes as follows: if one of you, the defector, incriminates the other, while the partner remains silent, then the defector will be convicted of a lesser crime and his sentence cut to one year for providing enough information to put in in prison his partner. Meanwhile, his silent confederate will be convicted of a more serious crime and burdened with a four-year sentence.

If you both remain silent, and thus cooperate with each other, there will be insufficient evidence to convict either of you of the more serious crime, and you will receive a sentence of two years for a lesser offense. If, on the other hand, you both defect by incriminating each other, you will both be convicted of the more serious crime, but given reduced sentences of three years for at least being willing to provide information.

Literature provides us with endless variants of the dilemma. We can have different circumstances, different punishment and temptations and details of imprisonment. Whatever the formulation, there is a simple central idea that can be represented by a table of options, known as payoff matrix. This can sum up all four possible outcomes of the game, written down as two entries on each of the two lines of the matrix.

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

Table 2.4 – Payoff matrix of Prisoner's Dilemma

Let's see how the payoff matrix (Table 2.4) works beginning with the first row:

1. both cooperate (this means a sentence of 2 years, here written as -2 to underline the years of normal life you lose);
2. you cooperate and your partner defects (-4 years for you and -1 for him);

On the second row of the matrix we have:

3. you defect and your partners cooperates (the opposite situation of before, in this case you will have -1 years and you partner -4 years);
4. both defect (-3 years each)

From a selfish point of view, the best outcome for you is (3), then (1), then (4) and finally (2), whereas for your confederate the order would be (2), (1), (4) and (3). If you see yourself as rational and selfish then you will think as follow:

Your partner will either cooperate or defect. If he cooperates, you should defect, because is my preferred outcome. If he defects, you should defect in order to avoid the worst possible outcome for you. In conclusion, it does not matter what your partner does, defecting is always your best choice.

Defecting is called a dominant strategy in a game with this payoff matrix, that is, the strategy is always the best to adopt, regardless what strategy is used by the other player. As a matter of facts, defecting is always your best strategy: if you both cooperate, you get two years in prison but you only get one year in prison if you defect, hence you defect. If the other person defects and you cooperate, the you get four years of jail, but you only get three years if you both defect, hence you defect.

The problem with this chain of reasoning is that if your partner think in the same way then he would reach the same conclusion. As a consequence you both defect and you both spend three years in prison. The Dilemma arise because if you both follow the best, most rational dominant strategy it leaves you worse off than if you had both remained silent (you both end up with the third best outcome, instead of the second best by cooperating). This is the essence of the Prisoner's Dilemma: the deviation from personal and social interest. If you had trusted each other, by cooperating, you would both be better off than if you had both acted selfishly. This can help us to understand the meaning of cooperation not only in a game theory context but also from an economic standpoint: one individual pays a cost so that another receives a benefit (Nowack & Highfield, 2011).

In order to create the dilemma certain relations between the scores obtained in the various situations have to be in place. The highest win is possible is denoted by T, the prize of the temptation to cheat when the other has decided to cooperate. The worst result is S (Sucker's pay) and goes to those who decided to cooperate when the other player has decided to defect. R, the score of mutual cooperation must be greater than P, the penalty for the mutual defection. We create the Dilemma if T is greater than R which is greater than P which is greater than S.

In order to correctly define the dilemma we still need to establish that the players cannot escape the dilemma exploiting alternately each other. The score obtainable with an alternation of T and S must be less than that achieved with mutual cooperation, thus T + S <2 * R. Overall the two conditions that define the Prisoner's Dilemma are:

1. T > R > P > S
2. T + S <2 * R

If the "Prisoner's Dilemma" is played only once (single shot game) we would expect the emergence of dominant strategies of the game, that is, defection for both players. Even if the two players have a known number set of meetings, we would create a similar situation. As a matter of facts, there would be no incentive to cooperate in the last move, but also the penultimate, since both would be expected that there will be a defection by opponent at last. We could adopt a "backwards" reasoning and make mutual cooperation impossible.
The argument no longer holds if players meet repeatedly an indefinite number of times. Under normal conditions, it is virtually impossible to determine a priori how many meetings there will be. The possibility of cooperation is offered if the game is repeated for an indefinite number of times.
Listed below are the other assumption of a repeated Prisoner's Dilemma:

- The interaction is between two players at a time. A player interacts with all others, but always with one player at a time;
- Each player is able to remember the history of its previous meetings;

- There is no mechanism available to players that allows threats or commitments achievable with certainty;
- There is no way to be sure of what the other player will do;
- Individuals cannot base their decisions on the results of previous interaction of the opponent with a third player, the only information available regard the history of the previous interaction between the two players involved at the moment;
- There is no way to escape the interaction;
- There is no way to change the results to be obtained;

What makes possible the birth of cooperation is, in this model, only the possibility that the two agents can meet in a second chance. This possibility means that the choice made today only determines the outcome of the present but affect future choices of both contenders, the future rests on the present and affect the current strategy.

3

An introduction to Agent Based Models (ABMs)

3.1 Why do we use models?

> "Remember that all models are wrong: the practical question is how wrong do they have to be not to be useful" (Box, 1987)

I wanted to begin with the above quotation because I believe it captures the very essence of what modeling means. Nowadays models are used in every kind of discipline: mathematics, biology, chemistry, physics, etc. and also in social sciences and even in the study of disciplines such

as languages. Researchers can benefit a lot from the use of formal models as we will see in the next paragraphs, but the reason why I wanted to quote Box (1987) because in using models one should pay a lot of attention. As a matter of fact, sometimes we should not base our decisions entirely on models but take them as guidance to our way of thinking. So why is it important to use models? Page (2008) discusses four main reasons why it is important to understand and use models. The reasons are the following:

1. To be an intelligent citizen of the world
2. To be a clearer thinker
3. To understand and use data
4. To better decide, strategize, and design

3.1.1 How can the use of models make us more intelligent citizen of the world?

Models are in a sense abstraction and simplification of the real world, nonetheless they are useful because they can assist us in organizing and doing things in a better way and communicate better with other, in fact, models can be thought as the new lingua franca not only in the academic world but also in business and politics. Perhaps the most important feature that stands behind models is that they allow us to "tie ourselves to a mast". This sentence is taken from the Odyssey and it refers to the passage in which Ulysses' ship passes by the sirens. He wants to hear their singing (the sing of the sirens is beautiful and fatal at the same time) so he asks to his crew to tie him to the mast of the ship and he also have them putting wax in their ears. Using this stratagem he pre-commit not to steer his ship over the sirens but at the same time he is able to hear the beautiful singing of the sirens. We can

think about the use of the models the same way as tying ourselves to a mast, the mast of logic, meaning that they can help us to figure out which ideas are valuable since they give us the conditions and rigor under which logical reasoning can be undertaken. Another important feature of models is that they are fertile. In this context, fertility means that once you learn a model you can apply it across many different settings. This is particularly evident when we talk about the prisoner's dilemma model. As a matter of fact, it can be easily applied to businesses, politics, international relations and even to everyday situations such as standing in line at the supermarket and deciding whether to let a person with few item in hands to cut the line in front of you.

Furthermore, models make us humble, in the sense that after we construct a model we can get very different predictions of what we thought before. They make us humble because they make us see the full dimensionality of a problem. So, if we want to really make sense of the world we want to have a lots of formal models in our disclosures. (Page, 2012).

3.1.2 Models make us clearer thinkers

The second reasons that Page (2008) states, is that models make us clearer thinker. The first step we have to consider in writing a model is to name the parts, meaning that the formalization process make us think what our objectives are and which are the relevant features that we want in include in it. We also have to figure out how much we have to leave out in order to understand what is happening.

The following step consists in identifying the relationships among the parts and this will allow us to undertake an inductive exploration. Then we can try to understand the class of outcome. Is the outcome of the

model, random, cyclical, complex or an equilibrium? Models allow us to identify logical boundaries and also allows us to communicate in a better way our line of reasoning. It is not that difficult to see how models can make clearer thinker: they make us name the parts, identify relationships, understand the class of outcomes, identify logical boundaries and make us communicate better our ideas. We can be sure that any conclusion reached through the use of a formal model is at least logically consistent.

3.1.3 Models make us understand and use data

Another important reason why we create models is to use them with data. Models can allow us to understand patterns, predict points, and also to predict bounds. A model will not always tell us exactly what is going to happen, especially if related to the social sciences, because there is too much complexity and too much uncertainty but it can at least give us bounds about what is going to happen. An often overlooked characteristic of models is that they allow to retrodict and help us to understand the past. Perhaps one of the most famous example in the agent-based model scenario is the Artificial Anasazi model. Janssen (2009), constructing an archaeological model, showed how the collapse of the society of Anasazi, in Long House Valley in Arizona, evolved between 800 and 1350. Taking into account the population dynamics, the social-ecological systems and making a precise calibration with the historical data, Janssen was able to simulate in a very successful way the dynamics that brought this complex society to abandon the Long House Valley.

This bring us to the next point that is that models allow us to make an informed data collection, meaning that once you know which variables are important and have to be included in the model, you can go out and

focus your attention and research on those specific data. The world is full of data, models allow us to give a structure to those data and transform them into useful information and then turn that information into knowledge. Without models, data is just a bunch of number that are not valuable to anybody.

3.1.4 Models help us to better decide, strategize and design

Models are important tools for decision makers. When you have to make a decision, while running a business or simply choosing which apartment to buy, it's helpful to build or structure information in a way to make a better and more informed choice. The same applies when we want to strategize. That is the reason why we study game theory models and decision trees: they allow individuals to strategize better. Models improve our ability of making decisions and taking actions through the use of comparative statics and the possibility to rerun the model several times (Page, 2012).

3.2 Models in economics

In the social science context, precisely economics, models are employed in order to make prediction about economics policies. Models are used because of problems of conducting experiments on economic systems and because the system is often too large and complex to analyze in its entirely. If the model is too highly specified, it may not be capable of capturing important forms of response. On the other hand, if it is too general, it may not able to provide any clear prediction (Hindriks & Myles, 2006).

In assessing a policy we can distinguish between *positive* and *normative* analysis. Positive economics deals with *what is*, not with *what ought to be* and its task is "to provide a system of generalizations that can be used to make correct predictions about the consequences of any change in circumstances" (Friedman, 1953). Normative economics, instead, investigates what the best policies are and therefore it cannot be independent of positive economics. As we have started to describe in the previous paragraphs, models are not just about prediction and forecasting. As a matter of facts, Epstein (2008) gives us sixteen reasons why models can be useful:

1. Explain (very distinct from predict)
2. Guide data collection
3. Illuminate core dynamics
4. Suggest dynamical analogies
5. Discover new questions
6. Promote a scientific habit of mind
7. Bound (bracket) outcomes to plausible ranges
8. Illuminate core uncertainties
9. Offer crisis options in near-real time
10. Demonstrate tradeoffs - suggest efficiencies
11. Challenge the robustness of prevailing theory through perturbations
12. Expose prevailing wisdom as incompatible with available data
13. Train practitioners
14. Discipline the policy dialogue
15. Educate the general public
16. Reveal the apparently simple (complex) to be complex (simple)

I am not going through each point of the above, but I will just discuss the ones that I believe to be the most relevant to my purpose. In developing this research we will see how models can be useful to educate the general public. This is one of the main objective why I decided to make a simulation in the real world using the Lego NXT. Real robots will not only allow us to capture the imperfection and variability typical of the real world, but they will also assist us in explaining an agent-based simulation to non-specialists. The model reproduced through autonomous robotic agents can be a perfect complement for explaining a sophisticated and ambiguous interaction as the one of the prisoner's dilemma game to person that are not in the field.

3.3 What are ABMs?

We can formally define agent-based modeling as "a computational methods that enables a researcher to create, analyze, and experiment with models composed of agents that interact within an environment" (Gilbert, 2008). Agent-based simulation is a very useful tool for studying complex systems. With this methodology is possible to create a virtual world based on autonomous agents which can interact with each other and therefore enable us to see emerging phenomena. A system is said to be complex if it is not possible to analyze it by breaking it down into its constituent parts. Terna (2011) defines complexity as the characteristic of a system in which the action of the agents – who operates and interact individually, sometimes following very simple rule – produce aggregate outcomes that are not predictable from the apparent behavior of the individuals.

Why should we use agent-base modeling instead of the regular models? We use ABMs when we don't have the possibility to study a linear system, defined as a system in which the aggregate behavior is the sum

of the individual behaviors. On the contrary, a non-linear system is not simply referable to the parts that compose it. ABMs are thus particularly useful in studying situations in which it is not possible to deal with the equations that describe the system (even if they can be found ex-post). A very simple example can be found in Terna (2011): "A set of agents have to order a vector of data, so that they can operate an exchange, and they should use heterogenic ordering criteria, without necessarily express transitive coherences, meaning that if A is preferred to B and B is preferred to C, it can also be the case that A is preferred to C". It is practically impossible to determine ex-ante the equations of the problem. Non-linear complex system behaviors depend on the interaction between the parts and not just on the characteristics of the parts. Non-linear systems are pervasive in the social world (Gilbert & Terna, 2000). The availability of computers and informatics helped us to study complex systems not from top-down by decomposing them, but from bottom-up by composing them.

Agent-based simulation is therefore a type of model and the computer becomes our artificial lab, connected to a reality that we consider natural. For this reason sometimes we can refer to this methodology with the expression "artificial experiment" (Gilbert & Troitzsch, 2005).

Axtell (2000) observes that in social sciences the technique of agent-based simulation can be used in three distinct cases. We have the first case when the social phenomena can be described completely using equations. In this case simulation can be used as an alternative way to reach the same result of a mathematical model. The second use arise in the case in which the mathematical model, adopted for the analysis of the social process, can be described but it cannot be solved. Finally, the third case of use exists when an explanation based on equation is not

possible and therefore agent-based simulation is the only way to study these kind of problems.

3.4 Complexity

The purpose of ABM is to see emergent behavior starting from numerous simple interactions: the complexity in fact is not in the parts but in their interaction. All the complex behavior that we can see in nature or in social systems are created through the application of simple rules to complex interactions. It is very difficult to trace the origin of a complex behavior, instead it can be easier to create a model based on simple behaviors which brings global characteristics similar to the one under study.

Comim (1999) examines the Santa Fe Approach (SFA), that is the visions of complexity in the economic research according to the Santa Fe Institute. He states that the main purpose behind this approach is the study of what is called complex adaptive systems or adaptive nonlinear networks (ANN). These networks or systems – which might refer to proteins, ants or economic agents – are characterized by different properties which define a logical realm where many relatively independent parts are highly interconnected and interactive (Cowan, Pines, & Meltzer, 1994). These ANN are characterized by six properties (Arthur, Durlauf, & Lane, 1997):

1. Dispersed interaction: global results are produced by the interaction of many diverse and dispersed agents acting in anticipation of other agents. The property of diversity produces *perpetual novelty* in aggregate behavior;

2. No global controller: interactions among agents are produced by mechanisms of competition and coordination without the help of central mechanisms of control and coordination;
3. Cross-cutting hierarchical organization: global organization consists of many different levels. Units at any level may serve as *building blocks* of units that are at a higher level;
4. Continual adaptation: internal states of agents change in response to changes in the environment, where individuals constantly adapt to their accumulated experience. There is scope for evolution and evolving processes where systems might be considered as responsive entities to environment dynamics;
5. Perpetual novelty: new behaviors and new structures may stimulate the creation of new behaviors and new structures, producing an ongoing state of perpetual novelty. Statics is replaced by dynamics;
6. Out-of-equilibrium dynamics: given the state of perpetual novelty, the economy does not operate close to any optimum equilibrium; instead, out-of-equilibrium is a pre-condition for the dynamics of complex systems.

Most of the studies on complexity involved the use of spatial models, that are characterized by possibility of a disperse interaction among agents. In this kind of models it is possible to see an emergent global structure starting from local phenomena. This concept can be found in the famous model of spatial segregation by Schelling. Schelling worked on this model in order to understand two types of segregation that often occur in the social world. The first one is racial segregation and the other one is segregation by income. When Schelling studied segregation there were no computers. He thought about this models

just using a sketched checkerboard and pennies and nickels (American five-cent coin) to represent different races. He thought of each person as being located on a checkerboard, so he described the whole city as a giant checkerboard. The checkerboard can have a person living there or it can be blank.

In Figure 1 the blue and red cells represent different races living in the same neighborhood whereas the white cell represent an empty space. How do these agents decide whether to stay or to move? The rules at the base of this model are really simple. The agent is happy as long as the percentage of similar agents around him exceed the threshold value, for instance 30%. If it is not the case he moves. In the case of Figure 3.1, considering a threshold value of 30%, agent X would wish to stay since the percentage of similar agent nearby is 3/7 (43%).

1 Blue	2 Blue	3 White
4 Red	X Red	5 Red
6 Blue	7 Red	8 Blue

Figure 3.1 – Schelling's checkerboard

What is so striking about this models? Schelling discovered that even a small preference by members of both group for living close to their own type would lead to a significant overall segregation. At the macro level we get segregation, but at the micro level, people are very tolerant. Can people whose neighbors are composed by 70% of other races

considered racist? The implication of this simple model can be very powerful. But what concerns us so much is the overall emergent behavior and the fact that the micromotives and the macrobehavior may not always align (Schelling, 1978).

An important feature of models with autonomous agents is that there is the possibility to make the model more complete and complex in an incremental way. Continuing with the example above, it could be possible to extend the model by taking into account social interaction or perhaps include environmental constraints. Starting from the basic structure of the model, then running the model and after being able to modify the characteristics of the agents and the environment does not have to be underestimated. As a matter of fact, it enables the researcher to build the model through a very natural process. Furthermore, agent-based models are very convenient since they can be decomposed in pieces and used for more than one model. If the behavior of the agent is simple it is possible to modify the model and use most of its parts.

Gilbert and Terna (1999) state that "the task of modeler is thus to define the cognitive and sensory capabilities of the agents, the actions they can carry out and the characteristics of the environment in which they are located, to set up the initial configuration of the system, and then to observe what happens when the simulation is run".

These kind of modeling require though specific tools to implement it. Not every programming environment is well suited for this purpose. Gilbert and Terna (2000) also add that "the characteristics of the software we use are crucially important in assuring the success of this third way of formalizing models. Only if we use high quality software we are able to communicate the details of our model, allow other scholars to replicate the results, and avoid difficulties in modifying poorly written code. The best way to improve the quality of the

programming is to choose an object-oriented language. This choice simplifies the translation of the problem into a set of agents and events. From a computational point of view, agents become objects and events become steps activated by loops in the program. In addition, in a fully object oriented environment, events (or time steps) can be organized as objects. The key term here is object: a piece of code containing data and rules operating on them".

3.5 Representation of ABMs

Edmonds (2012) points out another important feature of ABMs. He states that ABMs "allow for explicit representation and exploration of the complex relationship between individual behavior and society – the Micro and Macro link. It does it by representing the states and actions of each relevant social actor within a complex computer simulation". He continues "in this technique social actors (people, firms, parts of the environment, etc.) are each represented by separate entities (called 'agents') within the simulation. The interaction between the social actors are modeled as messages between the agents (Figure 3.2 and Figure 3.3). The entities are given behavioral schemata in the form of sets of inter-related rules and each agent has its own individual characteristics (e.g. memory, habits etc.). When the whole simulation is set going all these rules are repeatedly evaluated in parallel, so the effects of each rule will depend upon the past outcomes of rules, resulting in a complex sequence of interactions between the agents".

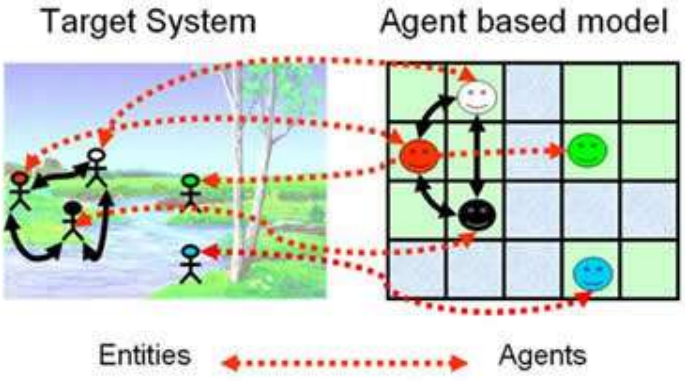


Figure 3.2 – Representation of the real world in an agent-based simulation

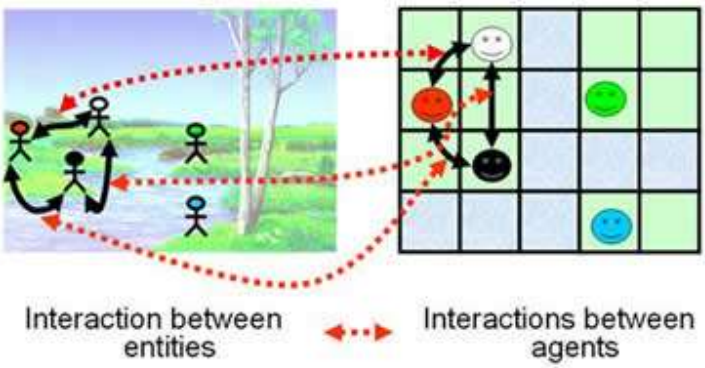


Figure 3.3 – Representation of the real world in an agent-based simulation

The technique of AMBs allows then to make precise and exhaustive representation of complex interactions typical of social sciences. Once the simulation is run, the result can be examined both at the individual and at the aggregate level. Furthermore, Edmonds (2012) argues that "this technique can be used in a very detailed and specific manner, enabling the coherent integration of various kinds of evidence (e.g. narrative, time-series, survey, SNA) within a sort-of dynamic computational description. This use contrasts markedly with the reductive modeling of Neo-classical economics, allowing the

simultaneous emergence of unpredictable social phenomena and the top-down constraint of individual action from norms and institutions. However their very complexity can make the simulations themselves difficult to understand completely and their level of detail can make validation challenging".

Another example of representation of an ABMs can be found in *Spatial finiteness, epidemics and climate shocks* by Barone, Gandolfo and Grimaldi (2011). In their project they created a world as a 32x32 square with centered origin. The environment is by assumption considered closed to the outside. The world is limited in the front by the sea (the blue line) and in the back and sides by rocks (the grey line). Both sea and rocks are inaccessible to the people of this world. During the set up each turtle originates on a random patch and a number of houses (equal to the initial number of farmer divided by the number of person per houses) are instantaneously created. As this process is irreversible, the patch on which an house is built, will never produce food again. In Figure 3.4, it is possible to view comparison between a stylized representation of the world and an actual snapshot of the NetLogo implementation.

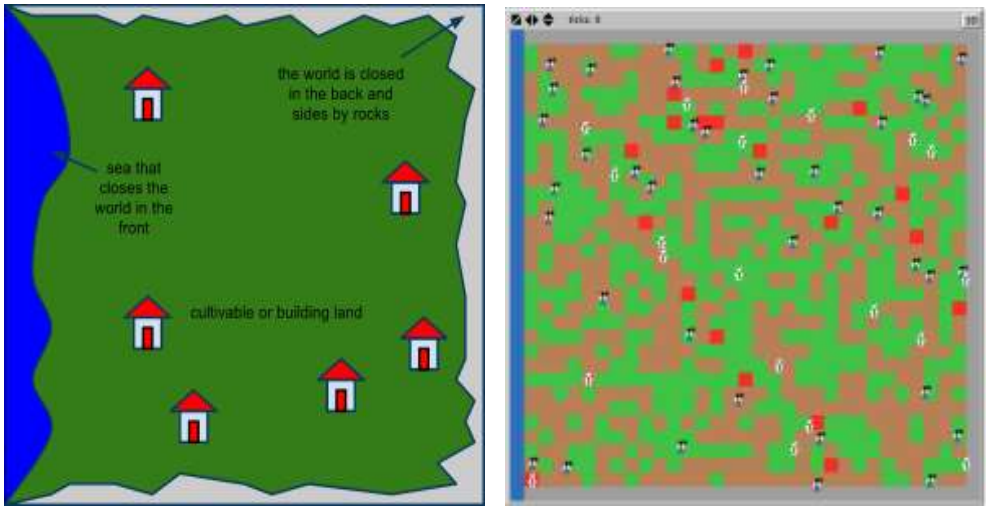


Figure 3.4 – Stylized representation versus NetLogo representation of the environment described in "Spatial finiteness, epidemics and climate shocks"

The same concept of ABMs representation can be applied for the model we talked about in the previous paragraph, the *Segregation Model*. In this model, a real representation of the ABMs, for instance for the city of New York, would look like the following picture.

Figure 3.5 – Racial segregation in New York

Figure 3.5 is taken from the website *Race and Ethnicity* by Fisher (2012). He created this map using data from Census 2000. The dots in the map represent city blocks which are considered to be populated by 25 people. Each red dot in this graph represents a city block that's majority Caucasian or White. Each blue dot represents a city block that's majority Afro American, each orange dot represents a city block that's majority of Latino and each green dot represents a city block that is majority Asian. We can see that New York is incredibly segregated and this could be due to the misalignment between micromotives and macrobehavior as we have explained in the paragraph before. In conclusion, as we have seen in these last two examples, ABMs allow for an explicit representation of reality and complex relationship that are present in our society. In this thesis project we will go even further, every agent in the simulation will be represented by a real object, a robot, that will behave in the same way as the agent in the virtual simulation.

3.6 Simulation in the social sciences: the third way

Social sciences, conversely of science of nature such as physics and chemistry, don't have the possibility to verify their hypothesis and theories in a laboratory. Computer simulations can be an important tool of research for social scientist. It can be used to satisfy the need of a laboratory in which to make controlled experiments.

Ostrom (1988), in his paper *Computer Simulation: the Third Symbol System,* proposed the use of computer simulation as a tool that social scientists can use to formulate theories of complex and interdependent social phenomena. Ostrom (1988) believes that natural language and mathematics are not the best tool to study social sciences: the former is very flexible but it cannot be verified with numbers, the latter can be

verified with computational tools but often make use of hypothesis that are unrealistic. Computer simulation is the right trade-off between natural language and mathematics since "it can be used for representing both qualitative natural language constructs and quantitative, mathematical constructs" (Ostrom, 1988). Computer simulation can therefore be thought as the third symbol system, that is the symbol through which we express our theoretical ideas and communicate them to others in the field. Ostrom (1988) outlines why computer simulation can be used to understand the dynamics under the five fundamental complexity of social behavior:

1. *Multiple manifestation.* Verbal theories researchers make no attempt to provide a theory of when one vs another manifestation of behavior will emerge. The third symbol system can help to provide a systematic theory that address this complexity.
2. *Qualitative structures.* Qualitative structures are important in the study of group processes. The emergence of roles and dominance structures, the patterning of formal and informal communication actions and the sequencing of communication content are all central to the understanding of social aggregates. Artificial programming languages provide an excellent symbol system for the theoretical representation of such structures.
3. *Modeling of response systems.* The third complexity pertains to the manner in which a latent variable is related to its observable variables. Social psychology is weak in the area of linguistic communication. This is an important response systems to understand; social behavior is dominated by people talking to one another. Yet we know very little about how thoughts, feelings, and plans get conveyed to other people. Computer

simulation provides a symbol system adept at formally representing these kind of processes. A great deal of work in artificial intelligence has already been devoted to understating natural language processing and pragmatics in linguistics. In terms of better understanding nonverbal behavior, theorist could benefit from drawing upon current work in robotics. This is a field where workers are trying to model the relation between motor systems (which in nonverbal behavior would include facial expressions, eye contact, and posture) and control systems. Some workers believe that computer simulation are unable to model affective and motivational processes. But this is completely untrue. Any theory of emotional dynamics that can be articulated in a verbal form, can also be expressed through the third symbol system.

4. *Interfacing of multiple systems.* It refers to the interdependences among two or more latent variables. The best verbal theories about for instance aggressive behavior, describe the emergence of that behavior as a concatenation of independent process. This theory is inadequate when we consider the case of three or more independent variables. These type of issue are at the heart of understanding the whole person as an integrated system. Computer simulations are well suited for specifying multiple interdependencies.
5. *Time.* Social behavior is extensive over time. The course of all social interactions involves continuous action on the part of all participants. Theory in social psychology will need to deal effectively with how people make use of this continuous social feedback in guiding their own actions. Time is the carrier of the other four complexities. Computer simulations come into play

when the theorist wants to combine modules into a larger scale model of social behavior.

3.7 Environment-Rules-Agents (ERA) scheme

During the simulation agents have to make decisions of different types. Terna (2000) proposes a general scheme that can be employed in building agent-based simulations, the Environment-Rules-Agents (ERA) scheme. The scheme is illustrated in Figure 3.6. According to this structure, agents are not able to communicate directly with each other, but they can communicate only through the environment. This makes the coding less complicated than it would be if agents could communicated with each other directly.

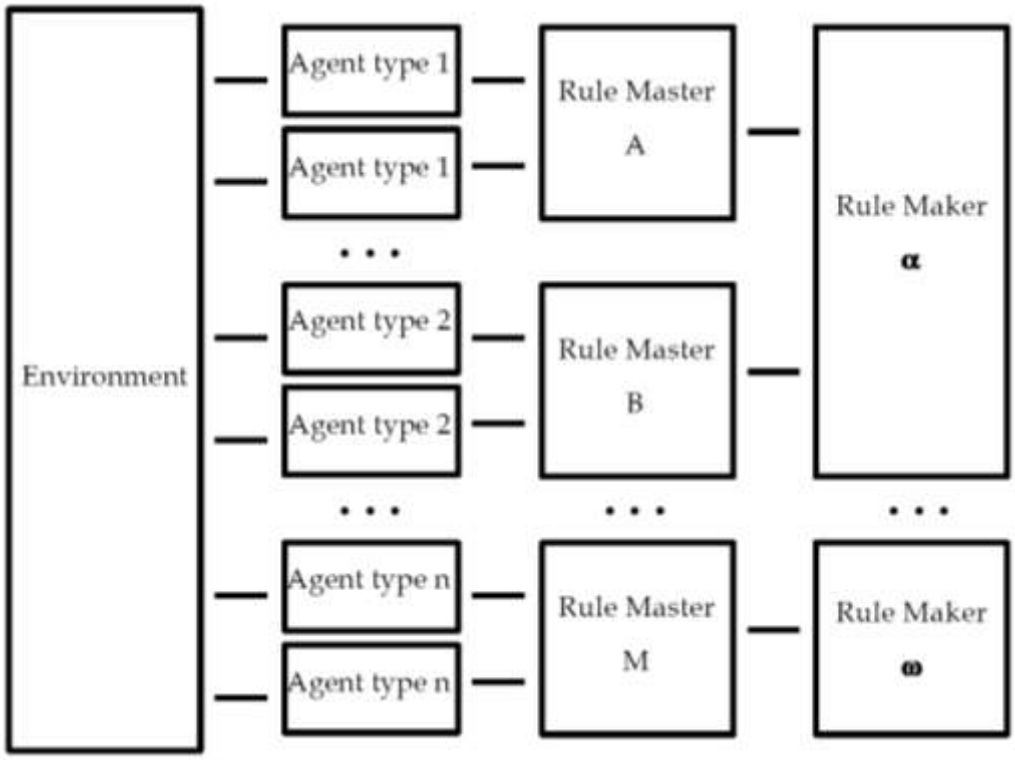


Figure 3.6 – ERA scheme, Environment-Rules-Agents

The aim of this structure is to manage the four different layers in the process of building an agent-based models. The four layers are:

1. the environment layer, in which agents can interact with each other;
2. the agents' layers, in which agents belonging to different classes are created. They can inherit properties, features, data and methods from more general classes;
3. the layer that manage the way through which agents decide their behavior. Agent behavior is determined by external objects, named Rule Master, that can be interpreted as abstract representations of the cognition of the agent.
4. the fourth layer concerns the rules creation. In this layer, Rule Master communicates with Rule Makers, whose role is to modify the rules mastering agent behavior, for instance by means of a simulated learning process.

The use of the ERA scheme brings many advantages. Terna (2000) states "although this code structure appears to be complex, there is a benefit when we have to modify a simulation. The rigidity of the structure then becomes a source of invaluable clarity", he also adds "a second advantage of using ERA structure is its modularity, which allows model builders to modify only the Rule Master and Rule Maker models or objects whenever one wants to switch from agents based on neural networks, to alternatives such as production systems, classifier systems or genetic algorithms".

4

Theory of the Actor and Social Simulations

4.1 Theory of the actor as explicit model for the simulation of social behavior

In the 1980s, with the evolution of artificial intelligence, a model for the simulation of social behavior has been developed in the University of Turin. This model, called EGO and designed by sociologist Luciano Gallino, methodologically arises in the family of computational models and it has the purpose of simulating – using artificial intelligence techniques – the mind of a real actor who must make a difficult decision. EGO is relevant to this thesis project since – as we will see later – it gives the body to the social actor. The introduction of the body

as a referent that the individual must take into account in making decisions is a major breakthrough in the field of social sciences. This model is therefore a useful means to move conceptually from ABMs to the world of robotics. In the following section I will provide the main theoretical lines of EGO. In particular, we will see what Gallino means with model, social actor and how he defines the referents of EGO.

"Given the many definitions recurring in scientific language, I precise that for the word "model" here I mean a unitary configuration, structured in a rigorous way under the linguistic profile and the mathematical formalization of variables and intervariable relationships *similar* to those that are supposed to be essential to explain dynamics, the transition from state to state, the behavior of a particular field of events, which is part of a larger and more permanent domain of events – a universe of objects designed to vary and relations between them – covered by a scientific theory. Every theory refers to a domain decomposable into *n* fields, each configurable with *m* models, and will generate *n+m* models. Each model comes with a logical and an empirical part. In the logical part, variables have a general character (can refer to any instance in a domain *D)* and are given with the full rank of positive and negative changes. In its empirical part, the model is "loaded" with variables of a special nature (referring to a particular instance *d* of the domain *D)* and are given with a defined mode. Introducing arbitrary variations in a logical model, and according to the relations created, we produce simulations of the dynamics of the states of the chosen field. Introducing variations derived from observations, and thus transforming the model from logic to empirical, we produce explanations and predictions. Defined in this way, the model is an essential tool for the construction and ongoing revision of a theory,

since it is the interface that must be placed between theory and reality" (Gallino, 1987).

As we have already said, this model methodologically can be put in the family of computational or cognitive models and it is proposed to simulate – using artificial intelligence techniques – the mind of a real actor who must make a difficult decision, and aims at the control of a theory of social actor.

"With theory of the actor I mean a theory that can explain and predict the ways in which an individual participates in one or more social systems, has acted or will act in different situations, in the presence of different initial parameters of his condition, including among others, internal states such as emotions, needs, goals, values, interpretive frameworks, processes of reasoning. Such a theory would seem to be a component of every sociological theory, especially what many still consider the theory that best characterizes our discipline, which is the theory of social systems. Lacking of a theory of the actor, the theory of the social system is transformed into a kind of implicit uncritical behaviorism. Situations, socio-anagraphic data, affiliations of class, are configured as inputs in a black box, of which the contents is unknown and it is precisely the missing actor, and from which emerge outputs in the form of strikes and votes, migration and deviant behavior, religious practices and ideologies " (Gallino, 1987)

Each instance of EGO, simulates a particular subject that in any situation tends to optimize, through its decisions, its biocultural reproductive success, that is the survival at the same time physical and symbolical. The hypothesis explored by Gallino is the following. Whatever is the decision he wants to make, a social actor simply try to survive: as a physical body, if possible (through some child); as individual sign left in the minds of others, or through ideas, the

elements of culture that shares with others. If the attempt to survive in a given form is too expensive or uncertain, he tries to achieve the same purpose in a different way. What is important for the individual is the global space, material and symbolic, in which he is represented as a being who continues to live, to exist in one form or another.

4.2 Psycho-sociological version of EGO

EGO in its psycho-sociological version (which traditionally uses the language of the humanities) argues that EGO (each instance of the model) evaluates in a more or less conscious way the consequences of his choices on a group of primary objectives in respect of which behavior is just a tool. Borgna (1992) states that, as regards the psycho-sociological version of the model, "these objectives can be defined as the skills acquired by living systems through evolution that have turned into some kind of background framework, of preferential purposes, in respect of which any other objective is replaced. They are three:

1. *survival,* defined as the ability of a system to last beyond the critical situations, by cushioning the capacity of the external environment;
2. *persistence*, i.e. the capacity to recognize oneself and be recognized by others as the same individual at different times;
3. *replication*, the ability of a system to produce copies of itself.

Overall, these three objectives constitute the ability to live and continue to live. EGO considers the consequences of his behavioral choices in terms of increased or decreased chance of survival, persistence and replication (from now on SPR) of four reference systems, its referents: its *body,* its *system of orientation,* its *biological related* and its *culturally related* (Figure 4.1).

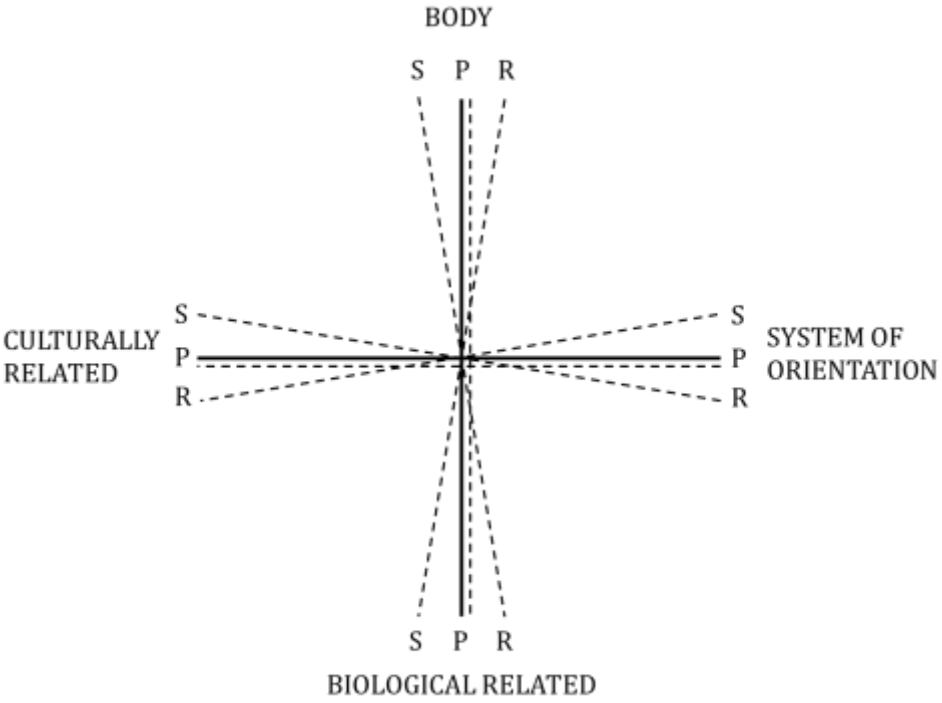


Figure 4.1 – Space of the reproductive bio-cultural success

The first two represent the informational structures, respectively biological and cultural of EGO; the other two are groups of individuals to which EGO constantly refers during life.

The orientation system (OS) can be defined as the informational structure which assumes an individual normally acculturated, which, thanks to it, retrieves the ability to orientation who has lost in the course of evolution. We can distinguish at least three components in the orientation system, corresponding to the three basic types of orientation that is necessary for human being: emotional, cognitive and evaluative.

The biological related (BR) of EGO are his blood relatives, and are divided into direct and collateral.

The culturally related (CR) are individual members of many types of groups with whom EGO shares some cultural traits, leading him to identify himself in some way with them and refer to them using a "we." The affinity can be carried out in at least three levels: that of the community (religious, territorial, ideological or political, military, business, scientific), that of the association and that of the group".

The fundamental hypothesis of the model used to explain the choice an actor of a certain social behavior is the following: *if the total value of the benefit in terms of SPR for one or more referents from a given behavior is bigger than the value of the total cost for the other referents, EGO will adopt that behavior.* In this way EGO "aims to live as long as possible, in time and in space, as a biological entity and/or as a cultural entity, as a system-individual and/or as part of a biological (the population of relatives) or cultural system (the population of similar cultural or symbolic)" (Gallino, 1987).

This is the hypothesis of the maximization of the overall fitness: the combined pressure of the interests of the individual, in the dual form of the maximization of their genetic representation and its cultural representation, predisposes him to behavior focused, consciously or unconsciously, to maximize its organic fitness (which has the body as selection unit), genetic (which has the gene as selection unit), mental (which has the system of orientation as selection unit) and culturgenetics (which has some traits of culture as selection unit). All the above together considered, determine the overall fitness of an individual.

Defining the space of the reproductive bio-cultural success as the space bounded by the referents that surround the behavioral choices of EGO, we can say that the objective of the actor is to optimize the global space of its bio-cultural reproductive success.

When a referent is likely to lose chances in terms of SPR, then Ego chooses a behavior that allows him to recover such damages in the form of SPR benefits of another referent: therefore, EGO is a decision maker that face from moment to moment the need to choose between survive and persist longer, or replicate more widely, as an organism; or as a person culturally differentiated; or through its biological related, bearers of the same genes; or through its culturally related, bearers of the same conceptual material.

4.3 Artificial Intelligence version of EGO

There is also a version of the model expressed in the language of artificial intelligence that is especially useful to write computer programs to simulate the behavior of the actor in question. The main criteria that guides the design of such a system are: (i)the psychophysical monism (the model does not intend to separate the mental from the physical structures from which it emerges); (ii) the evolutionary paradigm (requires that the model is equipped with structures and capacities similar to those that evolution has endowed the human brain); (iii) the strong equivalence (the model is not limited to a shallow simulation, but it intends to study the sub-systems as well); (iv) the extended computational approach (according to which it is possible to take the computational systems – the computer – as source model of a model of the mind and behavior).

As regards the general internal structure of EGO, "the functional subsystems are as follows:

- *Representation and recognition of referents:* it is the subsystem that constructs representations of entities to determine whether they should or should not be recognized as belonging to a class

of referents. If it recognizes them as such, it assigns a value to their SPR and communicate it to other functional subsystems;

- *Representation of situations:* is the subsystem that using semantic networks, represents both current and future situations in which the individual is or will be located as a result of his behavioral choices. These representations comprehend the location of the individual simulated the different situations as well as the positive or negative states in which he is or would be located and its position in relation to social systems interested. Then they are passed to the next subsystem for interpretation;
- *Schemes of interpretation:* is the subsystem that constructs or calls from memory schemes used to assign a meaning to situations in terms of risks or opportunities which reduce or increases the chances of SPR of the referent. For instance, when the overall assessment of the situation gives a number of benefits that exceeds the amount of costs then the module applies interpretive schemes moderately promising; but if the costs appear to outweigh the benefits it would apply to moderately threatening. By transferring to the module "affective states" the forecasted costs and benefits to SPR of referents and to "behavior plans" messages that reduce alternatives behavior;
- *Emotional states:* is the subsystem that activate the emotional states that predispose the system to a behavior that can eliminate the risk or take the opportunity. For instance, take the case in which the previous module has activated a promising interpretative scheme: the subsystem will modify the affective states, enhancing the positive and reducing the negative ones; the reverse is true if there is a threatening interpretative scheme. The output of this module serve as input for the

"scheme of interpretation", to which indicates the variation of emotional states, and the module "plans of behavior", to which indicates the predisposition or the inhibition of behavioral classes;

- *Plans for behavior:* is the subsystem that generates behavioral plans and represents the impacts of the alternatives on the basis of the information received from other subsystems, negotiate conflicts between plans of behavior designed by the instrumental subsystems and narrows the selection of the plans according to the predispositions and inhibitions.

When the subsystems described carried out their computations, the *central decision maker* module intervenes. It makes the choice of the behavior that optimizes the bio-cultural reproductive success of the entire system through the use of the following subsystem:

(i) *evaluation of consequences:* it analyzes the different levels of behavior and the consequences of each of them on the system, comparing them with each other;
(ii) *plans of alternative behavior*: it formulates alternative plans;
(iii) *decision:* that identify, among the various possibilities, the plan of conduct to implement and it communicates it to the subsystem of implementation.

The system has another subsystem, t*he initializer,* that acquires information related to the initial parameters of the individual who intends to simulate (sex, age, etc.). EGO is able to simulate many different EGOs. In other words, EGO can simulate social actors with different initial parameters. This subsystem is external, in the sense that it does not propose the simulation of the mental function, it

represents a technical necessity for the real system to operate " (Gallino et al, 1992).

In summary, the model is designed to simulate an actor confronted by difficult decisions, who must choose whether to leave one situation for the other possible situation (perhaps leaving a social system to occupy another) optimizing "the distribution of costs and benefits in terms of SPR (Survival, Persistence, Replication)" (Gallino et al, 1992).

In my thesis, EGO is seen as an explicit model that can be used to assess the continuous tensions among the four type of referents and their ultimate goals, in the decision-making processes that I have discussed in the chapter regarding game theory, in particular the prisoner's dilemma. While in the prisoner's dilemma game two players chose a strategy of collaboration or defection according to the payoff matrix, with EGO each actor makes his choice taking into consideration the costs and the benefits that his decision bring to the referents. Relevant to this work is the fact that EGO, in his choice, takes into account his body and the body of his biological related.

In the next chapter, I will introduce the use of robotic in the economic field and therefore I will have to take into account the body of the actor/robot in the simulation.

5

Economics and Robotics

5.1 Simulation for economics, robotics and AI

The idea of this thesis slowly arose in my mind. I believe I had the opportunity to start this project because of three converging factors. The first factor was the opportunity to learn about simulation for economics, agent-based models, complexity and emerging behavior in the course Simulation Models for Economics held by Prof. Terna. In addition, during this course I had the opportunity to get familiar with more than one programming languages. The one we studied in depth was NetLogo, which is a programming language particularly suited for simulation in the social science and the exploration of emergent phenomena.

The second factor that contributed to the idea behind my thesis project was the discovery of Lego Mindstorms NXT (Figure 5.1), a programmable robotics kit released by Lego in July 2006.

Figure 5.1 – Lego Mindstorms NXT

I found out about Lego NXT in summer 2011, right after finishing my course in Simulation Models for Economics and I immediately got interested in it. I have always been a Lego fan and I decided to buy one because I thought it would be a perfect tool to enhance my programming skills while have a real life feedback of what I was doing. In fact, the modularity of Lego kit allows to start from very simple robots that can be used in a turtle-like manner, to very complex one that can solve very hard task, for instance the Rubik Cube (http://youtu.be/bVtXvgbc8nY). Furthermore, on the Internet, there is a vast and active community of developers and Lego NXT fans that share everyday new programs, new robots and precious advices. Another feature of Lego NXT that influenced my purchasing decision, was the possibility of programming the NXT Intelligent Brick (this is how the computer is called) with a large variety of unofficial languages such as NXC, NBC, leJOS NXJ, RobotC, Robolab, NXT-Phyton and not just the official one provided by Lego.

The third factor that contributed to the idea behind my thesis project was the reading of the paper *Behavioral AI Experiments and Economics* presented by Birk and Wiernik (1996) during the 21th European Conference on Artificial Intelligence. The paper is about the promising possibilities of cooperation between behavioral AI and economics and describes an experiment in which robotic agents live in an ecosystem together with competitors. Birk and Wiernik begin talking about the increasing interest in the "use of Alife-tools to test economic models and to justify their underlying assumptions on an experimental basis". They continue "a common example is the attempt to explain and justify Adam Smith's "invisible hand hypothesis". Important work is being done for example by the economist L. Tesfatsion (Tesfatsion, 1995) and goes back to the beginning of evolutionary computing by J.H.Holland (Holland, 1992). Economic analysis has avoided questions about the way in which economic agents make choices when confronted with an evolving world. Holland was among the first ones to investigate economic areas such as technological innovation, which he did by viewing economic systems as complex adaptive systems. The advantages of using Alife-techniques from the economist's point of view are as follows. Economics tries to model real-world processes. But it is usually impossible for the researcher to "tinker" with these processes. He cannot change parameters intentionally or make "repeated runs" with identical conditions to obtain desired data. But the work so far has only relied upon simulations. We believe that software simulations are no real alternative. They bear the great danger that they – implicitly – match the model, whose validity should be tested. Behavioral AI experiments, on the other hand, are a feasible alternative. They are complex enough to be meaningful and they are open for controlled changes. The most important feature is that they are based in the real-

world. Though set-ups are open for (any) changes, some are much more feasible than others. For instance, a trade of information is easier to implement than a direct exchange of energy between real robots. Thus, the researcher must (re)establish which constraints and settings are "natural" or "easy given" and therefore commonly accepted prepositions" (Birk & Wiernik, 1996). The experiment was designed by L. Steels and D. McFarland and it was structured in the following way. The basic ecosystem consists of small autonomous vehicles, a charging station, and competitors (Figure 5.2). Videos of the robots working in the ecosystem as described can be found at the following links: http://youtu.be/9xSGSNIzE-I and http://youtu.be/SZfHuqz3u3g .

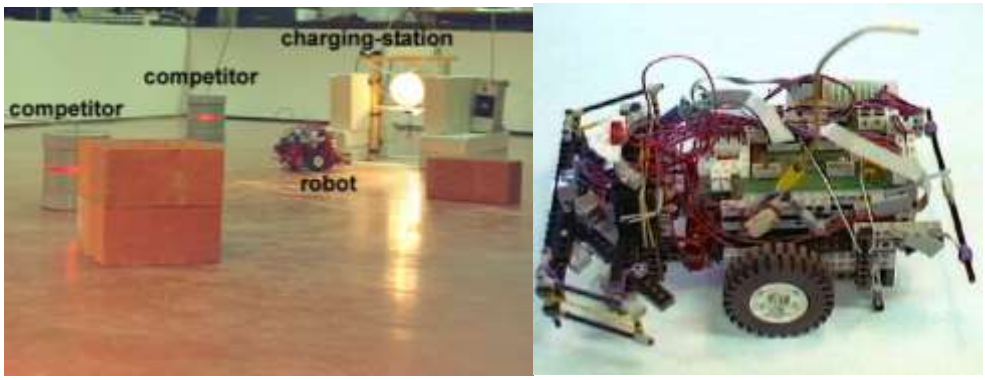


Figure 5.2 – Ecosystem with autonomous robots and competitors (left) and close-up of the robot (right)

The vehicles are small Lego-robots, which are capable of recharging in a charging station. The competitors in the ecosystem are boxes housing lamps. They are connected to the same global source of energy as the charging stations. Therefore, they compete for the same energy of the vehicles. But if a robot knocks against one of these boxes the light inside the box dims and more energy is available for the robot in the charging stations. If a robot does not "fight" the light slowly "recovers". The "fight back" mechanism was not explicitly programmed, but it emerges out of two simple behaviors: retracting when hitting an obstacle and being

attracted by the light of the lamps. The economics behind the experiment comes from the idea of "behavioral resilience" that consists in the extent to which an activity resists squashing by other activities. If an animal does no feeding, for instance, the cost will be relatively high, but if it abstains from washing itself, the cost will be relatively low. Three different activities of the robot are observable:

1. Eating (recharging)
2. Hunting (pushing against the competitive lamps)
3. Free-time (anything else)

In the experiment Birk and Wiernik tried different growth rate for the competitors recovery (how fast the light recovers) and they noticed that hunting activity starts to squash free-time activity as the growth rate of the competitors increases. Thus, they were able to observe directly the behavioral resilience by making a sequence of experiments with increasing environmental pressure (by changing the growth rate of the competitors).

They came to the conclusion that the hunting activity has relatively high resilience, whereas free-time has a low one. Birk and Wiernik used behavioral resilience as an indirect measure of elasticity of demand. In this case, the hunting activity showed a relatively inelastic demand function and free-time showed a relatively elastic one, meaning that if the price of hunting goes up by one percent, on average, hunting will decrease by a smaller percentage. The opposite is true for the free-time activity.

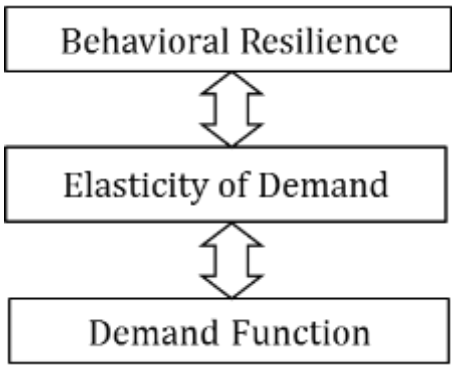


Figure 5.3 – Relationship between behavioral resilience, elasticity of demand and demand function.

The increasing environmental pressure lead to the disappearance of free-time activity, implying, in economic terms, that free-time is a luxury activity. Furthermore, Birk and Wiernik conclude with a consideration about economic models and their assumptions: "static economic models have their limitations with respect to the ecosystem considered. For instance, simple utility theory assumes substitutes, which are not "natural" in our ecosystem. In addition, the robot's activities are sequential. In order to "eat", the robot must first "hunt". Common utility theory does not take this into account" and they continued "in our experiment, we experienced limitations in our set-up as well as in the models used. We believe that this is not a drawback, but actually the strength of our approach to merge the fields of economics and behavioral AI. This leads to new insights on which constraints and settings are "natural" or "easy given" and therefore commonly accepted prepositions with respect to both communities. So, not only can a model's validity be tested, but also its plausibility" (Birk & Wiernik, 1996).

To summarize, three factors contributed to the idea behind my thesis project:

1. The course Simulation Models for Economics at the *University of Turin,* in which I had the opportunity to get familiar with computer science and problems on complexity;
2. The discovery of Lego NXT and its versatility in terms of programming languages, hardware and not less important their low cost compared to standard robotic kits;
3. The experiments implemented by the *Artificial Intelligence Laboratory* at the *Vrije Universiteit Brussel,* in which autonomous robotics agents built with Lego were used to investigate their behavior in real-world environment in such a way to create complex ecosystems.

These three elements really made me interested in exploring how autonomous robotic agents could help the current development of simulation models for economics. Can the real-world uncertainty and imprecision help the cause of modelers? What is the added value of engaging in a robotic simulation experiment?

We already read in Ostrom (1988) that modelers "could benefit from drawing upon current work in robotics, a field where workers are trying to model the relation between motor systems (which in nonverbal behavior would include facial expression, eye contact, and posture) and control systems". Currently, great effort is being put into the project iCub, a humanoid robot (Figure 5.4) developed at IIT as part of the EU project RobotCub and subsequently adopted by more than 20 laboratories worldwide (Robotics, Brain and Cognitive Sciences dept. at

the Istituto Italiano di Tecnologia, 2012). The goal of RobotCub was to study cognition through the implementation of a humanoid.

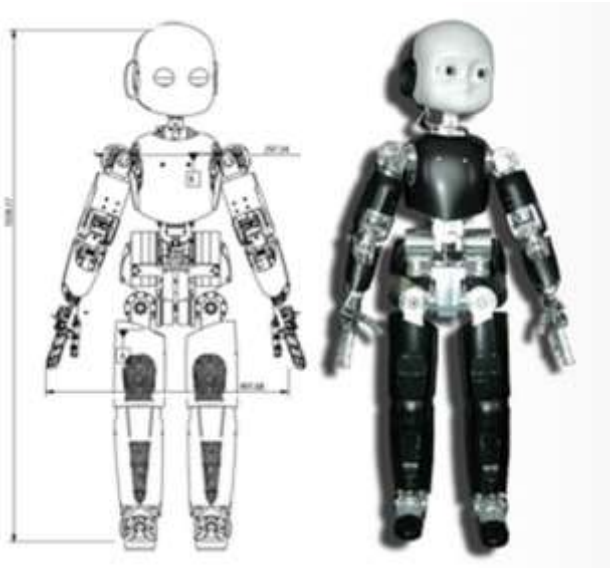


Figure 5.4 – iCub, a humanoid robot

The motivation behind the strongly humanoid design "is the embodied cognition hypothesis, that human-like manipulation plays a vital role in the development of human cognition. A baby learns many cognitive skills by interacting with its environment and other humans using its limbs and senses, and consequently its internal model of the world is largely determined by the form of the human body. The robot was designed to test this hypothesis by allowing cognitive learning scenarios to be acted out by an accurate reproduction of the perceptual system and articulation of a small child so that it could interact with the world in the same way that such a child does" (Metta, Sandini, Vernon, Natale, & Nori, 2008). It is interesting to see that scientists need a "body" in order to study cognition, a purely abstract field.

Another important advantage that could come from the use of autonomous robotic agents is the educational one. In the paper *Bifocal modeling: a framework for combining computer modeling, robotics and real-world sensing,* Wilensky and Blikstein (2007) explain a very useful concepts for our purpose. They describe how agent-based models and robotics can be complementary, and how they can be useful in the educational context. Multi-agent modeling dramatically changed scientists' mindset and practice. Instead of departing from very complicated macroscopic behaviors, scientists started to use massive computation power to simulate systems with thousands of simple agents, behaving according to simple rules (Blikstein & Wilensky, 2006). They discuss how to link robotics/sensing and multi-agent computer simulation by creating a platform that enable learners to connect virtual and physical models as to validate, refine, and debug their computer models using real-world data. Wilensky and Blikstein observed two different group of students involved in the realization of a project for the course *Learning Environment Design.* One group created a purely virtual model, whereas the other group had to make a physical + virtual model (bifocal model). The computer program used for the virtual model was NetLogo (Uri Wilensky is the author of NetLogo), the same programming language that I am going to use for my simulation. Wilensky and Blikstein found out that the second group of students were much more involved in the project and started to reflect about epistemology of modeling, about assumptions and limitations of the computer models. Furthermore, students engaged in the physical + virtual model were able to calibrate and adjust the coefficients for the virtual model much more carefully. An interesting project undertaken by the second group of students was related to the energy loss, which is a fundamental process in physics that is usually ignored in computer

models. On-screen agents can move freely in the virtual world without ever experiencing any friction, unless the modeler decides to include it in the model (Blikstein & Wilensky, 2007). When dealing with the physical world the choice does not exist: energy loss and friction are a fact of nature you have to deal with. As a matter of fact, the students were unable to predict accurately how a virtual sphere starting from inclined plane would travel, compared to a real one, just using the conventional Newtonian equations (Figure 5.5).

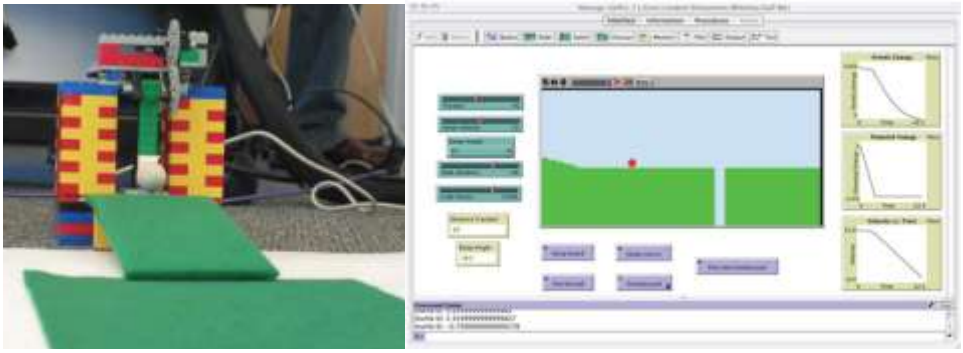


Figure 5.5 – The Newtonian motions apparatus (left), in which a sphere is launched at the top of the ramp by a robotic arm, rolls down, and eventually stops on the green carpet, and the NetLogo model (right)

Setting the project in this way, students were able to identify the causes of mismatch, which are normally overlooked in introductory courses. They realized that the amount of new variables needed to make the virtual model as close as possible to reality would be enormous: air resistance, irregularities of the floor, variability in the initial impulse of the robotic arm and even the path of the sphere movement was never completely rectilinear, just to name few. At the end of the project, the

students that created both a physical and a virtual model were extremely aware of the limitations and the dangers of the latter approach. Wilensky and Blikstein concluded that the *bifocal* approach enabled students to rapidly investigate their hypotheses and observe alternative outcomes, debugging their own models and algorithms. Furthermore, by using "the power of computation and representation, bifocal modeling constitutes a research tool for students which offloads aspects of the interpretative and menial encumbrance of scientific practice, freeing cognitive resources that can be allocated in the direction of validation of the hypotheses" (Blikstein & Wilensky, 2007). In developing this research we will see how models can be useful to educate the general public. This is one of the main purposes why I decided to make a simulation in the real world using the Lego NXT. Real robots will not only allow us to capture the imperfection and variability typical of the real world, but they will also assist us in explaining an agent-based simulation to non-specialists. Many people struggle to comprehend that virtual agents in the computer can make autonomous decisions, and instead believe that the actions the agents are making is just because they are programmed to do so. The model reproduced through autonomous robotic agents, can thus be a perfect complement for explaining sophisticated and ambiguous interactions as the one of the prisoner's dilemma game to person that are not in the field.

5.2 A robotic experiment with the introduction of artificial neural network

An important paper about cognitive science and adaptive behavior is *Robots that have emotions* by Parisi and Petrosino (2010). In this article, they describe how simulated robots can be said to *have* emotions. They

set up five different simulations in order to discover the differences in the behavior of robots that are endowed with and emotional circuits and robots that don't. In order to construct robots that *have* emotions (i) they created robots that have many different motivations that compete with one another for the control of the robot's behavior so that the robot has to decide which motivation should control its behavior at any given time; (ii) they included a special *emotional circuit* in the neural network that controls the robot's behavior.

5.2.1 The motivational level and the cognitive level of behavior

External stimuli and information by the senses are not the only determinant of behavior. For instance, an animal will respond to the external stimulus of food only if the animal is hungry. So, while explaining behavior we also need to take in to consideration the motivation which currently control the behavior. Parisi and Petrosino state that "it is the currently active motivation, together with the sensory stimuli, that makes it possible to predict what the animal will do. Stimuli are not enough". Furthermore, a behavior can be controlled by only one motivation at a time, for this reason animals possess a mechanism for deciding which one of their different motivation should control their behavior. This level of functioning is called *strategic* or *motivational level.* A second level of functioning is the *tactical* or *cognitive level* and it is responsible for the execution of the behavior that satisfy the motivation. These two levels are complementary and fundamental for the survival of an individual. Parisi and Petrosino (2010) assume "a simple, implicit, mechanism for deciding which particular motivation will control the robot's behavior at any given time: all the robot's motivations have a quantitative level of intensity

and the motivation which wins the competition is the motivation which currently has the highest level of intensity". As we have already seen in Birk and Wiernik (1996), the study of motivations can be made through the use of competing motivations in animals or robots. Parisi and Petrosino will use the same approach.

5.2.2 Emotions

Parisi and Petrosino define emotions as follows: *states of animal's body/brain that increases the correctness and effectiveness of the motivational decisions of the animal by influencing the current intensity of the different motivations*. Emotions are part of the strategic level at which alternative motivations compete for the control of the animal's behavior but motivational decisions need not to be necessarily accompanied by emotional states. Emotions tend to emerge in complex situations in which animals have to make fast and correct decisions. Slow decision making and errors in the motivational decision could compromise an animal's survival and reproductive chances. Parisi and Petrosino hypothesize that "emotional states are an evolved mechanism for making the strategic level of behavior more effective, less subject to error, and faster". In their research, they show that adding an emotional circuit to the neural network of the robots, they are able to achieve a higher level of fitness.

5.2.3 Robots that *have* emotions

A robot in order to have emotions needs to have motivations. In their experiments Parisi and Petrosino use an artificial neural network to simulate a very simplified brain in which behavioral and motivational decisions are implicit (and not explicit as in a rule-based approach).

5.2.4 Robots that have to take motivational decisions

Parisi and Petrosino run the simulations through Evorobot, a computer program that simulates Khepera Robots (Figure 5.6). This kind of robots have a cylindrical body of 55mm diameter and 30mm height, light, infrared, and ground sensors, and two DC motors with incremental encoder controlling the two wheels.

Figure 5.6 – Khepera Robots

The neural network that controls the robot's behavior has two sets of inputs: (i) one encodes the perceptual properties of the different objects which are present in the robot's environment (sensors); (ii) one encodes different states of the robot's body (hunger, thirst, pain). Furthermore, each internal unit of the artificial neural network and each motor has a *bias* that defines a spontaneous level of activation. A genetic algorithm, with a population of 100 robots that reproduce selectively based on their individual fitness and with random changes in the inherited genotype, has been used to develop the weight of the connections of the neural network. The experiments are set up such that all robots have to satisfy two different motivations:

1. some have to look for food and for water;
2. some for food and try to escape from a predator;

3. some for food and for a mating partner;
4. some for food and take care of their offspring;
5. some for food and rest when their body incurs in some physical damage and has to heal from the damage.

A robot cannot purse both motivation simultaneously, so at any given time the robot has make a choice.

The first type of robots needs both energy and water to survive and the environment is a seasonal one, meaning that in one epoch there is more food than water and in the following epoch there is more water than food (a robot's life lasts for 10 epochs, each of a maximum of 1500 time steps). The robot periodically generates one offspring, hence, the robot's fitness is equivalent to the length of its life. The robot's body has two internal stores having the same size, one for energy and one for water. In this experiment the behavior that evolves in the robots can be described as "Look for food if you feel more hungry than thirsty and look for water if you feel more thirsty than hungry".

The second type of robots in order to survive have to eat food and at the same time have to avoid being killed by a predator. In this case, the role of external stimuli in deciding which motivation wins the competition with other motivation is even stronger. The results of this experiment showed that at the end of the simulation most robots are able to make the correct decision of quickly switch from looking for food to flying away from the predator.

The third type robots, in order to transmits its genes to the next generation, have to find a mating partner. In this case, the fitness of the robot is the number of mating events. The mating partners are nonmoving object with a different color form food tokens. At the end of the simulation, thanks to two input units encoding the location of the

mating partner and one input unit encoding the level of energy in the body (hunger) the robots are able to choose appropriately at any given time between the motivation to eat and the motivation to find a mating partner.

The fourth type of robots live in an environment containing two bounded zones, a food zone and an offspring-care zone. For each time step spent in the offspring-care zone, the survival chances of the offspring already generated increase by some fixed quantity. Fitness is not the length of their life or the quantity of food eaten or the number of mating episodes but the time spent in the offspring-care zone. The robot must then find the right balance between the competing motivation of eating and take care of the offspring. The result of the simulations show that the robots evolve the capacity to decide between eating, and thus remaining alive, and taking care of the offspring and insure their survival.

The fifth and final type of robots live in an environment in which they have to eat in order to survive and at regular intervals their body may incur in physical damages. The robot has to rest or reduce speed to heal. At the end of this simulation, robots appear to be able to take the appropriate motivational decisions between resting (healing) and looking for food and ignoring the pain.

5.2.5 Adding an emotional circuit to the robot's neural network

In all the above experiments, robots cannot be said to have emotions. In order to add emotion, Parisi and Petrosino added and emotional circuit to the Artificial Neural Network and afterwards they compared the simulations. The emotional circuit is made up of one or two emotional

units to which some of the input units send their activation and which in turn send their activation to the internal units (Figure 5.8b) or directly to the motor units (Figure 5.8c).

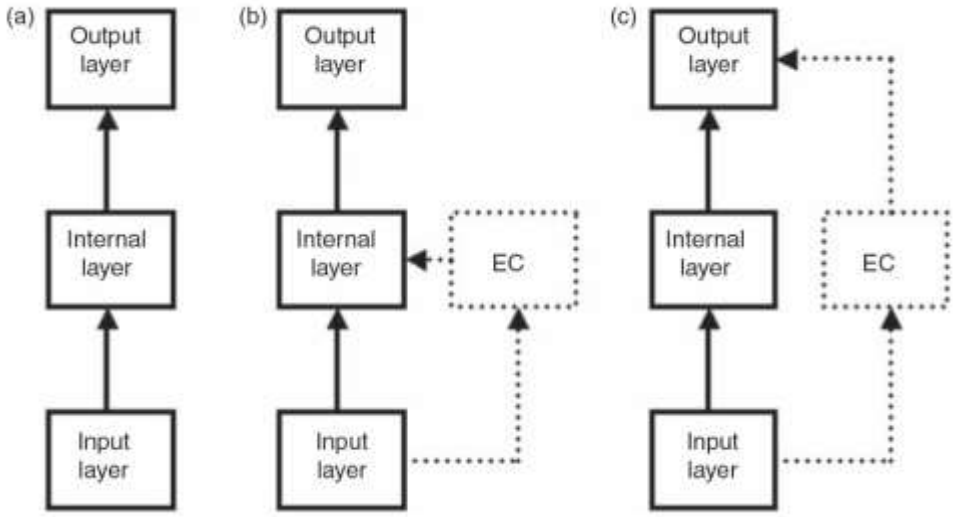


Figure 5.8 – Neural network architecture for robots without an emotional circuit (a) and for robots with an emotional circuit linked to either the internal units (b) or the motor units (c)

The emotional units differ from the internal units for these elements: (i) they have no bias; (ii) they have an activation threshold; (iii) their activation persists in subsequent cycles. Parisi and Petrosino then compared the two type of robots (with and without emotional circuit) to find out which had the highest fitness. They found out that "possessing an emotional circuit lead to more effective behavior and higher level of fitness, and this is true for all five types of robots", precisely, "an architecture with the emotional circuit directly connected to the motor output units turns out to be better for the food/predator, food/mating partner, and food/offspring-care zone robots, while the opposite is true for the food/water and food/pain robots".

Furthermore, the robots with the emotional circuit are able to switch from one activity to another much more quickly than the robots without the emotional circuit.

As we will see in the simulation section, we could take advantage of some concept used in this paper. For instance, we could define the fitness of the robots as the number of laps that they have accomplished in a certain amount of time and comparing it to the average payoff gained by the payoff matrix constructed in the prisoner's dilemma way. This would allow us to have another way of measuring the performances of the different strategy in a more direct way.

6

Agent-Based Simulation vs. Robotic Simulation

6.1 Introduction to the simulation project

This chapter will focus on a new model that has been specifically designed in order to compare two type of simulations: a virtual simulation and a real simulation. In the following paragraph, I will describe how the experiment on the prisoner's dilemma developed and how the simulation is constructed. In the next section, I will introduce the virtual simulation environment that I am going to use to run the experiment on the computer (also with an explanatory example). Then, I will explain the NetLogo program I have created for my thesis project and will introduce the Lego NXT and its programming language. Finally, I will undertake a simulation experiment using both methods and I will confront the results.

6.2 Agent-Based Simulation vs. Robotic Simulation: a Repeated Prisoner's Dilemma Experiment

As specified in the title of this thesis, the aim of this work is to compare a simulation experiment made with NetLogo on the computer and a simulation in the real world through the Lego NXT. I decided to make something that was not extremely complex, in terms of movements and sensors, so that I could replicate it with an agent-based robotic simulation built and programmed in the programming language NXC (Not eXatcly C). Carrying out some researches on game theory and the Prisoner's Dilemma I found several models in NetLogo Model Library. My idea is to revive the famous game of Prisoner's Dilemma so that it can be applied through the use of robots, that on the one hand simplify and assist in explaining an agent-based simulation for non-specialists, while on the other hand introduce a bit of variability and imprecision typical of the world in which we live. The structuring of the experiment I propose evolved in the following way. Everything started from the so-called *chicken game*, which is one of the possible configurations of game theory of non-zero-sum games. Its name refers to the metaphor of the cowardice and generally presents itself as a challenge between two people in front of a risky situation, in which one must prove which of the two players is the most courageous. A usual formulation is as follows: two individuals drive facing each other at high speed. Each must decide at the last minute whether to turn to the right to avoid a collision or not. We have the following cases:

1. neither player turns and you have the collision. This is the worst result and assigns the value 0 to both players;
2. both players turn at the last moment to avoid the collision. It’s a good result, the same for both, although they both lose "prestige"

and no one can be considered the winner. You assign a value of 3 to each player;

3. one of the two turns and the other does not. The first loses much "prestige" and it is assigned a value of 1, while the other is considered the winner and it is assigned 5.

		Player B	
		Cooperate (turn)	Defect (no turn)
Player A	Cooperate (turn)	3, 3	1, 5
	Defect (no turn)	5, 1	0, 0

Table 6.1 – Payoff matrix of the chicken game

The analysis of the situation shows that if both sides are looking for the maximum benefit, both will get the worst result. Turning would seem the best strategy. If they both turn they both get a good result, but neither of them want to turn before the other, since it involves the payment of 1 instead of 5 in favor of the rival. The game can be analyzed from the point of view of cooperation: turning must be interpreted as cooperation and not turning as defection, if both cooperate we get a good overall result. Perhaps the most significant feature is that the game can represent a form of negotiation in which each participant tries to delay the necessary concession to avoid disaster until the last moment, as a means to force the other player to act rationally and be the one who avoids the collision. Both this game and the prisoner's dilemma show the difficulty of finding a solution to this kind of situations in which both confrontation and cooperation are possible. Such situations are at least disturbing because they show the

antagonism that sometimes occurs between immediate individual interests and those of the community.

For this purpose, I decided to experiment the Prisoner's Dilemma, taking inspiration from the game above proposed and I decided to structure the robotic simulation in the following way and then replicated it on NetLogo for a comparison.

Two robots follow a yellow line in the shape of a square, but in which there are alternative ways, as shown in the Figure 6.3 (the blue and the red line). One robot goes in the clockwise direction and one in the counterclockwise direction. By traveling in opposite directions, the two robots will eventually meet and once they come face to face, they will play the Prisoner's Dilemma to decide who will take the short route (the yellow one) and who will have to take the longer way (blue or red). The shape of a square was chosen so that the dilemma can be repeated an infinite number of times, in fact the two robots following the closed geometric shape inevitably will meet each other again and again. The two robots interact through the Prisoner's Dilemma with the following payoff matrix:

		Player B	
		Cooperate (long way)	Defect (short way)
Player A	Cooperate (long way)	4, 4	0, 5
	Defect (short way)	5, 0	2, 2

Table 6.2 – Prisoner's Dilemma matrix used for the simulation

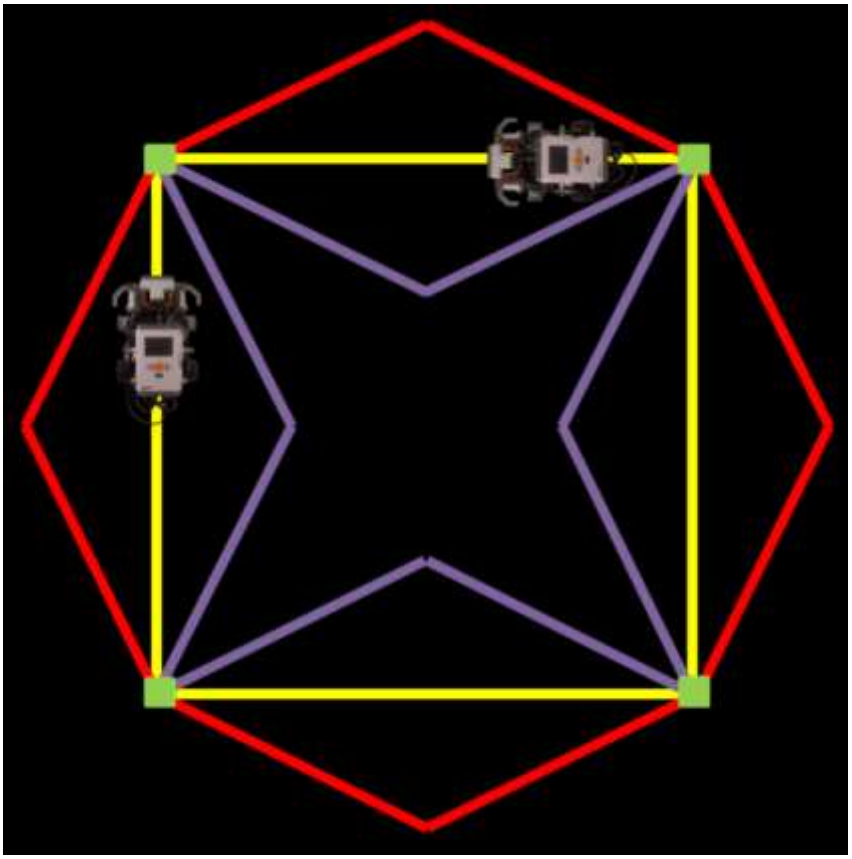


Figure 6.3 – Environment of the simulation

In a game structured in such a way, cooperating means to leave the main street (the yellow one) to your opponent and therefore following a longer route, respectively the blue or the red depending on the robot. Defecting instead mean that you want to go for the shortest route at all costs and therefore continue on the yellow line.

If both robots cooperate, since they cannot take at the same time the short route. The two players decide who take the short way through a "coin flip", that is p = 1/2.

If a robot decides to cooperate while the other decides to defect we will have the following situation: the cooperative robot will be gentle and will leave the short way to the other and it will take the longer street.

For instance, in the case that the blue robot cooperates, he will take the long way, in this case the blue track inside the square.
If both robots decide to defect, neither has the possibility of exploiting the black path, and both will follow their relative colored paths.

		Player B	
		Cooperate (long way)	Defect (short way)
Player A	Cooperate (long way)	The two players decide who will take the short way through a "coin flip", that is p = 1/2.	Player 2 takes the short way, whereas player 1 takes the long way
	Defect (short way)	Player 1 takes the short way, whereas player 2 takes the long way	Both players take the long way

Table 6.4 – Rules of movement that robot will following according to their strategies

As we have seen in the previous chapter when discussing about the experiment of Parisi and Petrosino, one possible way of assessing the performance of a strategy, in addition of calculating the average payoff, would be to count the number of laps of the square completed in a predetermined time.
At the same time, I will develop the same simulation in NetLogo, creating a setting equal to that of the simulation through robots. This will allow us to study the differences and the addition that a real simulation can create.

In the next section, I will introduce the software used for the virtual simulation: NetLogo. I will give an overview of the program and a simulation through an explanatory example.

6.3 Introduction to NetLogo

This section is used to understand the basics of NetLogo. Starting from a simple example will help us to understand the next simulation. We start by seeing what the main features of NetLogo are, how it works and how can be programmed from scratch. I will also explain the meaning of each line of code of the explanatory example. The next section will show how to approach NetLogo, then we will go through the programming language.

NetLogo is a freely downloadable, agent-based software package that was created at the Center for Connected Learning and Computer-Based Modeling at Northwestern University, directed by Uri Wilenksy. It is a multi-agent programming language and modeling environment for the simulation of complex social and natural phenomena. NetLogo is written in Java so that it can run on all major computer platforms. The realization of the agent-based model is done in a specific virtual environment. NetLogo was created through the union of *StarLisp* and *Logo*. From the former, similar to Lisp, it inherited the structure of multiple agents, whereas form the latter stems the possibility of controlling a *turtle.* The biggest lack of Logo was that there was the possibility to create and control only one turtle; in NetLogo instead multiple turtles can be created (Tisue & Wilensky, 2004).

NetLogo users can enter instructions for independent agents that operate in parallel in the model. This makes the analysis of connections between the individual behavior of the agents and the characteristics that emerge from the interaction of populations of individuals possible.

The studies of complex phenomena through NetLogo may affect a wide variety of fields of study, for example, the natural sciences, biology, medicine, mathematics, social sciences, economics and psychology.

6.4 Features and overview of NetLogo

The NetLogo world is populated by agents. All of the agents can interact with each other and perform multiple tasks concurrently. In the model there can be four types of agents (Wilensky, 1999):

1. *Turtles*, they are agents that move around the world. The world is two dimensional and is divided up into a grid of patches;
2. *Patches*, each patch is a square piece of "ground" over which turtles can move;
3. *Links*, they are agents that connect two turtles;
4. *Observer*, he doesn't have a location, thus, you can imagine him as looking out over the world of turtles and patches.

When NetLogo starts up, there are no turtles. The observer can make new turtles. Patches can make new turtles too. (Patches can't move, but otherwise they're just as "alive" as turtles.)

Patches have coordinates. The patch at coordinates (0, 0) is called the origin and the coordinates of the other patches are the horizontal and vertical distances from this one. We call the patch's coordinates pxcor and pycor. Just like in the standard mathematical coordinate plane, pxcor increases as you move to the right and pycor increases as you move up.

The total number of patches is determined by the settings min-pxcor, max-pxcor, min-pycor, and max-pycor When NetLogo starts up, min-pxcor, max-pxcor, min-pycor, and max-pycor are -16, 16, -16, and 16

respectively. This means that *pxcor* and *pycor* both range from -16 to 16, so there are 33 times 33, or 1089 patches total. (You can change the number of patches with the Settings button.)

Turtles have coordinates too: *xcor* and *ycor*. A patch's coordinates are always integers, but a turtle's coordinates can have decimals. This means that a turtle can be positioned at any point within its patch; it doesn't have to be in the center of the patch.

Links do not have coordinates. Every link has two ends, and each end is a turtle. If either turtle dies, the link dies too. A link is represented visually as a line connecting the two turtles.

NetLogo allows you to define different "breeds" of turtles and breeds of links. Once you have defined breeds, you can go on and make the different breeds behave differently.

In NetLogo, commands and reporters tell agents what to do. A command is an action for an agent to carry out, resulting in some effect. A reporter is instructions for computing a value, which the agent then "reports" to whoever asked it. Typically, a command name begins with a verb, such as "create", "die", "jump", "inspect", or "clear". Most reporter names are nouns or noun phrases. Commands and reporters built into NetLogo are called primitives. Commands and reporters you define yourself are called procedures. Each procedure has a name, preceded by the keyword *to* or *to-report*, depending on whether it is a command procedure or a reporter procedure. The keyword *end* marks the end of the commands in the procedure. Once you define a procedure, you can use it elsewhere in your program.

In many NetLogo models, time passes in discrete steps, called "ticks". NetLogo includes a built-in tick counter so you can keep track of how many ticks have passed.

Another feature of NetLogo that is important for my simulation is the ability of turtles to draw. The drawing is a layer where turtles can make visible marks. In the view, the drawing appears on top of the patches but underneath the turtles. Initially, the drawing is empty and transparent. You can see the drawing, but the turtles (and patches) can't. They can't sense the drawing or react to it. The drawing is just for people to look at. Turtles can draw and erase lines in the drawing using the *pen-down* and *pen-erase* commands. When a turtle's pen is down (or erasing), the turtle draws (or erases) a line behind it whenever it moves. The lines are the same color as the turtle. To stop drawing (or erasing), use *pen-up*. Lines drawn by turtles are normally one pixel thick. If you want a different thickness, set the *pen-size* turtle variable to a different number before drawing (or erasing). In new turtles, the variable is set to 1.

The way the world of patches is connected can change. By default the world is a torus which means it isn't bounded, but "wraps". Therefore, when a turtle moves past the edge of the world, it disappears and reappears on the opposite edge and every patch has the same number of "neighbor" patches. If you're a patch on the edge of the world, some of your "neighbors" are on the opposite edge.

However, you can change the wrap settings with the Settings button. If wrapping is not allowed in a given direction then in that direction (x or y) the world is bounded. Patches along that boundary will have fewer than 8 neighbors and turtles will not move beyond the edge of the world.

6.5 An explanatory example: Prisoner's Dilemma N-Player Iterated

In this section I will explain the main features of NetLogo through the *Prisoner's Dilemma N-Player Iterated* from the Models Library. I chose to use this models as an explanatory one because it relatively simple but at the same time it is about a relevant topic for my thesis project. This model is a multiplayer version of the iterated Prisoner's Dilemma. It is intended to explore the strategic implications that emerge when the world consists entirely of Prisoner's Dilemma like interactions. In this model, turtles represent individuals that play a certain strategy:

- blue turtles play a *defect* strategy, that is always defect;
- red turtles play a *cooperate* strategy, that is always cooperate;
- gray turtles play a *random* strategy, that is randomly cooperate or defect;
- lime turtles play the *tit-for-tat* strategy, that is if an opponent cooperates on this interaction cooperate in the next interaction with them. If an opponent defects on this interaction, defect on the next interaction with them. Initially cooperate;
- turquoise turtles play the *unforgiving* strategy, that is cooperate until an opponent defects once, then always defect in each interaction with them.

In order to determine a single "best" strategy, Wilenski created a world with multiple agents playing a variety of strategies in repeated Prisoner's Dilemma situations. The turtles with different strategies wander around randomly until they find another turtle to play with. Note that in this model, each turtle remembers their last interaction

with each other turtle. While some strategies don't make use of this information, other strategies do (tit-for-tat). When two turtles interact, they display their respective payoffs as labels. Each turtle's payoff for each round will determined as follows:

		Partner's Action	
		Cooperate	Defect
Turtle's Action	Cooperate	3, 3	0, 5
	Defect	5, 0	1, 1

Table 6.5 – Payoff matrix of the model Prisoner's Dilemma N-Players Iterated

This way of determining payoff is the opposite of how it is done in the basic Prisoner's Dilemma model. In basic Prisoner's Dilemma, you were awarded something bad with jail time. In this model, something good is awarded with money. As we have seen, the experiment conducted in this thesis will use a matrix similar to this one, in which something good is awarded instead of something bad.

In the bottom-left corner of Figure 6.6, we can see a plot of the average payoff of each strategy in an interaction vs. the number of iterations. This is a good indicator of how well a strategy is doing relative to the maximum possible average of 5 points per interaction.

Now that we have introduced the model we can start to explore the user interface and afterwards we can move on to the programming language.

6.5.1 NetLogo User Interface

Figure 6.6 shows NetLogo's user interface after opening and running the model. Model controls are on the left. On the right is the graphics window, in which the "world" of the model is made visible.

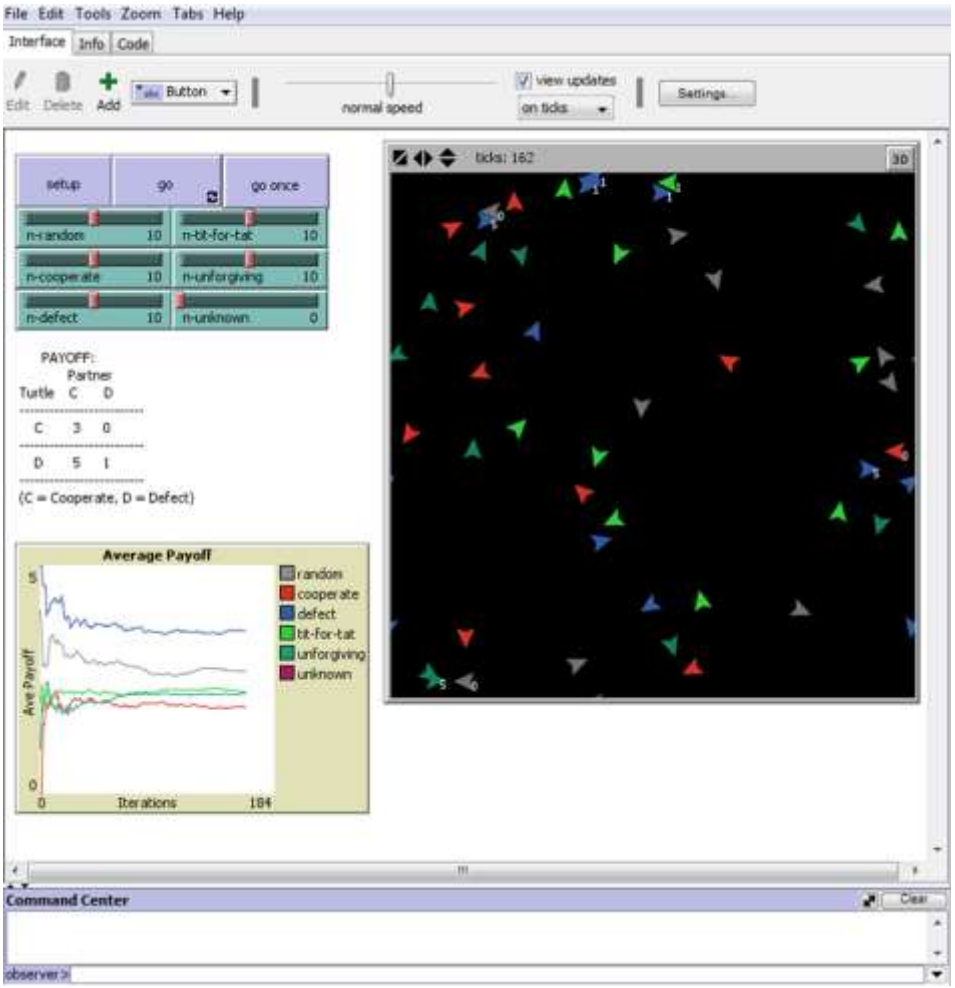


Figure 6.6 – NetLogo's user interface, with model Prisoner's Dilemma N-Players Iterated

In this screen shot, we see only NetLogo's "Interface" tab. The other tabs contain documentation and the actual model code. The "Interface" tab is also an interface builder. No distinction is made between using a model and editing it. You can move, modify, or create interface

elements at any time. Agents can be inspected and altered. The code for the model can be changed without restarting the simulation. Furthermore, the “Info” tab is where the model is usually described in every feature and the “Code” tab is the place where you write the language code.

In the top-left corner of Figure 3, we can see three buttons. Buttons provide an easy way to control the model. Usually a NetLogo model has at a "setup" button, to set up the initial state of the world, and a "go" button to make the model run continuously. Some models will have additional buttons that perform other actions. Every button contains different NetLogo code and that code is run when the button is pushed. In this model, the author has created two different type of “go” buttons. The first type is a “forever-button”, that means that once it is pressed it keeps running the code over and over until it is pressed a second time. The second type of button is a “once-button”, that means that the code is run only once. The “once-button” can be useful to watch the simulation developing one tick at the time and therefore better understanding what is actually happening. I found the “once-button” very useful for debugging purposes as it allows to analyze very slowly the ordering of the procedures and actions. Buttons correspond to a procedure in the “Code” tab, and when a button is pressed it run the corresponded procedures. It is possible to change the label name of the button if the actual name of the procedure is misleading if used in the interface. When you put code in a button, you must also specify which agents you want to run that code. You can choose to have the observer run the code, or all turtles, or all patches, or all links. If you want the code to be run by only some turtles or some patches, you could make an observer button, and then have the observer use the ask command to ask only some of the turtles or patches to do something.

Right below the buttons there are six elements called Sliders. Their structure enables to set a min and a top value and an amount of increment. Sliders are considered by NetLogo as global variables that can be change from both interface and from the program itself. For that reason, they are in the interface where we display the results by modifying the values of some variables and see what happen. In this models the sliders allow the user to modify the initial number of turtles that will play a certain strategy. For instance, the slider *n-cooperate* will determine the number of turtle that will play a cooperative strategy and the slider *n-defect* will determine the number of turtles that will play a strategy of defection. The same is true for the remaining sliders. In this manner, we can create the proportion of players of a certain strategy in the world and therefore create our custom environment for the simulation. Playing with the sliders will allow the user to experiment very different initial states of the “world” and observe the results of running the model with a variety of populations and population sizes.

On the right of Figure 3, we can see the “view”. The "view" in NetLogo lets you see the agents in your model on your computer's screen. As your agents move and change, you see them moving and changing in the view. Of course, you can't really see your agents directly. The view is a picture that NetLogo paints, showing you how your agents look at a particular instant. Once that instant passes and your agents move and change some more, that picture needs to be repainted to reflect the new state of the world. Repainting the picture is called "updating" the view.

Above the “view” there is the speed slider of the model interactions, hence the simulations. The speed of the evolution is set with that slider.

In the bottom of Figure 3 there is a box where commands can be typed in and run as observer or a specific agent.

6.5.2 Programming Language

In order to provide an introduction for a basic understanding of the programming language, in this section we will explain every line of code of the model previously presented. We will attach the NetLogo language in the black box and we will comment on the bottom of every set of procedures.

```
1.

globals [
  num-random
  num-cooperate
  num-defect
  num-tit-for-tat
  num-unforgiving
  num-unknown

  num-random-games
  num-cooperate-games
  num-defect-games
  num-tit-for-tat-games
  num-unforgiving-games
  num-unknown-games

  random-score
  cooperate-score
  defect-score
  tit-for-tat-score
  unforgiving-score
  unknown-score
  ]
```

1. Normally, the first part of the model is devoted to define the all the variables. Agent variables are places to store values (such as numbers) in an agent. An agent variable can be a global variable, a turtle variable, a patch variable, or a link variable. In the lines of code above all the global variables of the model are defined. If a variable is a global variable, there is only one value for the variable, and every agent can access it. We can think of global variables as belonging to the observer. Global variables can be created using the *globals[ ]* keyword at the beginning of the code. The first block of global variables (*num-random, num-cooperate, num-defect,* etc.) is used to store the number of turtles

with each strategy. The second block of global variable (*num-random-games, num-cooperate-games*, etc.) is used to store the number of interactions by each strategy, whereas the last block of global variables is used to store the total score of all turtles playing the same strategy.

```
2.

turtles-own [
  score
  strategy
  defect-now?
  partner-defected?
  partnered?
  partner
  partner-history
]
```

2. Some variables are built into NetLogo. For example, all turtles and links have a *color* variable, and all patches have a *pcolor* variable. The patch variable begins with "p" so it doesn't get confused with the turtle variable, since turtles have direct access to patch variables. If you set the variable, the turtle or patch changes color. Other built-in turtle variables including *xcor*, *ycor*, and *heading*. It is also possible to define new turtle variables using the *turtles-own* keyword. In this case the author created seven turtle variable. The variable *score* is used to store the payoff values gained in each interaction and to create the average payoff plot. *Strategy* is a turtle variable containing the strategy that the turtle plays. *Defect-now?* is a Boolean variable, meaning that it can have two values: true (or 1) or false (or 0). It is used to contain the current decision of each turtle, for instance if you set the variable to *true* it means that the turtle cooperates. *Partner-defected?* contains the current decision of the opponent partner. *Partner?* is used to notify when a turtle successfully grabs a partner (we will see later how). The variable *partner* contains the *who* of the opponent partner (*nobody* if not partnered). *Who* is a built-in turtle variable. It holds the turtle's "who

number" or ID number, an integer greater than or equal to zero. It is not possible to set this variable; a turtle's who number never changes. Who numbers start at 0. A dead turtle's number will not be reassigned to a new turtle until you use the *clear-turtles* or *clear-all* commands, at which time who numbering starts over again at 0. *Partner-history* is a list containing information of past interactions with other turtles indexed by *who* values. Lists let you store multiple pieces of information in a single value by collecting that information in a list. Each value in the list can be any type of value: a number, or a string, an agent or agentset, or even another list.

```
3.

to setup
  clear-all
  store-initial-turtle-counts
  setup-turtles
  reset-ticks
end
```

3. Here we have the *setup* procedure for the homonym button in the NetLogo interface. It is one of the principal procedure allowing the initial setup of our "world". The primitive command c*lear-all* (also *ca*) resets all global variables to zero, and calls clear-ticks, clear-turtles, clear-patches, clear-drawing, clear-all-plots, and clear-output. The *setup* procedure calls the *store-initial-turtle-counts* and *setup-turtles* procedures. I will describe them in the next paragraph. Modeler usually prefers to divide the code in different procedures (that is why I will specify later what they are about) in order to exploit modularity of programming. Then we have *reset-ticks* that resets the tick counter to zero, sets up all plots, then updates all plots (so that the initial state of the world is plotted). Normally *reset-ticks* goes at the end of a setup procedure.

```
4.

to store-initial-turtle-counts
  set num-random n-random
  set num-cooperate n-cooperate
  set num-defect n-defect
  set num-tit-for-tat n-tit-for-tat
  set num-unforgiving n-unforgiving
  set num-unknown n-unknown
end

to setup-turtles
  make-turtles
  setup-common-variables
end
```

4. The procedure *store-initial-turtle-counts* record the number of turtles created for each strategy. The number of turtles of each strategy is used when calculating average payoffs. The number of turtles of each strategy can be modified through the sliders *n-random*, *n-cooperate*, etc. in the user interface. Setup-turtles calls two procedure that we will describe below, but what is a procedure? Commands and reporters you define yourself are called procedures. Each procedure has a name, preceded by the keyword *to* or *to-report*, depending on whether it is a command procedure or a reporter procedure. The keyword *end* marks the end of the commands in the procedure. Once you define a procedure, you can use it elsewhere in your program.

```
5.

to make-turtles
  crt num-random [ set strategy "random" set color gray - 1 ]
  crt num-cooperate [ set strategy "cooperate" set color red ]
  crt num-defect [ set strategy "defect" set color blue ]
  crt num-tit-for-tat [ set strategy "tit-for-tat" set color lime ]
  crt num-unforgiving [ set strategy "unforgiving" set color
turquoise - 1 ]
  crt num-unknown [set strategy "unknown" set color magenta ]
end

to setup-common-variables
  ask turtles [
    set score 0
    set partnered? false
    set partner nobody
    setxy random-xcor random-ycor
  ]
  setup-history-lists
end
```

5. The procedure *make-turtles* creates the appropriate number of turtles playing with each strategy and also assigns different colors for different strategies. C*reate-turtles number [commands]* is used to generate new turtles at the origin. New turtles have random integer heading and the color is randomly chosen. In this case the number of turtles to be created is assigned with the sliders *n-random, n-cooperate,* etc. in the interface which passes a constant that is stored respectively in the variables *num-random*, *num-cooperate*, etc.. The commands are written in the square brackets. In this case, we have two commands. The first one, sets the turtle variable *strategy* to the given value, in this case a string containing the name of the strategy. The second one, assignes the appropriate color the group of turtle with the same strategy. Turtles that use a Random strategy have grey color; turtles that use a cooperative strategy have red color and so on. The variable *color* is built into NetLogo and refers to turtles.

The procedure *setup-common-variables* sets the variables that all turtles share. Furthermore, it also assigns a random position to each turtles through the command *setxy random-xcor random y-cor*. The command *setxy x y* is used to set the turtles coordinates; in this case the author passes a random value for the x-coordinate and a random value for the y-coordinate. This procedure also calls the *setup-history-lists* procedure, which is described below.

```
6.

to setup-history-lists
  let num-turtles count turtles
  let default-history []
  repeat num-turtles [ set default-history (fput false default-
history) ]
  ask turtles [ set partner-history default-history ]
end
```

The procedure setup-history-lists initialize *partner-history* list in all turtles. The command *let* creates a new local variable and gives it the given value, in this case, the total number of turtles. This is achieved through the command *count agentset*, which reports the number of agents in the given agentset (here turtles). The second local variable is initialized to be a list. This is done by adding the square brackets after the variable name. The *repeat* command used in the following line is structured in this way: *repeat number [commands]* and it runs *command* a *number* of times. In this context, it is used to create a list with *num-turtle* elements for storing partner histories. It repeat for *num-turtles* times this command *[set default-history (fput false default-history)]*, which adds item (here *false*) to the beginning of a list *default-history* and reports the new list. Finally, it assigns the *default-history* list to the *partner-history* list for the whole turtle agentset.

```
7.

to go
  clear-last-round
  ask turtles [ partner-up ]
  let partnered-turtles turtles with [ partnered? ]
  ask partnered-turtles [ select-action ]
  ask partnered-turtles [ play-a-round ]
  do-scoring
  tick
end
```

7. Now, we can begin with the runtime procedures. This is the main procedure, it represents the start of the simulation. *Go* defines the code of the homonym button on the interface. When pressing the *go* button the model runs continuously. Usually the *go* procedure is kept clean from code and it just calls the other procedures in the correct order. First, it calls the *clear-last-round* procedure which will be discussed in the next section; then it asks the turtles agentset to run the *partner-up*

procedure which basically have turtles try to find a partner. In the next line the author creates a local variable called partnered-turtles initialized with all the turtles that have a partner. Remember that *with* takes two inputs: on the left, an agentset (usually "turtles" or "patches"). On the right, a boolean reporter. It reports a new agentset containing only those agents that reported true. In other words, the agents satisfying the given condition, in this case the condition is *partnered?* which is the short form of *partnered? = true*. Then the author asks all partnered turtles to *select-action* and to *play-a-round*. The command *tick* advances the tick counter by one and updates all plots. Remember that the tick counter has to be started with *reset-ticks* (we saw this in the *setup* procedure).

```
8.

to clear-last-round
  let partnered-turtles turtles with [ partnered? ]
  ask partnered-turtles [ release-partners ]
end

to release-partners
  set partnered? false
  set partner nobody
  rt 180
  set label ""
end
```

8. The procedure *clear-last-round* use the same method as in the *go* procedure in order to interact just with the partnered turtles to which it asks to *release-partner.* In this last procedure the author sets the Boolean variable *partnered?* to false and the variable *partner* to *nobody* (This is a special value which some primitives such as turtle, one-of, max-one-of, etc. report to indicate that no agent was found). Furthermore, the turtles turn right of 180 degrees to leave. This is accomplished with command *rt* which is the short form of *right*. At the end, the built-in turtle variable *label* has been assigned an empty string.

In this way nothing will appear in the view. As we will see later on, the *label* variable is used to display the payoff gained by the turtles during the Prisoner's Dilemma interaction.

```
9.

to partner-up
  if (not partnered?) [
    rt (random-float 90 - random-float 90) fd 1
    set partner one-of (turtles-at -1 0) with [ not partnered? ]
    if partner != nobody [
      set partnered? true
      set heading 270
      ask partner [
        set partnered? true
        set partner myself
        set heading 90
      ]
    ]
  ]
end
```

9. This procedure is a turtle procedure and it is called in the *go* procedure. The first *if* is used to check that the turtle is still not partnered. The command *not* reports true if the Boolean (here *partnered?*) is false, otherwise reports false. The third line is used to move around the turtles randomly. The operation *(random-float 90 - random-float 90)* allows for a random turn. *Random-float number* works as follows: If *number* is positive, reports a random floating point number greater than or equal to 0 but strictly less than number. If *number* is negative, reports a random floating point number less than or equal to 0, but strictly greater than number. If *number* is zero, the result is always 0. The command *fd 1* moves the turtles one step forward.
A second control is made with *if partner != nobody.* If the condition is true, then set *partnered?* to true and set the *heading* 270, that is look toward west. It also set *partnered?* to true for the partner turtle and set the variable *partner* with the turtle who asked me to do what I'm doing right now.

```
10.

to select-action ;;turtle procedure
  if strategy = "random" [ act-randomly ]
  if strategy = "cooperate" [ cooperate ]
  if strategy = "defect" [ defect ]
  if strategy = "tit-for-tat" [ tit-for-tat ]
  if strategy = "unforgiving" [ unforgiving ]
  if strategy = "unknown" [ unknown ]
end

to play-a-round
  get-payoff     ;;calculate the payoff for this round
  update-history ;;store the results for next time
end
```

10. The procedure *select-action* by using a set of *if* controls choose an action based upon the strategy being played. We will analyze the action corresponding to a certain strategy later in the code. Remember that the strategy is assign in the *make-turtles* procedure. The *play-a-round* procedure calls *get-payoff* and *update-history* procedures that are described below.

```
11.

to get-payoff
  set partner-defected? [defect-now?] of partner
  ifelse partner-defected? [
    ifelse defect-now? [
      set score (score + 1) set label 1
    ] [
      set score (score + 0) set label 0
    ]
  ] [
    ifelse defect-now? [
      set score (score + 5) set label 5
    ] [
      set score (score + 3) set label 3
    ]
  ]
end

to update-history
  if strategy = "random" [ act-randomly-history-update ]
  if strategy = "cooperate" [ cooperate-history-update ]
  if strategy = "defect" [ defect-history-update ]
  if strategy = "tit-for-tat" [ tit-for-tat-history-update ]
  if strategy = "unforgiving" [ unforgiving-history-update ]
  if strategy = "unknown" [ unknown-history-update ]
end
```

11. The *get-payoff* procedure calculate the payoff for the current round. First, it assigns to *partner-defected?* the value of *defect-now?* of the current partner through the command *of.* It works this way: *[reporter] of agent* it gives you the value of the reporter for that agent.
Second, the *ifelse* statement check if *partner-defected? = true,* if it is so it runs the commands in the first set of square bracket, otherwise it runs the second set of square bracket. The same happens more internally but the condition to satisfy is *defect-now? = true*. In other word, the first *ifelse* sorts between the two columns of Figure 3 (page …), whereas the more internal *ifelse* sorts between the two rows. These two crossroads allow turtles to reach the right payoff according to their own decision and the partner's decision. Furthermore, they display the relative payoff in view through the mean of the *label* command. The *update-history* procedure updates the partner-history based upon the strategy being played. As we will see, in this model only one strategy uses the partner-history, the tit-for-tat strategy.

```
12.

to act-randomly
  set num-random-games num-random-games + 1
  ifelse (random-float 1.0 < 0.5) [
    set defect-now? false
  ] [
    set defect-now? true
  ]
end

to act-randomly-history-update
end
```

12. *Act-randomly* defines the action given by the Random strategy. First, it updates a counter to keep track of how many interactions turtles using this strategy occur. Then it sets the *defect-now?* variable with probability 1/2 to true or false. This is achieved through the use of an *ifelse* statement in which the condition to be satisfied is the generation

of a random number between 0 and 0.999 that has to be lower than 0.5. This "trick" allow chose with p=1/2 both decisions. The *act-randomly-history-update* is an empty procedure since this strategy does not need the history of interactions.

```
13.

to cooperate
  set num-cooperate-games num-cooperate-games + 1
  set defect-now? false
end

to cooperate-history-update
end
```

13. The *cooperate* procedures defines the action given by the Cooperate strategy. It is really simple, it just sets the *defect-now?* variable to false. As in the previous procedure, there is a counter (*num-cooperate-games*) that keeps track of the total number of interactions occurred. Again, the history of interaction is not needed for this strategy.

```
14.

to defect
  set num-defect-games num-defect-games + 1
  set defect-now? true
end

to defect-history-update
end
```

14. The *defect* procedures defines the action given by the Defect strategy. It is really simple, it just sets the defect-now? variable to true. As in the previous two procedures, there is a counter (*num-defect-games*) that keeps track of the total number of interactions. Yet again, the history of interaction is not needed for this strategy.

```
15.

to tit-for-tat
  set num-tit-for-tat-games num-tit-for-tat-games + 1
  set partner-defected? item ([who] of partner) partner-history
  ifelse (partner-defected?) [
    set defect-now? true
  ] [
    set defect-now? false
  ]
end

to tit-for-tat-history-update
  set partner-history (replace-item ([who] of partner) partner-
history partner-defected?)
end
```

15. The *tit-for-tat* procedures defines the action given by the Tit-for-Tat strategy. This is a more complicated strategy since it makes use of the past information. As the other strategies, it starts with a counter (*num-tit-for-tat-games*) of the total number of interactions occurred. The command *item index list* reports the value of the item in the given list with the given index. In this case, the list is *partner-history*, whereas the index is given by the ID (*who*) of the opponent player (*partner*). Then, it this value to *partner-defected?*. Subsequently, it checks if the previous move of the opponent was a defect or cooperate and sets *defect-now?* accordingly.

As we have already said, this strategy needs past information to be known and updated. This is done in the *tit-for-tat-history-update* procedure. The update of the history is done through the command *replace-item index list value* that replaces an item in that list. *Index* is the index of the item to be replaced, starting with 0. Note that *replace-item* is used in conjunction with *set* to change a list. In this case, the index value is the ID of the partner turtle, the list in which to replace is *partner-history* and the value will be a true or a false contained in the *partner-defected?* variable. Here the variable to be updated is *partner-history.*

```
16.

to do-scoring
  set random-score      (calc-score "random" num-random)
  set cooperate-score   (calc-score "cooperate" num-cooperate)
  set defect-score      (calc-score "defect" num-defect)
  set tit-for-tat-score (calc-score "tit-for-tat" num-tit-for-tat)
  set unforgiving-score (calc-score "unforgiving" num-unforgiving)
  set unknown-score     (calc-score "unknown" num-unknown)
end

to-report calc-score [strategy-type num-with-strategy]
  ifelse num-with-strategy > 0 [
    report (sum [ score ] of (turtles with [ strategy = strategy-
type ]))
  ] [
    report 0
  ]
End
```

16. What the *do-scoring* procedure does is to calculate the total scores of each strategy. It uses a reporter procedure (*calc-score)* which takes two inputs (*strategy-type num-with-strategy*) and it is defined below. Remember that a reporter procedure uses *report* to report a value for the procedure. First, it checks that the number of turtles playing a strategy is greater than zero. If it is so, it reports a value equal to the total score for a strategy (if any turtles exist that are playing it). Otherwise, it reports 0. The total scores are then stored in the *random-score, cooperate-score, defect-score,* etc. that are going to be used to create the plot.

The model presented in this section was a trivial example to better understand the NetLogo structure and its power. Now that we have introduced NetLogo, we can move to the code of the simulation I created for this thesis project. In the next section, I will describe and analyze the code I have written for this experiment.

6.6 NetLogo code of the virtual simulation

In this part I will provide a detailed description of the code of the program I developed for this thesis project.

6.6.1 The environment

This first section is entirely devoted to the description of the NetLogo code relative to the environment in which the virtual simulation takes place. Even though the first lines of code are not about the creation of the environment, I decided to start with this part because I believe that for this simulation is very important to understand how the *view* is built. Before starting to describe the actual code, I would like to begin the description with the help of Figure 6.7. Looking at this picture, it is possible to understand at first glance how the *view* of the model is created. It consists of seven layers, each created through a dedicated procedure. These procedures are called in the following order in the *setup* procedure:

1. *black-background*
2. *make-corners*
3. *make-red-path*
4. *make-blue-path*
5. *make-yellow-path*
6. *make-fake-green-corner*

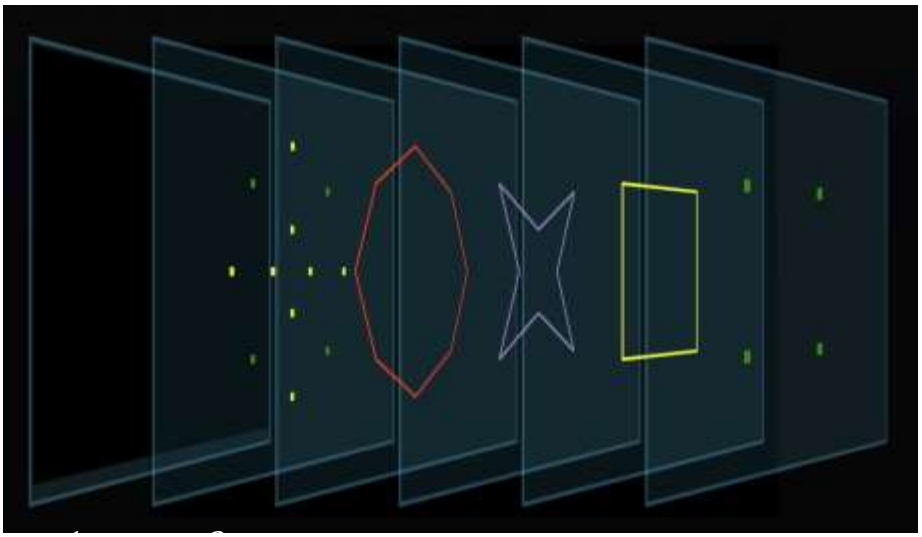


Figure 6.7 – Representation of the layers that create the environment

All the graphic elements will be visible at the end of the setup procedure, except layer 2. This layer is created with the purpose of signaling all the corners of the paths to the turtles and it is just composed by colored patches. In other words, when a turtle steps on a yellow or green patch it will know where it is in the environment and will make movements accordingly. We will talk about this in more detail in the *move-static* and *move-dynamic* procedures. At the end of the *setup* the *view* will look as Figure 6.8.

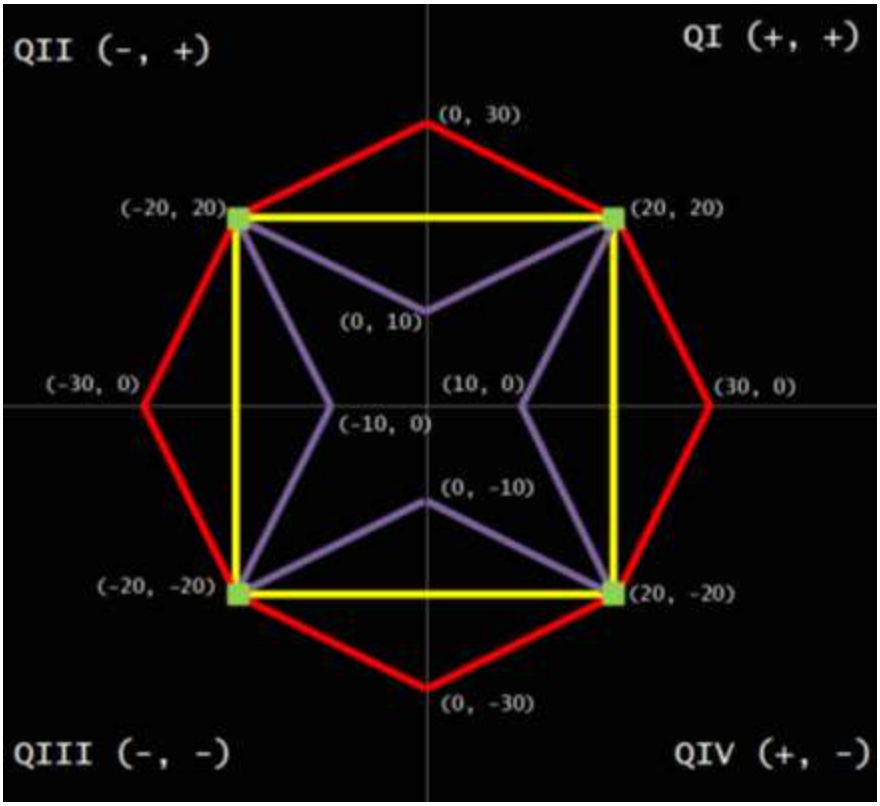


Figure 6.8 – NetLogo view after setup

Furthermore, in the above picture I have also added some coordinates so as to give some reference point. This will be useful for understanding all the procedures that will make use of spatial coordinates. Further

features of the environment will be given in the code descriptions. For instance, we will see how the length of the long way has been chosen.

6.6.2 The main procedures

Now let's analyze the actual NetLogo code. We will follow the order of the procedures as specified in the *setup*. As we previously did for the explanatory example, I will attach the NetLogo language in the black box and I will comment on the bottom of every set of procedures.

```
1.

to black-background
  ask patches [set pcolor black]
end
```

1. The procedure *black-background* is straightforward. It is just needed to create the black background. This is accomplished by asking the patch agentset to turn its color to black. Remember that *pcolor* is a built-in patch variable that holds the color of the patch. In case you want to change color to a turtle you just need to use *color*. As we will see later, the black colored background is not by chance. It was chosen in order to improve the performance of the light sensor of the robots in real world simulation. I will discuss this in more detail in the robotic experiment section.

```
2.

to make-corners
  ask patch -20 -20 [set pcolor green]
  ask patch  20  20 [set pcolor green]
  ask patch -20  20 [set pcolor green]
  ask patch  20 -20 [set pcolor green]

  ask patch   0  30 [set pcolor yellow]
  ask patch  30   0 [set pcolor yellow]
  ask patch   0 -30 [set pcolor yellow]
  ask patch -30   0 [set pcolor yellow]

  ask patch   0  10 [set pcolor yellow]
  ask patch  10   0 [set pcolor yellow]
```

```
  ask patch   0 -10 [set pcolor yellow]
  ask patch -10   0 [set pcolor yellow]
end
```

2. The procedure *make-corners* is used to create all the patches needed for signaling purposes in this models. We have a total of 12 colored patches, but none of these is visible in the view. The green patches are place under the corners of the yellow square, the main path. They have three purposes: (i) when the turtles are following the yellow path, they signal the corner so that the turtles know when to turn right/left; (ii) when a turtle has to follow the long way and thus goes back in order to take the long way, the green patch is used a signal to set the *long-way?* variable to *true*; (iii) when a turtles comes from the long way, the green patch signals the entrance to the normal path (yellow line) and the *long-way?* variable to *false*.

The yellow patches are placed under the top-corner of each triangle that constitutes the long way of the path (blue and red line). We will see their use in the *move* procedures, but basically they are used to signal the turtles when to switch the variable *first-half* to *false*, and thus make the correct movements accordingly.

```
3.

to make-red-path
  crt 1
  ask turtles [
    set color red
    set size 1
    setxy -20 20
    pen-down
    set pen-size 5
    facexy 0 30
    fd 22.36067977
    facexy 20 20
    fd 22.36067977
    facexy 30 0
    fd 22.36067977
    facexy 20 -20
    fd 22.360679771
    facexy 0 -30
    fd 22.36067977
    facexy -20 -20
    fd 22.36067977
```

```
    facexy -30 0
    fd 22.36067977
    facexy -20 20
    fd 22.36067977
    die
  ]
end
```

3. The procedure *make-red-path* contains the instructions to draw the red path. I used the word “draw” because it is what actually happen. I created a turtle and I set its pen down (*pen-down*). When a turtle's pen is down, all movement commands cause lines to be drawn, including jump, setxy, and move-to. I also could have done the same with the following line of code: *set pen-mode “down”*. What instead is not straightforward, is the reason behind the length size of the first half and the second half of the long way, for instance the segment between the coordinates (-20, 20) and (0, 30). I will try to explain this problem with the help of Figure 6.9 that is a detailed representation of half long way.

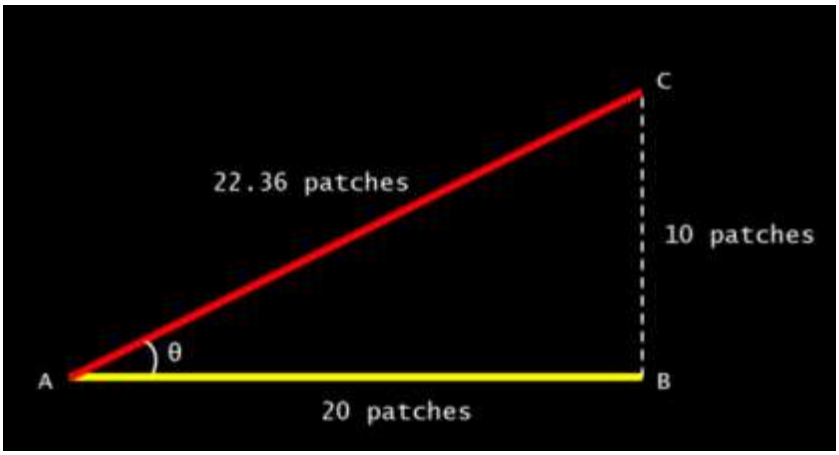


Figure 6.9 – Detailed representation of half long way

The length size of the catheti is not random. In fact, the size of BC is exactly half the size of AB. This particular proportion has been chosen in order to avoid numbers whose decimal part is too long to be

represented when the computer calculates the *xcor* and *ycor* of the turtles. This problem arises from the fact that not every floating point number can be represented in the binary form. This is the reason why, when the turtle is following the long way from the coordinates (-20, 20) and (0, 30) it will update its *xcor* and *ycor* by adding 1 to the *xcor* and 0.5 to the *ycor*. I chose 0.5, because this number can be represented in a binary form and therefore have no approximation errors when computed by the CPU. For instance, if you convert the number 5.092860082799219 (this is the kind of numbers you get from NetLogo *xcor* and *ycor* when a turtle is moving in a random way) to the binary form (IEEE 754 floating point is the standard used by all modern CPUs) the most accurate representation would be 5.09286022186279296875, which is not exactly the same (after the sixth decimal digit the number is different). Instead, when you convert 0.5 into its binary form the most accurate representation would be 0.5, which is exactly the same (all computations above have been made through the website www.binaryconvert.com/convert_float.html). This allow us to avoid imprecisions in the turtles movements that will lead them to miss the yellow patch located at the top corner of the long way.

```
4.

to make-blue-path
  crt 1
  ask turtles [
    set color blue
    set size 1
    setxy -20 20
    pen-down
    set pen-size 5
    facexy 0 10
    fd 22.36067977
    facexy 20 20
    fd 22.36067977
    facexy 10 0
    fd 22.36067977
    facexy 20 -20
    fd 22.360679771
    facexy 0 -10
    fd 22.36067977
    facexy -20 -20
```

```
    fd 22.36067977
    facexy -10 0
    fd 22.36067977
    facexy -20 20
    fd 22.36067977
    die
  ]
end
```

4. The *make-blue-path* procedure is structured in the exact same way of the *make-red-path* procedure but instead of drawing outside the yellow square, it draws inside. The same reasoning for the length of the long way can be applied here.

```
5.

to make-yellow-path
  crt 1
  ask turtles [
    set color yellow
    set size 1
    setxy -20 20
    pen-down
    set pen-size 5
    facexy 20 20
    fd 40
    facexy 20 -20
    fd 40
    facexy -20 -20
    fd 40
    facexy -20 20
    fd 40
    die
   ]
end
```

5. The procedure *make-yellow-path* is used to create the main way, the yellow one. This is the way that the robots normally follow until they meet each other and it is also the way that they prefer to travel because it allows to conclude a lap of the circuit more rapidly (thus gaining higher payoffs). As we will see in the *move* procedures, turtles don't actually follow lines even though when running the simulation it looks so.

```
6.

to make-fake-green-corner
  set-default-shape corners "square"
  create-corners 1[
  set xcor -20
  set ycor -20
  set color green
  set size 2
  ]
  create-corners 1 [
  set xcor 20
  set ycor 20
  set color green
  set size 2
  ]
  create-corners 1 [
  set xcor -20
  set ycor 20
  set color green
  set size 2
  ]
  create-corners 1 [
  set xcor 20
  set ycor -20
  set color green
  set size 2
  ]
end
```

6. The procedure *make-fake-green-corner* is self-explanatory. It creates green squares on the corners of the yellow rectangle. They are "fake" since they are used to represent the underlying green patches that are not visible because they are under the turtle's drawing. Notice the corners are made with turtles that take the shape of a square. If they are "fake" why did I make them? I made them for two reasons. Firstly, because I wanted to create an environment identical to the real setup of the robotic experiment. Secondly, I created them because we have in reality green patches beneath them, but they are just not visible because covered by the drawing.

Now that we have introduced the environment of the simulation we can start describing the lines of code of the rest of the program.

```
7.

globals [
   move-rule
   long-way-s?
   long-way-d?
```

```
  result-static
  result-dynamic
  c-static-payoff
  d-static-payoff
  tft-static-payoff
  r-static-payoff
  wsls-static-payoff
  c-dynamic-payoff
  d-dynamic-payoff
  tft-dynamic-payoff
  r-dynamic-payoff
  wsls-dynamic-payoff
  num-static-cooperate-games
  num-static-defect-games
  num-static-tft-games
  num-static-random-games
  num-static-wsls-games
  num-dynamic-cooperate-games
  num-dynamic-defect-games
  num-dynamic-tft-games
  num-dynamic-random-games
  num-dynamic-wsls-games
  first-move-rule?
  static-previous-round
  dynamic-previous-round
  static-payoff
  dynamic-payoff
  num-games
  ]
```

7. As we know the first part of the model is devoted to define the all the variables. In *globals[ ]* all the global variables are defined. Here, I will discuss the use of each of them. Starting from the beginning, we define:

- *move-rule*, this variable is used to notify the turtles which action to make in a way that they respect the outcome of the last round played. *Move-rule* will take the following values:
    - 0, when both turtles have to go forward, that is when they have to follow the yellow line because they have not met yet;
    - 1, when the turtle that travel in a clockwise direction (also called "static") will have to go back and take the long way (in sign of cooperation);
    - 2, when the turtle that travels in a counterclockwise direction (also called "dynamic") will have to go back and take the long way (in sign of cooperation);

    - 3, when both turtles have to take their respective long way. This situation arises from a mutual defection.
- *long-way-s?* and *long-way-d?*; these two variables work in the same way but one is for the *static* turtle and one is for the *dynamic* turtle. They are boolean variables and they will store a *true* (meaning "take the long way") or a *false* (meaning "don't take the long way") that will be used for two purposes. The first one is to signal the turtle when to follow the long path. The second one is to check that the turtles do not play a round of the game while they are in radius but on different path.
- *result-static* and *result-dynamic*; these two variables will store the payoffs relative for the *static* and for the *dynamic* player respectively.
- *c-"player"-payoff, d-"player"-payoff, tft-"player"-payoff, r-"player"-payoff, wsls-"player"-payoff*; these variables are used to store the payoffs relative to the correspondent strategy of each player. In this case the letter at the beginning of the variable is just the first letter of the name of the strategy.
- *num-"player"-cooperate-games, num-"player"-defect-games, num-"player"-tft-games, num-"player"-random-games, num-"player"-wsls-games*; these variables are used as counters for the number of interaction of the correspondent strategy of each player.
- *first-move-rule?*; this is a Boolean variable. As we will see, it is used to prevent that the first turtle that comes back modifies *move-rule* to 0 and doing so not allowing the second turtle to see the previous *move-rule* (was 3) and thus not allowing it to go back to the green patch. This will be explained in more detail in the description of the *move* procedure.

- *static-previous-round* and *dynamic-previous-round*; these two variables are used to store the previous move for each player. As we will see, not all strategy will make use of this information.
- *static-payoff* and *dynamic-payoff*; these two variables are used to store the total payoffs gained by each player. The total is updated every interaction.
- *num-games*; this variable is used as a counter of the total number of interaction that occur in the simulation.

```
8.

breed [statics static]
breed [dynamics dynamic]
breed [corners corner]

statics-own [
  quadrant
  first-half?
  ]

dynamics-own [
  quadrant
  first-half?
  ]
```

8. The breeds, in other words three types of agent having different features intrinsically. The breed *statics* is used to define the behavior of the robot, that in our simulation experiments will not change the strategy every interaction. That is why I called it "static". The breed *dynamics* is used to define the behavior of the robot that in our simulation experiment is going to change the strategy every interaction. That is why I called it "dynamic". The breed *corners* will be just used to create the green square shaped turtles that will represent the corners in the *view*.

The turtle variables *quadrant* and *first-half?* defined for both breeds will have the following scope. The first one, is used to identify the quadrant (I, II, III, IV) in which he *static/dynamic* player is located. The second

one, is used to signal whether the turtle is in first of second half of the long way (the red or blue path).

```
9.

to setup
  clear-all
  black-background
  make-corners
  make-red-path
  make-blue-path
  make-yellow-path
  make-fake-green-corner
  statics-setup
  dynamics-setup
  set move-rule 0
  set long-way-s? false
  set long-way-d? false
  set first-move-rule? true
  set c-static-payoff 0
  set d-static-payoff 0
  set tft-static-payoff 0
  set r-static-payoff 0
  set wsls-static-payoff 0
  set c-dynamic-payoff 0
  set d-dynamic-payoff 0
  set tft-dynamic-payoff 0
  set r-dynamic-payoff 0
  set wsls-dynamic-payoff 0
  set num-static-cooperate-games 0
  set num-static-defect-games 0
  set num-static-tft-games 0
  set num-static-random-games 0
  set num-static-wsls-games 0
  set num-dynamic-cooperate-games 0
  set num-dynamic-defect-games 0
  set num-dynamic-tft-games 0
  set num-dynamic-random-games 0
  set num-dynamic-wsls-games 0
  set static-previous-round 0
  set dynamic-previous-round 0
  set static-payoff 0
  set dynamic-payoff 0
  set num-games 0
  check-sum
  reset-ticks
end
```

9. The procedure setup will create the environment, will set out the agent by calling the procedures *static-setup* and *dynamic-setup* and will initialize all the variables. Furthermore, it will call the procedure *check-sum* which I will describe in detail later. What it basically does is to check that the values of the sliders which defines the probability of using a certain strategy sum up to 100 before starting the simulation.

```
10.

to default-setup
  clear-output
  set static-cooperate 20
  set static-defect 20
  set static-random 20
  set static-tit-for-tat 20
  set static-win-stay-lose-shift 20

  set lock-static-cooperate? false
  set lock-static-defect? false
  set lock-static-random? false
  set lock-static-tit-for-tat? false
  set lock-static-win-stay-lose-shift? false

  set wsls-slider-static 50

  set dynamic-cooperate 20
  set dynamic-defect 20
  set dynamic-random 20
  set dynamic-tit-for-tat 20
  set dynamic-win-stay-lose-shift 20

  set lock-dynamic-cooperate? false
  set lock-dynamic-defect? false
  set lock-dynamic-random? false
  set lock-dynamic-tit-for-tat? false
  set lock-dynamic-win-stay-lose-shift? false

  set wsls-slider-dynamic 50
end
```

10. Here we have the *default-setup* procedure for the homonym button in the NetLogo interface. Pressing the button will allow the user to set the values of the slider of both turtles to their default settings. This is useful to quickly realign all the sliders before setting up a new scenario. In Figure 6.10, a screenshot of the sliders is shown. What we see, is that each slider is paired with a switch called *lock-static-"strategy"* for the first set of sliders and *lock-dynamic-"strategy"* for the second set of sliders. The combination of each slider and its switch will be used when running the code behind the button *setup-sliders*, that once pressed will readjust all the sliders whose switch is not locked in accordance with the values of the locked sliders. Furthermore, the slider *wsls-slider-"breed"* can be used to setup a probability of cooperation after a defect/defect for the Win Stay Lose Shift strategy.

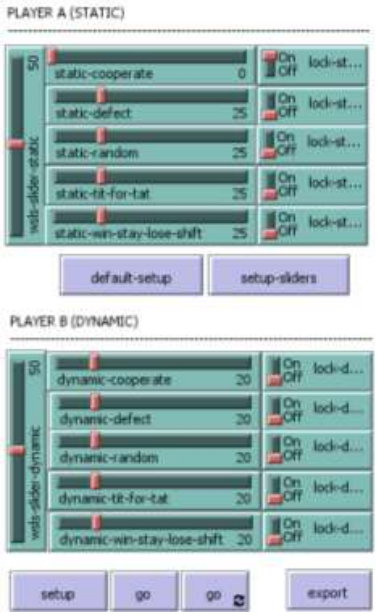


Figure 6.10 – Screenshot of the sliders of the NetLogo model

```
11.

to setup-sliders
let sum-locked 0
  let to-distribute 0
  let num-sliders-free 0
  let final-number 0

  if lock-static-cooperate? = true [ set sum-locked sum-locked +
static-cooperate ]
  if lock-static-defect? = true [ set sum-locked sum-locked +
static-defect ]
  if lock-static-random? = true [ set sum-locked sum-locked +
static-random ]
  if lock-static-tit-for-tat? = true [ set sum-locked sum-locked +
static-tit-for-tat ]
  if lock-static-win-stay-lose-shift? = true [ set sum-locked sum-
locked + static-win-stay-lose-shift ]
free)

  set to-distribute (100 - sum-locked)

  if lock-static-cooperate? = false [ set num-sliders-free num-
sliders-free + 1  ]
  if lock-static-defect? = false [ set num-sliders-free num-
sliders-free + 1 ]
  if lock-static-random? = false [ set num-sliders-free num-
sliders-free + 1 ]
  if lock-static-tit-for-tat? = false [ set num-sliders-free num-
sliders-free + 1 ]
  if lock-static-win-stay-lose-shift? = false [set num-sliders-free
num-sliders-free + 1  ]

  set final-number to-distribute / num-sliders-free
  if final-number < 0 [ set final-number 0 ]
```

```
  if lock-static-cooperate? = false [ set static-cooperate final-
number ]
  if lock-static-defect? = false [ set static-defect final-number ]
  if lock-static-random? = false [ set static-random final-number ]
  if lock-static-tit-for-tat? = false [ set static-tit-for-tat
final-number ]
  if lock-static-win-stay-lose-shift? = false [ set static-win-
stay-lose-shift final-number ]
end
```

11. Here we have the *setup-sliders* procedure for the homonym button in the NetLogo interface. This button is used to help the user to rearrange the values of the sliders with respect to the ones that are locked. For instance, what is going to happen when I press the *setup-sliders* button, if the slider *static-cooperate* is set to 60 (meaning that the first player will use a cooperative strategy 60% of the times) and the lock switch is *on*? What the procedure does in first place, is to count how many points have to be distributed and it does this with the first block of *if*. It is going to store in the local variable *sum-locked* the total number of points of the sliders with the lock switch *on* (true). In second place, it will count how many points have to be redistributed to the sliders with the lock switch *off* (false). Then it will count how many sliders set on *off* there are. In this example, the local variable *to-distribute* would contain 40 (that is 100 – 60). In third place, it will count how many points have to be redistributed for the sliders for which the lock is *off*. It accomplish this task by counting how many sliders with the lock *off* and then it divide *to-distribute* by this number and place this value in the local variable *final-number*. In our example, it would be 40/4, that is 10. Finally, it sets all the sliders with lock *off* to the value stored in *final-number*. The same lines of code are repeated for the block of sliders of the other player, the dynamic. These lines of code are not included in the box above, but they will be available in Appendix A.

```
12.


to check-sum
  ifelse ((static-cooperate + static-defect + static-random +
static-tit-for-tat + static-win-stay-lose-shift) = 100) and
((dynamic-cooperate + dynamic-defect + dynamic-random + dynamic-
tit-for-tat + dynamic-win-stay-lose-shift) = 100) [
    output-print "You can start the simulation"
    ][
    output-print "Error: the sum is not 100\nSimulation will be
blocked\nCheck your sliders and start again"
    ]
end
```

12. The procedure *check-su*m is called in the *setup* procedure. It just checks that the sliders of both players have reasonable values. The sum of each block of sliders must be 100 since they represent the probability of strategy to be played by the player. If both players' sliders have sum 100 it will print "You can start the simulation" in the output area, otherwise it will print "Error: the sum is not 100. Simulation will be blocked. Check your sliders and start again". *n* is used to create a new line inside a string.

```
13.
to statics-setup
  set-default-shape statics "nxt"
  create-statics 1
  ask statics [
    set first-half? true
    set size 7
    set color red
    ifelse random 2 = 0 [
      ifelse random 2 = 0 [set xcor 20] [set xcor -20]
      set ycor (random 38 - 19)
      ][
      ifelse random 2 = 0 [set ycor 20] [set ycor -20]
      set xcor (random 38 - 19)
      ]
    ifelse (ycor < 20) and (ycor > -20) [
      ifelse xcor > 0  [
        facexy 20 -20
        ][
        facexy -20 20
        ]
      ][
      ifelse ycor = 20 [
        facexy 20 20
        ][
        facexy -20 -20
        ]
      ]
  ]
end
```

13. The procedure *static-setup* is called in the *setup* procedure and it initializes the turtle variables and place the turtle with a random position on the yellow line, always with the right heading (the *static* turtle travels in a clockwise direction). Furthermore, this procedure applies the shape *nxt* to the turtle. I created this shape so that the turtle resembles the real robots. In Figure 6.11, it is possible to confront the NetLogo shape and the real Lego NXT Robot side by side from the same perspective. The NetLogo shape is stylized but it still keeps the main characteristics of the real robot: four wheels, the front bumper, the ultrasonic sensor, the NXT brick (the small Lego computer) with the display.

There is also a *dynamic-setup* procedure which is also called in the *setup* procedure. I did not put the code here because it is the same as for the *static*, the only differences are the color (blue, as the path it has to follow) and the orientation (it will have to travel in a counterclockwise direction).

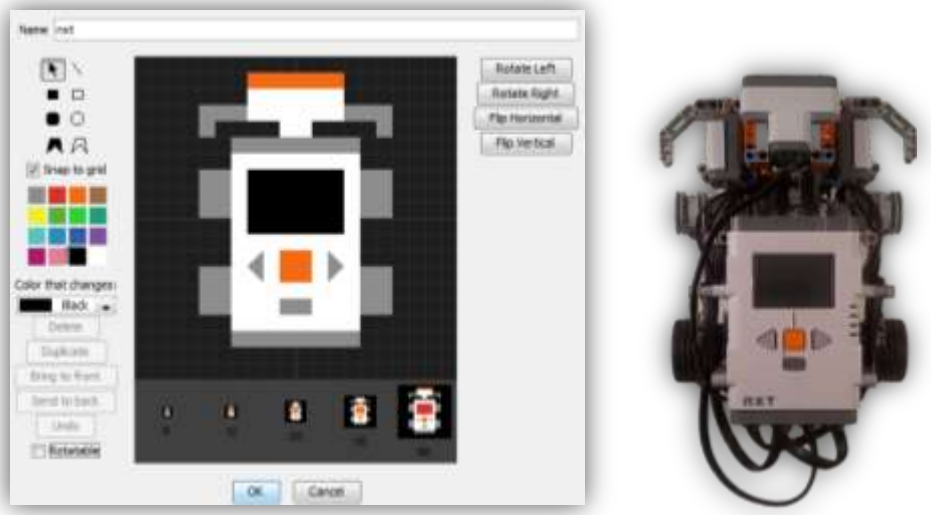


Figure 6.11 – Comparison between the turtle shape in NetLogo and the real Lego NXT robot

```
14.

to go
  move-statics
  move-dynamics
  tick
end
```

14. Here we have the *go* procedure for the homonym button in the NetLogo interface. This is the main procedure of the program. It calls two procedures: *move-statics* and *move-dynamics*. Then it advances the tick counter by one and updates all plots.

```
15.


to move-s [new-x new-y]
  ask statics [
    set xcor (xcor + new-x)
    set ycor (ycor + new-y)
  ]
end
```

15. *move-s* is command procedure that takes two input. It is used to set the *xcor* and the *ycor* (when the static turtle enters the long way) by updating the old values. This is done to avoid approximation errors in the value of the coordinates when taking the oblique paths. We also have a *move-d* procedure which is used by the dynamic turtle.

```
16.

to move-dynamics
    ask dynamics [
      if ((count statics in-radius 1 = 1) and (long-way-d? = false)
and (long-way-s? = false)) [play-a-round 1]
      if long-way-d? = false [
        ifelse move-rule != 1 [
        ifelse move-rule = 2 or move-rule = 3
          while [pcolor != green] [fd -1]
          set long-way-d? true
           ifelse move-rule = 3 [
            ifelse first-move-rule? = true [
              set first-move-rule? false
              ][
              set move-rule 0
              set first-move-rule? true
              ]
          ][
          set move-rule 0
```

```
          ]
        ][
        fd 1
        ]
      ][
      fd 0
      ]
      ]
      if xcor > 0 and ycor >= 0 [set quadrant 1]
      if xcor <= 0 and ycor > 0 [set quadrant 2]
      if xcor < 0 and ycor <= 0 [set quadrant 3]
      if xcor >= 0 and ycor < 0 [set quadrant 4]
     if long-way-d? = true [
      ifelse first-half? = true [
      if quadrant = 1 [
        facexy   0 10
        move-d -1 -0.5
        ]
      if quadrant = 2 [
        facexy  -10 0
        move-d 0.5 -1
        ]
      if quadrant = 3 [
        facexy   0   -10
        move-d 1 0.5
        ]
      if quadrant = 4 [
        facexy   10    0
        move-d -0.5 1
        ]
      if pcolor = yellow [
        set first-half? false
        ]
      ] [
      if quadrant = 1 [
        facexy 20 20
        move-d 0.5 1
        ]
      if quadrant = 2 [
        facexy -20 20
        move-d -1 0.5
        ]
      if quadrant = 3 [
        facexy -20 -20
        move-d -0.5 -1
        ]
      if quadrant = 4 [
        facexy 20 -20
        move-d 1 -0.5
        ]
      ]
    ]
    if (pcolor = green)  [
      ifelse long-way-d? = false [lt 90] [
        if quadrant = 1 [facexy -20  20]
        if quadrant = 2 [facexy -20 -20]
        if quadrant = 3 [facexy  20 -20]
        if quadrant = 4 [facexy  20  20]
      ]
      set first-half? true
      set long-way-d? false
      ]
    ]
end
```

16. The procedure *move-dynamic* is one of the most important procedures in the program. It regulates all the movements that the

turtles make, but also calls the procedure *play-a-round* when the two turtles are *in-radius* 1 (meaning that they are within 1 patch of distance from one another). This is not the only condition that has to be satisfied before the *play-a-round* procedure is called: we also have to make sure that none of the turtles is taking the long way and therefore both *long-way-d* and *long-way-s* have to be *false*. In the following piece of code all the movements are defined in accordance to the variable *move-rule* whose value is set in the *play-a-round*.

Move-rule can take 4 values which are associated with movements of both turtles (see Figure 6.12). If *move-rule = 0,* it means that both turtles have to go forward; if *move-rule = 1,* it means that the *static* turtle has to go back and take the long way; if *move-rule = 2,* it means that the *dynamic* turtle has to go back and take the long way; if *move-rule = 3,* it means that both turtles have to go back and take the long way. The movement in the long way is done using the procedure *move-d* which updates the *xcor* and the *ycor* of the turtle in accordance to the quadrant in which it is located.

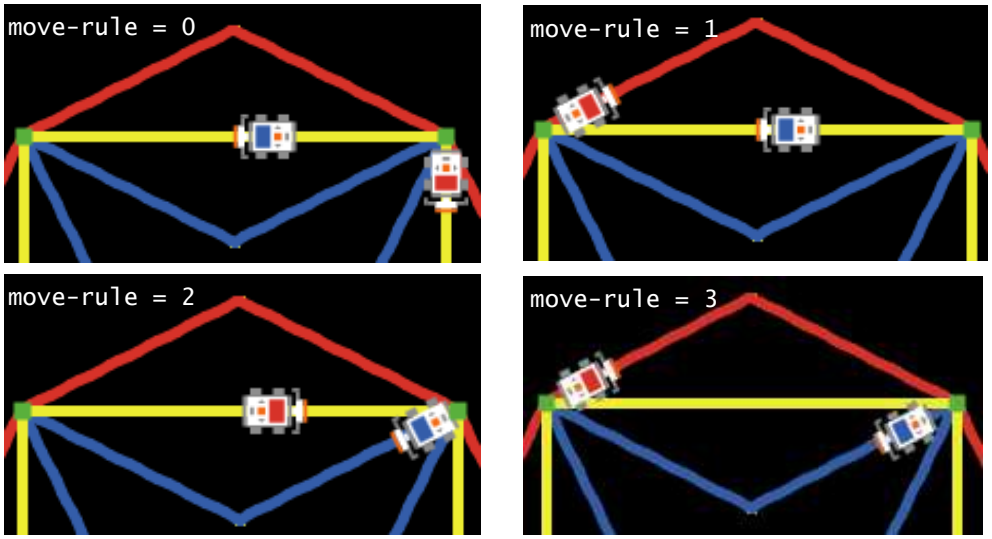


Figure 6.12 – Movements rules set in the play-a-round procedure throught the variable move-rule which can take 4 values

The last pieace of code puts the *dynamic* turtle back on track on the yellow line even if it is coming from either the long way or the short way and also reinitialized the variables *first-half* and *long-way-d* for the next game. The procedure *move-static* cointains the exact same commands but intended for clockwise movements. Furthermore, the *move-static* procedure does not call the *play-a-round.* If it was so it could happen that the two players would play two games in one meeting.

```
17.

play-a-round
  set result-static random 100
  set result-dynamic random 100
  ifelse result-static < static-cooperate [ set result-static 1 ]
     [ ifelse result-static < (static-cooperate + static-defect) [
set result-static 0 ]
          [ ifelse result-static < (static-cooperate + static-
defect + static-random) [ set result-static 3 ]
               [ ifelse result-static < ( static-cooperate +
static-defect + static-random + static-tit-for-tat) [ set result-
static 2 ]
                    [ set result-static 4 ]
               ]
          ]
     ]

  ifelse result-dynamic < dynamic-cooperate [ set result-dynamic 1
]
     [ ifelse result-dynamic < (dynamic-cooperate + dynamic-defect)
[ set result-dynamic 0 ]
          [ ifelse result-dynamic < (dynamic-cooperate + dynamic-
defect + dynamic-random) [ set result-dynamic 3 ]
               [ ifelse result-dynamic < (dynamic-cooperate +
dynamic-defect + dynamic-random + dynamic-tit-for-tat) [ set
result-dynamic 2 ]
                 [ set result-dynamic 4 ]
               ]
          ]
     ]

  set num-games num-games + 1

  let static-strategy 0
  let dynamic-strategy 0

  if result-static = 0 [set static-strategy defect 0]
  if result-static = 1 [set static-strategy cooperate 0]
  if result-static = 2 [set static-strategy tit-for-tat 0 ]
  if result-static = 3 [set static-strategy random-strategy 0]
  if result-static = 4 [set static-strategy win-stay-lose-shift 0]

  if result-dynamic = 0 [set dynamic-strategy defect 1]
  if result-dynamic = 1 [set dynamic-strategy cooperate 1]
  if result-dynamic = 2 [set dynamic-strategy tit-for-tat 1]
  if result-dynamic = 3 [set dynamic-strategy random-strategy 1]
```

```
  if result-dynamic = 4 [set dynamic-strategy win-stay-lose-shift
1]

  calculate-payoff static-strategy dynamic-strategy result-static
result-dynamic

  set-move-rule static-strategy dynamic-strategy

  let translation-static ""
  if result-static = 0 [ set translation-static "Defect     "]
  if result-static = 1 [ set translation-static "Cooperate  "]
  if result-static = 2 [ set translation-static "Tit-for-tat"]
  if result-static = 3 [ set translation-static "Random     "]
  if result-static = 4 [ set translation-static "WSLS       "]

  let translation-dynamic ""
  if result-dynamic = 0 [ set translation-dynamic "Defect     "]
  if result-dynamic = 1 [ set translation-dynamic "Cooperate  "]
  if result-dynamic = 2 [ set translation-dynamic "Tit-for-tat"]
  if result-dynamic = 3 [ set translation-dynamic "Random     "]
  if result-dynamic = 4 [ set translation-dynamic "WSLS       "]

  let translate-static-outcome ""
  if static-strategy = 0 [ set translate-static-outcome " C"]
  if static-strategy = 1 [ set translate-static-outcome " D"]

  let translate-dynamic-outcome ""
  if dynamic-strategy = 0 [ set translate-dynamic-outcome " C"]
  if dynamic-strategy = 1 [ set translate-dynamic-outcome " D"]

  output-type "GAME #" output-type num-games output-type "
PLAYER A: " output-type translation-static output-type "
DECISION:" output-type translate-static-outcome output-type "
PAYOFF: " output-type static-payoff-current-round output-type "
PLAYER B: " output-type translation-dynamic output-type "
DECISION:" output-type translate-dynamic-outcome output-type "
PAYOFF: " output-type dynamic-payoff-current-round output-print ""

end
```

17. The procedure *play-a-round* does several things. In the first part the two players chose the strategies in accordance with the probabilities given by the sliders in the interface. I will try to explain how the mechanism work with the help of Figure 6.13.

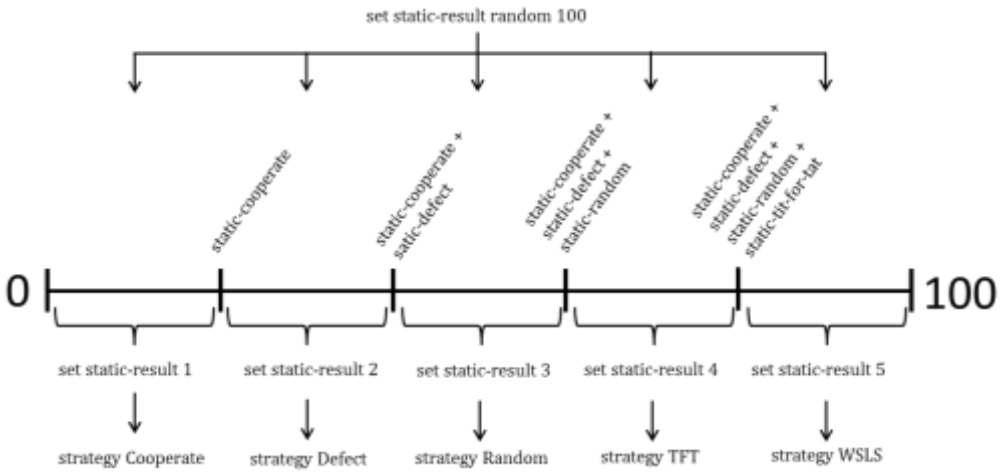


Figure 6.13 – Structure of the play-a-round procedure

At the beginning of the procedure a random number is generated and it is stored in the variable *static-result*. The size of the 5 intervals in which the random number can fall (represented on a line in Figure 5) depend on the values of the sliders *static-cooperate, static-defect, static-random, static-tit-for-tat* and *static-win-stay-lose-shift*. For instance, if *static-cooperate* is set on 100 there will be just one interval and the random number will always fall in it. According to the interval the variable *static-results* is set to 0, 1, 2, 3, 4. This value is then used in the following piece of code to call the correspondent strategy and store the outcome of the decision in the variable *static-strategy* that will contain a 0 if it decides to cooperate or a 1 if it decides to defect. The same mechanism is used for the *dynamic* turtle.

The *play-a-round* procedure also calls the *calculate-payoff* and the *set-move-rule* procedures. I will describe these two procedures later with the help of their NetLogo code.

The last part of the *play-a-round* procedure is devoted to the creation of the line of text in the output area. Through a series of *if* statements I translated the number of the strategy into a string of text and I did the

same for the outcome of the strategy. Then using the command *output-type* I print in the output box the following information:

- *num-games;* the number of games played by the two players;
- *translation-static*; the strategy that player A (static) is using;
- *translate-static-strategy*; the outcome of the strategy: C or D;
- *static-payoff-current-round;* the payoff obtained in the current game for player A;
- *translation-dynamic;* the strategy of player B (dynamic) is using;
- *translate-dynamic-strategy;* the outcome of the strategy: C or D;
- *dynamic-payoff-current-round;* the payoff obtained in the current round for player B.

```
18.

to-report cooperate  [player]
  ifelse player = 0 [
    set num-static-cooperate-games (num-static-cooperate-games + 1)
  ][
  set num-dynamic-cooperate-games (num-dynamic-cooperate-games + 1)
  ]
  report 0
end

to-report defect [player]
  ifelse player = 0 [
    set num-static-defect-games (num-static-defect-games + 1)
  ][
  set num-dynamic-defect-games (num-dynamic-defect-games + 1)
  ]
  report 1
end

to-report random-strategy [player]
  ifelse player = 0 [
    set num-static-random-games (num-static-random-games + 1)
  ][
  set num-dynamic-random-games (num-dynamic-random-games + 1)
  ]
  ifelse random-float 1 < 0.5 [
    report 0
    ][
    report 1
    ]
end

to-report tit-for-tat [ player ]
  ifelse player = 0 [
    set num-static-tft-games (num-static-tft-games + 1)
    ][
    set num-dynamic-tft-games (num-dynamic-tft-games + 1)
```

```
    ]

  let strategy 0
  ifelse player = 0 [
    set strategy dynamic-previous-round
    ][
    set strategy static-previous-round
    ]
  report strategy
end

to-report win-stay-lose-shift [ player ]
  let strategy 0
  ifelse player = 0 [
    set num-static-wsls-games (num-static-wsls-games + 1 )
  ][
  set num-dynamic-wsls-games (num-dynamic-wsls-games + 1)
  ]
  ifelse (dynamic-previous-round = 0) and (static-previous-round =
0) [
    set strategy 0
    ][
    ifelse (dynamic-previous-round = 1) and (static-previous-round
= 1) [
           ifelse player = 0
           [ ifelse random 101 < wsls-slider-static [
               set strategy 0
               ][
               set strategy 1
               ]
            ][
            ifelse random 101 < wsls-slider-dynamic [
              set strategy 0
              ][
              set strategy 1
              ]
              ]
            ][
            set strategy 1
            ]
       ]
   report strategy
end
```

18. Above I have written all the lines of code that refer to the strategies. In this model we have 5 strategies. I this section I will analyze each one of them under the programming profile. Before starting, note that each strategy procedure has a counter (*num-"player"-"strategy"-games*) that keeps track of how many time it has been called by each players. This is the reason why when we use them we have to pass 1 input stating the player that is using it, 0 if it is the *static* and 1 if it is the *dynamic*. This information will be used to calculate the average payoff. Another thing to notice is that we are dealing with *reporting procedures*, that means

that in the body of the procedure we use *report* to report the value of the procedure. The procedures relative to the first three strategies are straightforward: the strategy Cooperate always reports 0, that is cooperate; the strategy Defect always reports 1, that is defect; the strategy Random reports 0 or 1 with a probability of 1/2 (*random-float 1 < 0.5*).

The *tit-for-tat* procedure is not much harder either. It sets the local variable *strategy* to be equal to the *dynamic-previous-move* if it is called by the *static* or sets the local variable *strategy* to be equal to the *static-previous-move* if it is called by the *dynamic.* Finally, it reports *strategy.*

The *win-stay-lose-shift* procedure follows this rules: if both players have cooperated in the last round, than cooperate once again. If both players have defected, then cooperates with a certain probability which is given by the slider *"player"-wsls-slider*. In all other cases it will defect. The first *ifelse* checks for a C/C and if it is so sets the local variable *strategy* to 0 (cooperate); otherwise it checks for the D/D case and if it so cooperates with a certain probability. Otherwise, in all the other cases will defect. At the end, it will report *strategy.*

```
19.

to calculate-payoff [ _static-strategy _dynamic-strategy _result-
static _result-dynamic ]
  ifelse _static-strategy = 0 [
    ifelse _dynamic-strategy = 0 [
       set-payoffs _result-static 4 0
       set-payoffs _result-dynamic 4 1
       set static-payoff-current-round 4
       set dynamic-payoff-current-round 4
       ][
       set-payoffs _result-static 0 0
       set-payoffs _result-dynamic 5 1
       set static-payoff-current-round 0
       set dynamic-payoff-current-round 5
       ]
     ][
     ifelse _dynamic-strategy = 0 [
         set-payoffs _result-static 5 0
         set-payoffs _result-dynamic 0 1
         set static-payoff-current-round 5
```

```
          set dynamic-payoff-current-round 0
          ][
          set-payoffs _result-static 2 0
          set-payoffs _result-dynamic 2 1
          set static-payoff-current-round 2
          set dynamic-payoff-current-round 2
          ]
    ]
end
```

19. The procedure *calculate-payoff* is a command procedure with 4 inputs. It takes the strategy of both players and the outcome of both strategies. Using these information it is possible to assign the right payoff to the right player also through the use of another command procedure, the *set-payoff* which I will describe below. The *calculate-payoff* procedure also stores the payoff of the game being played in the *"player"-payoff-current-round.* This variables are used for output purposes.

```
20.

to set-payoffs [strategy value player]
  if strategy = 0 [
    ifelse player = 0 [
      set d-static-payoff (d-static-payoff + value)
    ][
      set d-dynamic-payoff (d-dynamic-payoff + value)
      ]
  ]
  if strategy = 1 [
    ifelse player = 0 [
      set c-static-payoff (c-static-payoff + value)
    ][
    set c-dynamic-payoff (c-dynamic-payoff + value)
    ]
  ]
  if strategy = 2 [
    ifelse player = 0 [
      set tft-static-payoff (tft-static-payoff + value)
    ][
    set tft-dynamic-payoff (tft-dynamic-payoff + value)
    ]
  ]
  if strategy = 3 [
    ifelse player = 0 [
      set r-static-payoff (r-static-payoff + value)
    ][
    set r-dynamic-payoff (r-dynamic-payoff + value)
    ]
  ]
  if strategy = 4 [
    ifelse player = 0 [
      set wsls-static-payoff (wsls-static-payoff + value)
    ][
```

```
    set wsls-dynamic-payoff (wsls-dynamic-payoff + value)
    ]
  ]
end
```

20. The procedure *set-payoffs* takes three inputs: the strategy, the value of the payoff and the player. Using these three information it can sort payoffs according to the strategy and the player. The results are store in the variable *"strategy"-"player"-payoff*.

```
21.

to set-move-rule [strategy1 strategy2]
  ifelse strategy1 = 0 and strategy2 = 0 [
    ifelse random-float 1 < 0.5 [
      set move-rule 1
      set static-previous-round 0
      set dynamic-previous-round 0
      set static-payoff static-payoff + 4
      set dynamic-payoff dynamic-payoff + 4
      ] [
      set move-rule 2
      set static-previous-round 0
      set dynamic-previous-round 0
      set static-payoff static-payoff + 4
      set dynamic-payoff dynamic-payoff + 4
      ]
  ][
  ifelse strategy1 = strategy2 [
    set move-rule 3
    set static-previous-round 1
    set dynamic-previous-round 1
    set static-payoff static-payoff + 2
    set dynamic-payoff dynamic-payoff + 2
    ][
    ifelse strategy1 = 0 [
      set move-rule 1
      set static-previous-round 0
      set dynamic-previous-round 1
      set static-payoff static-payoff + 0
      set dynamic-payoff dynamic-payoff + 5
      ][
      set move-rule 2
      set static-previous-round 1
      set dynamic-previous-round 0
      set static-payoff static-payoff + 5
      set dynamic-payoff dynamic-payoff + 0
      ]
    ]
  ]
end
```

22. The procedure *set-move-rule* takes two inputs: the outcome of the strategy of player A (*static*) and the outcome of player B (*dynamics*) from the variables *"player"-strategy*. If both players cooperate then

move-rule gets a random value between 1 and 2 (remember that with *move-rule* = 1 the *static* turtle goes back and with *move-rule* = 2 the *dynamic* turtle goes back) and both players get +4 of payoff. If both player defect then *move-rule* is set to 3 and both players are assigned a +2 gain. If *static* player cooperates and *dynamic* player defects, the former gets 0 gain and the latter gets a +5 gain and the *move-rule* is set to 1. The opposite happens if we are in the last case, that is a D/C.

Now that we understand how the NetLogo model has been created and how it works, we can move to the next section in which I will undertake a plan of simulation to discover what is the winning mix of strategies that player A can use when confronting player B in repeated Prisoner's Dilemma interactions. Then, I will introduce the hardware and the software of the robotic simulation and I will undertake a real simulation through the mean of two Lego NXT robots.

7

A simulation experiment

7.1 NetLogo simulation experiments

In this section, I will run a simulation experiment using the NetLogo model introduced in the previous chapter.

In this model we have two players, Player A (also called Static) and Player B (also called Dynamic). Each player is endowed with five strategies: Cooperate, Defect, Random, Tit For Tat, Win Stay Lose Shift. The probability of using a certain strategy is set through the sliders denoted with the number 1 in Figure 7.1 for Player A and denoted with the number 2 for Player B. When running the model, the two players will meet each other repeatedly through Prisoner's Dilemma like interactions.

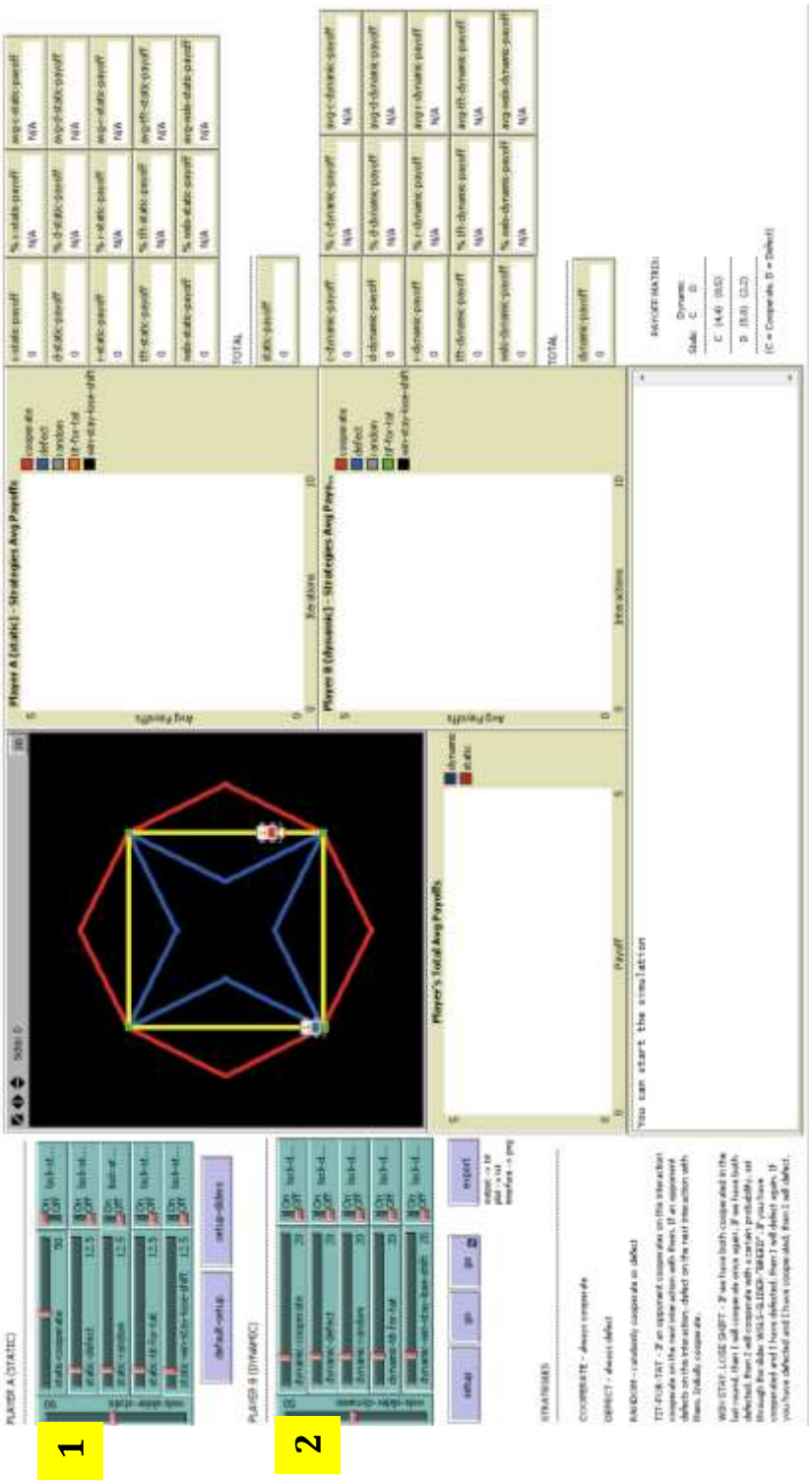


Figure 7.1 – NetLogo interface

The experiment is structured in the following way. Player B always plays with the sliders set to 20 for each strategy. This means that he uses each strategy with a probability of 1/5. The sliders of this player will never change in our simulation experiment. The fact that this player can switch strategy – among his portfolio – every round of the game, makes this model closer to reality. As a matter of fact, in real life it is very improbable that a person undertakes choices always using the same methodology. Sometimes a strategy can be changed in order to disconcert the opponent. This is exactly what this model attempts to simulate.

Player A, instead, will change the probability of using a certain strategy according to the following rules. The slider of each strategy (*static-cooperate, static-defect, static-random, static-tit-for-tat, static-win-stay-lose-shift*) will be set on the values 0, 25, 50, 75, 100 and the remaining sliders accordingly. In Figure 7.2, it is possible to view the slider *static-cooperate* set on the five different values.

Figure 7.2 – Settings of the slider static-cooperate

Once the slider of the strategy under observation is set and all the other sliders are readjusted, the simulation can be run. A total of 25 simulations has been run in order to explore all the possible initial settings: 5 for the slider *static-cooperate*, 5 for the slider *static-defect*, 5

for the slider *static-random*, 5 for the slider *static-tit-for-tat*, 5 for the slider *static-win-stay-lose-shift.* These simulations are intended to explore the importance of a certain strategy in a player's portfolio and to point out the best mix of strategies that Player A can use when competing with a player of the typology of B. After running all the 25 simulations for 1000 games, I recorded the final payoff of each player in Table 7.3. The complete NetLogo interfaces with all the plots and all the monitors can be found in Appendix B.

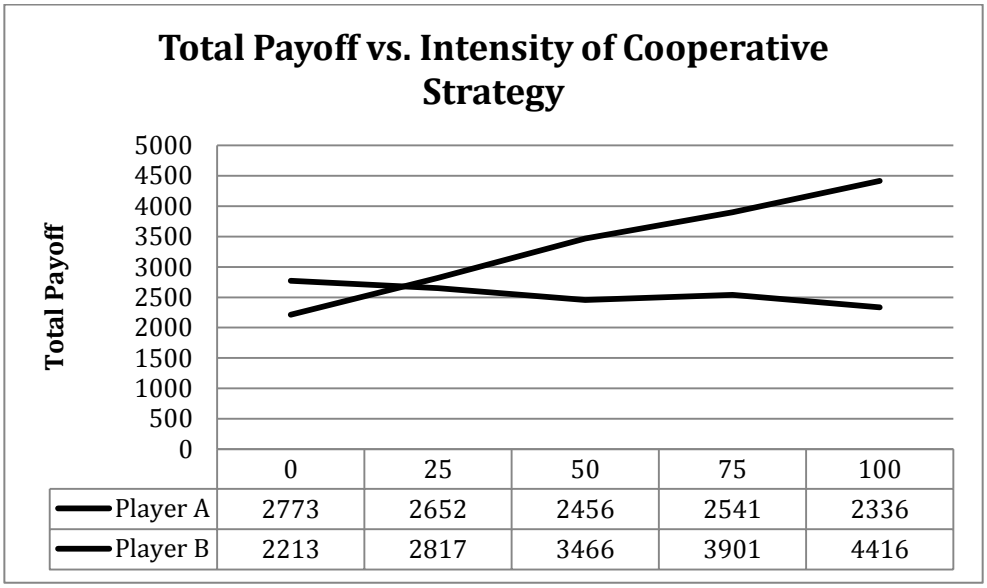
Slider modified	Value	Player A Total Payoff	% change	Player B Total Payoff	% change	Player A - Player B	Player A + Player B
static-cooperate	0	2773		2213		560	4986
static-cooperate	25	2652	-4,36	2817	27,29	-165	5469
static-cooperate	50	2456	-7,39	3466	23,04	-1010	5922
static-cooperate	75	2541	3,46	3901	12,55	-1360	6442
static-cooperate	100	*2336*	-8,07	**4416**	13,20	*-2080*	**6752**
static-defect	0	2616		3116		-500	5732
static-defect	25	2624	0,31	2594	-16,75	30	5218
static-defect	50	2899	10,48	2149	-17,15	750	5048
static-defect	75	3008	3,76	1683	-21,68	1325	4691
static-defect	100	**3020**	0,40	*1320*	-21,57	**1700**	*4340*
static-random	0	2706		2646		60	5352
static-random	25	2726	0,74	2656	0,38	70	5382
static-random	50	2677	-1,80	2827	6,44	-150	5504
static-random	75	2701	0,90	2671	-5,52	30	5372
static-random	100	2720	0,70	2865	7,26	-145	5585
static-tit-for-tat	0	2768		2628		140	5396
static-tit-for-tat	25	2593	-6,32	2718	3,42	-125	5311
static-tit-for-tat	50	2675	3,16	2785	2,47	-110	5460
static-tit-for-tat	75	2668	-0,26	2703	-2,94	-35	5371
static-tit-for-tat	100	2657	-0,41	2662	-1,52	-5	5319
static-win-stay-lose-shift	0	2669		2819		-150	5488
static-win-stay-lose-shift	25	2585	-3,15	2635	-6,53	-50	5220
static-win-stay-lose-shift	50	2775	7,35	2715	3,04	60	5490
static-win-stay-lose-shift	75	2731	-1,59	2596	-4,38	135	5327
static-win-stay-lose-shift	100	2863	4,83	2473	-4,74	390	5336

Table 7.3 – Results of the simulation experiment

In Table 1, I have highlighted the winning player and I also have added columns for the percentage change of the total payoff when increasing the slider by 25, for the difference and the sum of the total payoff of the two players. The difference from the total payoff of Player A and Player B can help to quantify the magnitude of the win/loss of player A. Whereas, the sum of the total payoff of the two players can be interpreted as an indicator of the total gain of the society. In the following sections, I will analyze how the intensity of a strategy influences the final payoff of each player. Here, the word "intensity" must be interpreted as a greater probability of using a certain strategy.

7.1.1 Player A - Cooperate strategy (*static-cooperate*)

At first glance, we can see that when we increase the intensity of the Cooperative strategy over the others, the total payoff Player A decreases and the total payoff of player B increases. A mindless (or unconditional) cooperator is more subject to exploitation by his opponent and this is exactly what happens here, but what we also see is that a player that unconditionally cooperates, brings advantages at a societal level. As a matter of fact, the sum of the payoffs of the two players is the greatest of all 25 simulations exactly when Player A uses a Cooperative strategy every round, that is *static-cooperate* set to 100. Figure 7.4, shows how the gap between the payoffs of the two players expands when the use of the Cooperative strategy increases.



	0	25	50	75	100
Player A	2773	2652	2456	2541	2336
Player B	2213	2817	3466	3901	4416

Figure 7.4 – Line plot of total payoff of Player A and Player B

Furthermore, I have also regressed the total payoff of Player B (independent) on the total payoff of Player A (dependent). What stands out is that the correlation coefficient R is equal to 0.91, that is a strong linear relationship between the two variables. R-squared is equal to 0.83. It indicates that 83% of the variability of Total Payoff A is explained by the Total Payoff B. The coefficient of the regression is equal to -0.18 meaning that when the total payoff of Player B increases by 1 unit, the total payoff of Player A goes down by 0.18 unit[1]. In conclusion we have a strong negative linear relationship between the two total payoffs when Player A plays a Cooperative strategy. When the payoff of Player B goes up, the payoff of Player A goes down.

[1] The regression has been calculated using 4 cases, respectively setting the slider *static-cooperate* to 25, 50, 75, 100.

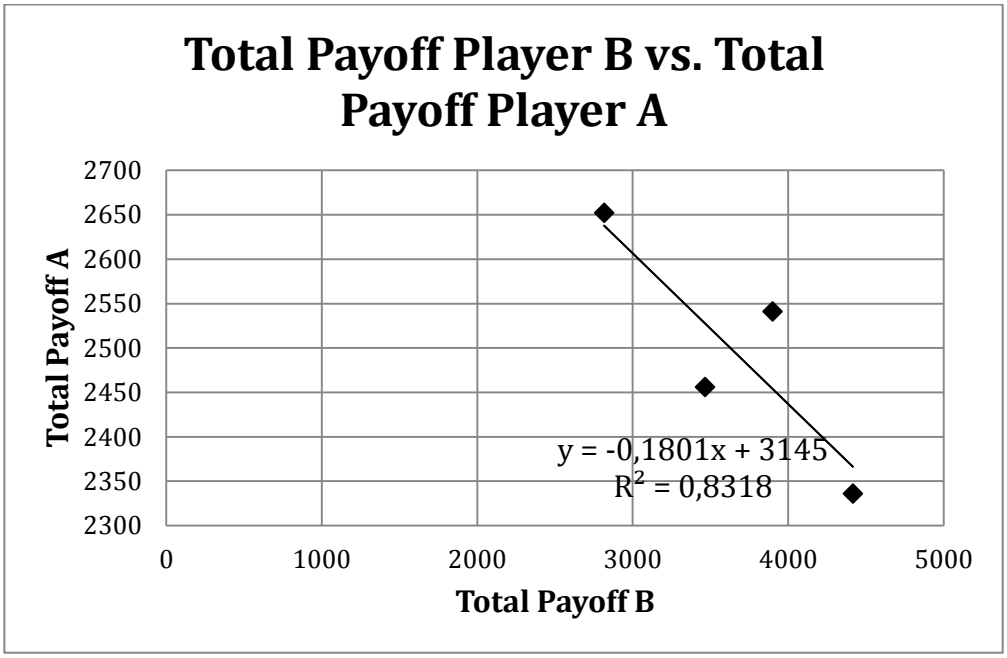


Figure 7.5 – Relationship between total payoff of Player A and B

7.1.2 Player A – Defect strategy (*static-defect*)

When we increase the intensity of the Defect strategy what we find is exactly the opposite situation. A mindless (or unconditional) defector tries to exploit his opponent every round. What we also see is that a player that unconditionally defect, does not brings advantages at a societal level. As a matter of fact, the sum of the payoffs of the two players is the lowest of all 25 simulations exactly when Player A uses a Defect strategy every round, that is *static-defect* set to 100. In Figure 7.6, the total payoff of each player relative to a change in the slider *static-defect* is depicted.

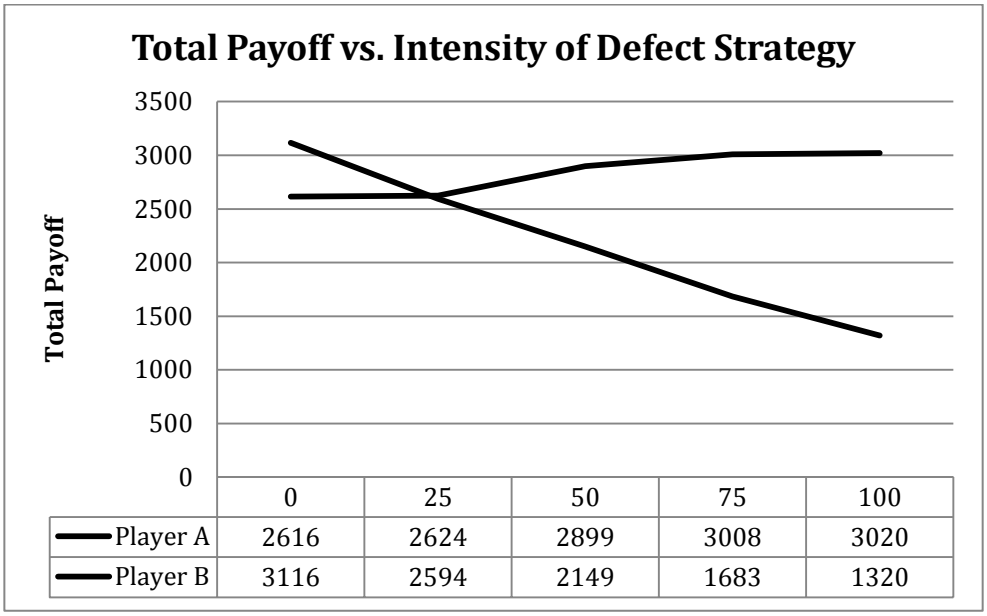


Figure 7.6 – Line plot of total payoff of Player A and Player B

Furthermore, I have also regressed the total payoff of Player B (independent) on the total payoff of Player A (dependent). What stands out is that the correlation coefficient R is equal to 0.93 that is a strong linear relationship between the two variables. R-squared is equal to 0.86. It indicates that 86% of the variability of Total Payoff A is explained by the Total Payoff B. The coefficient of regression is equal to -0.31 means that when the total payoff of Player B increases by 1 unit, the total payoff of Player A goes down by 0.31 unit. In conclusion, we have a strong negative linear relationship between the two total payoffs when Player A plays a Defect strategy.

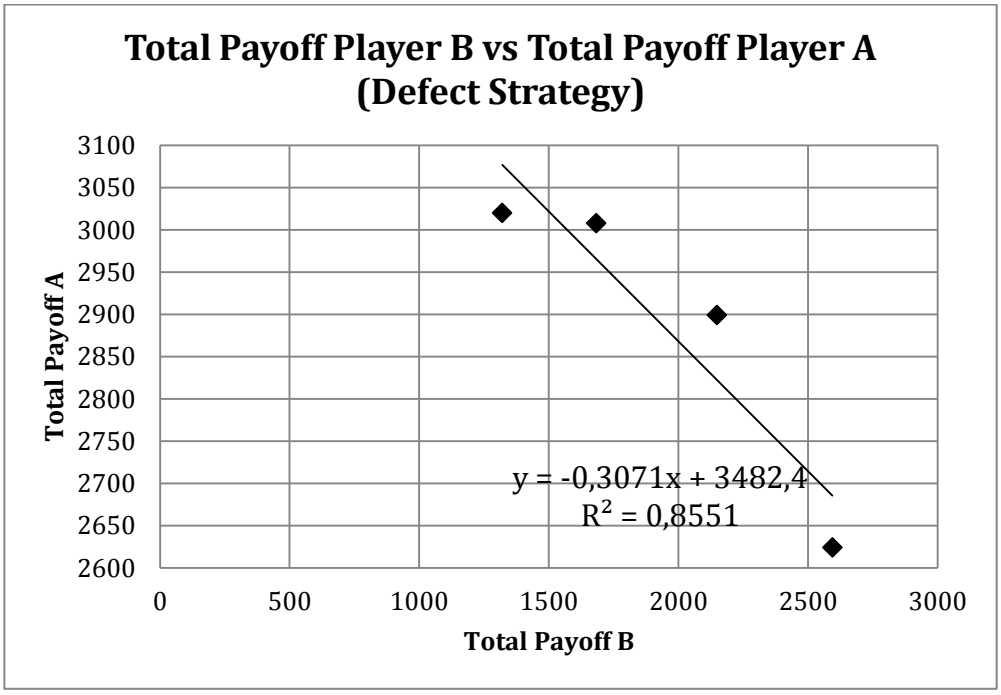


Figure 7.7 – Relationship between total payoff of Player A and B

7.1.3 Player A – Random strategy (*static-random*)

When we increase the intensity of Random strategy for Player A the results are not very clear. In our simulations, Player A wins 50% of the times. What's more is that the total payoff of both players is not very different. Playing a Random strategy against a players endowed with a portfolio of five strategies used with the same intensity is not the best possible decision for Player A. Setting the slider *static-random* to 100 will lead Player B to a higher total payoff.

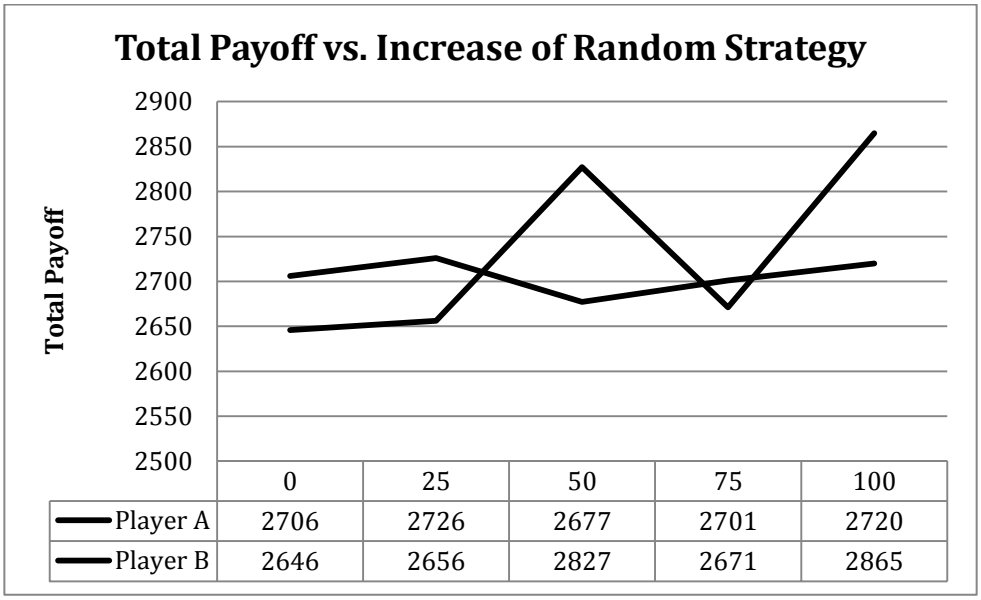


Figure 7.8 – Line plot of total payoff of Player A and Player B

What we also see from Figure 7.8 is that there is no relationship between the payoff of Player B and A. The R-squared of the regression is 0.1 that means that there is no correlation between the two total payoffs. These results are very similar to the ones we would get playing a Random vs. Random.

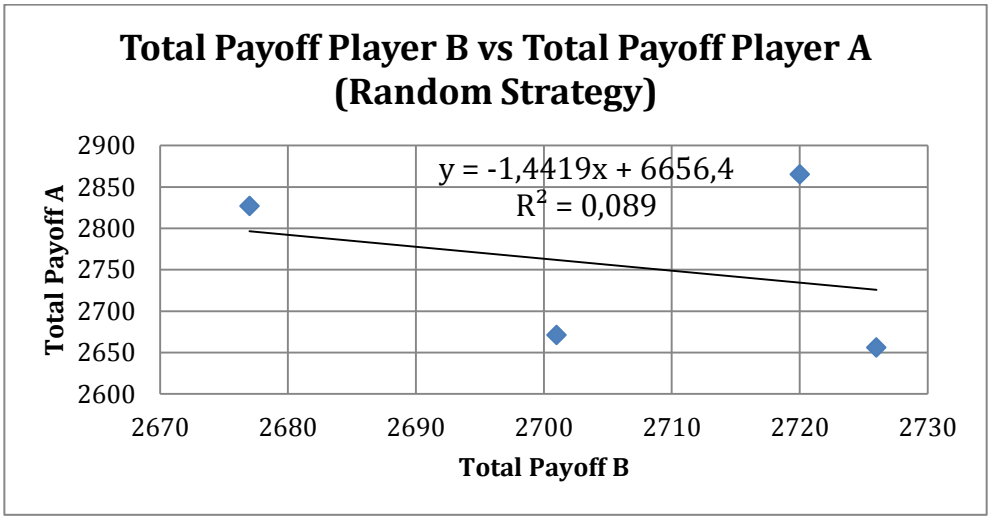
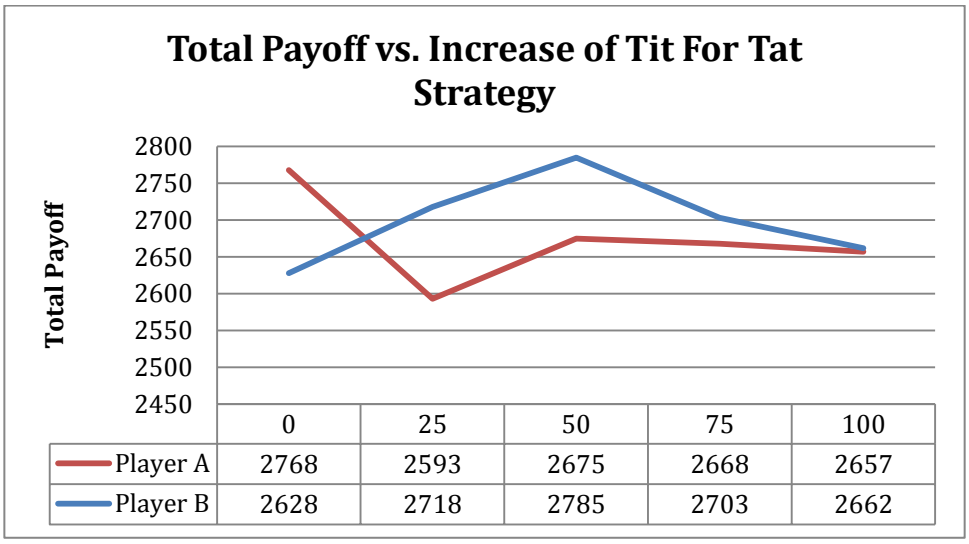


Figure 7.9 – Relationship between total payoff of Player A and B

7.1.4 Player A - Tit For Tat strategy (*static-tit-for-tat*)

When I started to look at the results of the simulation concerning the Tit For Tat (TFT) strategy, I could not make sense out of it. Its performance was not really good. Player A wins when he is not using this strategy at all (*static-tit-for-tat* = 0). When he starts to use this strategy even with a little intensity Player B wins. Also, the total payoffs of both players after 1000 games are very close to each other when the slider *static-tit-for-tat* is set on 75 and 100 (see Figure 7.10).



	0	25	50	75	100
Player A	2768	2593	2675	2668	2657
Player B	2628	2718	2785	2703	2662

Figure 7.10 – Line plot of total payoff of Player A and Player B

Knowing that the TFT performs poorly against the Random strategy I started to explore in this direction. The performance of TFT against Random can be analyzed as follows. Since Random defects for an average 50% of all moves, half of TFT's moves are punishments (defections) and the other half are rewards (cooperative moves). Now, since Random does not cares what moves the other player makes, it responds – on average or in the long run – half of the punishments by TFT with defection and half of them with cooperation. The same holds for the rewards of the TFT player. Consequently, TFT gets an average

score of (T + P + R + S)/4, which is equal to 2.75 with the payoff matrix implemented in this simulation model (Eckhart, 2008). When running the simulation with Player A using only the TFT strategy (*static-tit-for-tat = 100)* I obtained an average payoff of 2.66 (Figure 7.11) which is very similar to the one calculated above. This result gives even more emphasis to the idea that Player B's mix of strategy brings a lot of "randomness" in his decisions.

Figure 7.11 – Average TFT payoff for Player after 1000 games highlighted in the red box

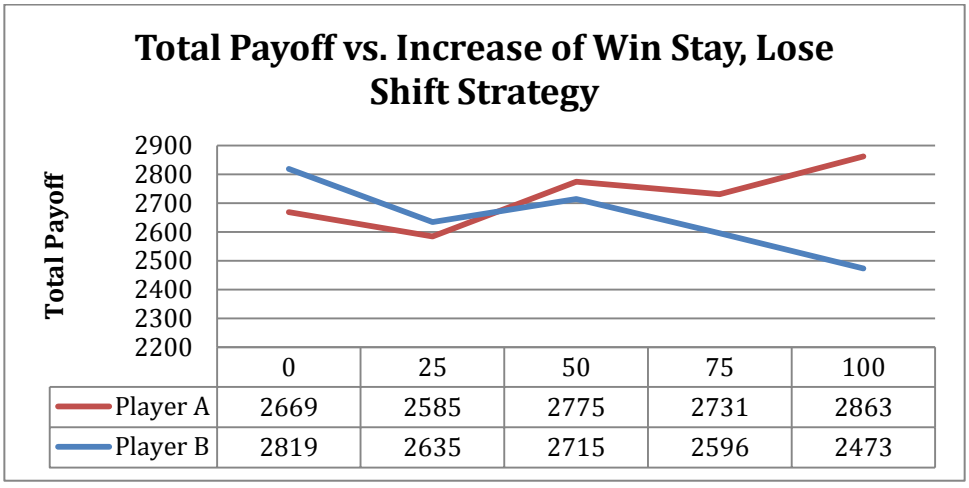
7.1.5 Player A - Win Stay, Lose Shift strategy (*static-win-stay-lose-shift*)

The Win Stay, Lose Shift (WSLS) strategy is the most complex one in my simulation experiments. It works in the following way:

> *If we have both cooperated in the last round, then I will cooperate once again.*
> *If we have both defected, then I will cooperate (with a certain probability).*

If you have cooperated and I have defected, then I will defect again.
If you have defected and I have cooperated, then I will defect.

This means that whenever the two players have done the same, then Player A will cooperate; whenever the two players have done something different, then Player A will defect. In other words, if Player A does well (obtains a high payoff), he will repeat his move. If he is doing badly (obtains a low payoff), he will change what he is doing.
In Figure 7.13, we can see that increasing the intensity of this strategy we get better results for Player A. As a matter of fact, when we set the slider *static-win-stay-lose-shift* on 50, 75 and 100 the total payoff of Player A are always greater than the total payoff of Player B and this becomes even more pronounced when WSLS is the only strategy played by Player A.



	0	25	50	75	100
Player A	2669	2585	2775	2731	2863
Player B	2819	2635	2715	2596	2473

Figure 7.13 – Line plot of total payoff of Player A and Player B

The WSLS strategy understands that mindless cooperators can be exploited and as the name suggests, WSLS continues exploiting its

opponent when it is not punished with revenge. The key of success of Player A when using this strategy comes from the "randomness" of the decision of Player B. WSLS does not have to interpret and remember the opponent's move. All it has to do is monitor its own payoff and make sure that it stays ahead in the game. This is particularly relevant in our case. When the opponents plays a variety of strategies mixed together, could send out wrong signals to other players. Probably, a player endowed with a mix of 5 strategies is going to play the games in such a way that they could be interpreted as random decisions by the other player who struggles to make a sense out of the opponent decisions. This is what is probably happening here. We know that WSLS is a strategy that does well when confronted with a lot of randomness and we also know that TFT does poorly against the Random strategy.

Furthermore, the way of thinking behind WSLS feels more "human". If we get away with exploiting someone in this round the we continue to do it in future rounds. WSLS on the contrary of TFT after a mutual defection tries to restore a good relationship and also understand when to exploit unconditional cooperators (Nowack & Highfield, 2011).

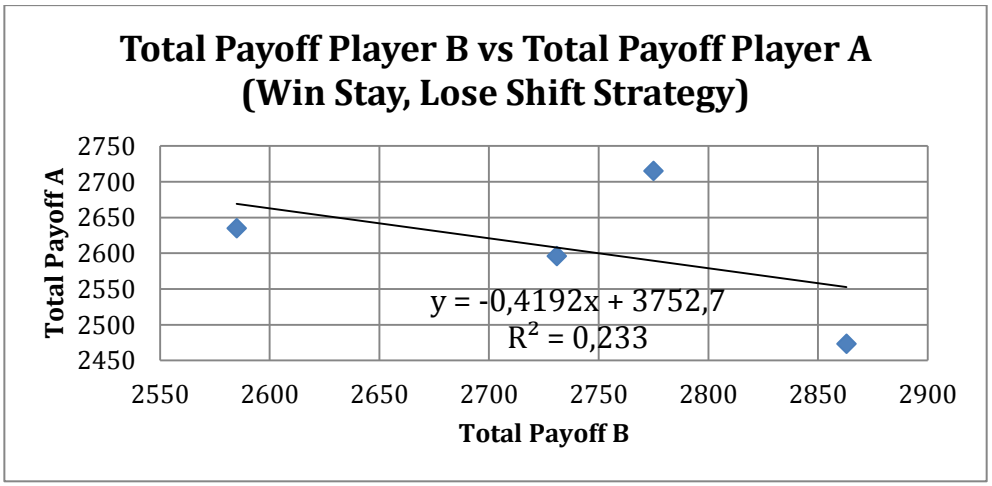


Figure 7.14 – Relationship between total payoff of Player A and B

7.2 NetLogo Simulations Experiment Conclusions

In conclusion, when we face a player endowed with five strategies (Cooperate, Defect, Random, Tit For Tat and Win Stay, Lose Shift), all used with the same probability, the strategy that will lead us to the greatest total payoff is the Defect strategy. However, if we look at both individual total payoff and the total payoff of the society, the best decision would be to use the strategy Win Stay, Lose Shift every game. Using this strategy, we will not only achieve a payoff higher than our opponent (2863 > 2473 and also not much smaller than the payoff of 3020 achieved with the Defect strategy), but we will also convey a significant gain in the total payoff of the society (compared to the Defect strategy, 5336 > 4340. See Table 7.15).

Slider modified	Value	Player A Total Payoff	Player B Total Payoff	Player A - Player B	Player A + Player B
static-defect	100	3020	1320	1700	4340
static-win-stay-lose-shift	100	2863	2473	390	5336

Table 7.15 – Comparison of the two best strategies of the simulations

We also saw that in this particular situation, the Tit For Tat strategy does not perform very well. This could be because a player endowed with a mix of five strategies, is going to play the games in such a way that they could be interpreted as random decisions by the other player that can have a hard time to make a sense out of his decisions and it is known that TFT does not perform very well against Random strategy.

7.3 A real-world experiment: robotic simulation

In this section, I am going to present the robotic simulation. I will begin describing the hardware (two Lego NXT kits) and the environment;

then I will introduce the software used for the programming of the robots (Not eXactly C). Finally, I will replicate in the real world some of the NetLogo simulations presented in the previous section and I will confront the results highlighting differences, gains, losses, problems or improvements of the two simulation methodologies.

7.3.1 The hardware: Lego Mindstorms NXT 2.0

For the construction of the two robots employed in the real simulation I have used two Lego Mindstorms NXT 2.0 sets. The kit NXT 8547 includes the following hardware:

- NXT Brick (number 1 in Figure 7.16). It uses a 32-bit ARM7 Atmel microprocessor running at 48 MHz with 256 kilobytes of non-volatile flash memory as well as 64 kilobytes of RAM. It is possible to download a maximum of 64 files on the NXT Brick's flash memory.
- LCD display is large 100 pixels wide and 64 pixels tall. Each pixel can individually set or cleared. It has a very fast 17 millisecond refresh cycle, which means you can also display animations.
- Sounds. The NXT includes support for simple tone generation. They can play MIDI files. One great feature in the NXT is the support for playing sound files that contain standard pulse-code modulation (PCM) samples. PCM is a method for representing an analog waveform digitally by uniformly sampling or recording the magnitude of the signal. Included with the NXT there is a huge collection of NXT sound files that are easy to incorporate into the programs using NXT-G, NXC or NBC.
- Communication. The NXT supports either wireless communication via Bluetooth at very fast 460.8 Kbits per second

or a super-fast (12 Mbits per second) wired USB communication using a standard USB cable that comes with the set. Because Bluetooth connections between a PC and an NXT or between two NXT are point-to-point, they will only ever affect the specific brick to which the message or the program is sent.

- One ultrasonic sensor (number 2 in Figure 7.16), it can detect the distance from other objects (it measures a distance between 0 and 255 cm with a precision of +/-3 cm);
- Two touch sensors (number3 in Figure 7.16), that can be activated through the button;
- One color sensor (number 4 in Figure 7.16), it can recognize colors and if programmed it can allow the robot to follow a line. It can distinguish 6 colors and it can work as a light sensor as well.
- Three NXT motors (number 5 in Figure 7.16). These are very sophisticated motors, they have a built-in rotation sensor that can measure 360 degree of rotation per revolution. The rotation values are sent back to the NXT via the same cable and port to which the motor is connected. A program running on the NXT can easily access these rotation counter values. This means that each motor is both an output that can be used to move or manipulate the robot's environment and an input that can provide feedback about the robot's motion and its surrounding to programs that you write. The built-in rotation sensor also allow you to write code that tells the motors to rotate a certain number of degrees either clockwise or counter-clockwise (Hansen, 2009).

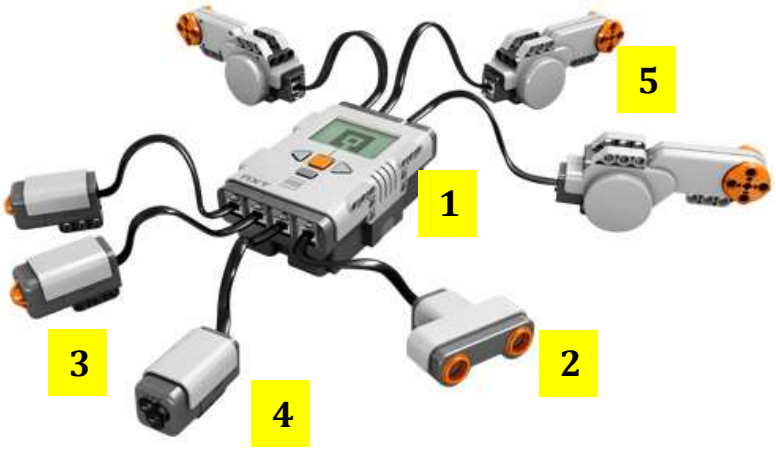


Figure 7.16 – Lego Mindstorms NXT Brick, sensors and motors

For what concern the shape of the robots, I have opted for the one depicted in Figure 7.17. I discovered this robot model on the website www.nxtprograms.com and I realized that the mechanical design created by Dave Parker was perfect for my purpose. It had all the characteristics I needed for my simulation: a color sensor for line following, touch sensors placed in the front bumper and a ultrasonic sensor to watch ahead. The complete building instructions of this robot model (called Multi-Bot) can be found on the website www.nxtprograms.com/NXT2/multi-bot.

Figure 7.17 – Robots used in the real simulation

Player A is recognizable from Player B by its black wheels as it is clearly visible in Figure 7.17. Player A is able to follow the yellow path in a clockwise direction and when he must take the long way he follows the red path, whereas Player B is able follow the yellow path in a counterclockwise direction and when he must take the long way he follows the blue path.

7.3.2 The real environment

The real environment for the robotic simulation is a 300x300 cm black sheet of paper. The length of the sides of the yellow path is 160 cm; the long ways (red and blue) are structured in the exact same way as in the virtual simulation, that is, the proportions of the catheti are respectively 1 and 0.5. Figure 7.18 shows a moment of the construction of the environment. All the measurements are available on Figure 7.19.

Figure 7.18 – The construction of the environment

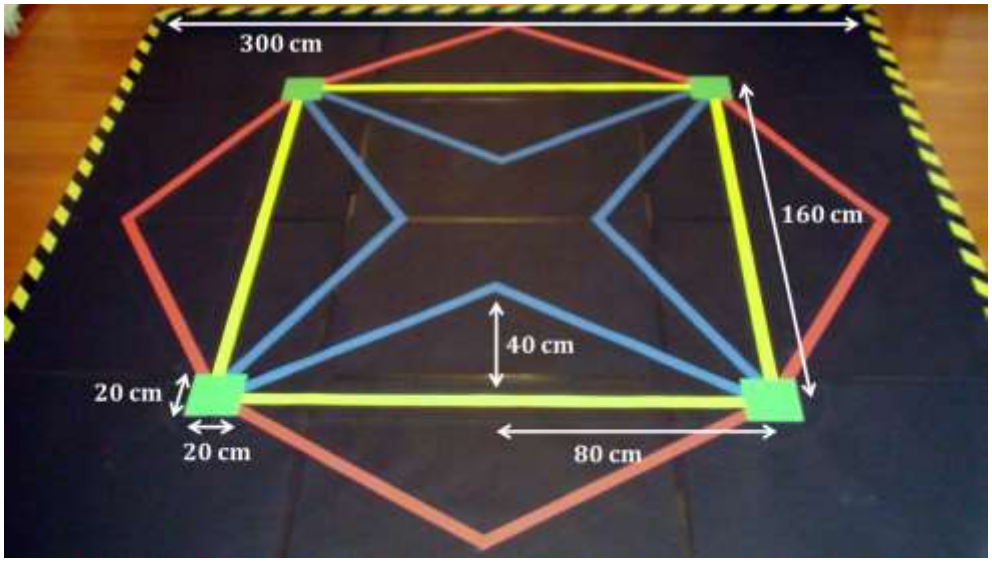


Figure 7.19 – The measures of the environment

The sizes of the real environment are not by chance. They have been chosen to allow robots to take turns in the corners and to have enough space when they are running against each other in the long way and in the short way without obstructing the passage. These kind of problems, are usually not taken into account when building a virtual simulation. As a matter of fact, turtle in NetLogo can overlap but robotic turtles in real world demand their own space for movements.

7.3.3 The software of the robots: Not eXactly C

The software used for the programming of the robots is NXC, short for Not eXactly C. NXC is a high-level programming language for the Lego Mindstorms NXT and it has a syntax like C. The Integrated Development Environment for NXC is the Bricx Command Center. The code of the two robots has been written with the help of my colleague I. Alvino and it is reported entirely in Appendix C. Two distinct programs have been written for the two robots. This was due to the fact that the movements that they make are specular and also because the implementation of the

Bluetooth communication required a structure of the type server-client, in this context called master-slave. The functioning of the robots works as follows:

1. They follow the yellow, red or blue line using the color sensor according to the value of the variable *move_rule* (I will explain it in point 3.c.i how this variable works). More precisely they follow the external edge of the line. The line following procedure uses a very simple, yet effective, algorithm. Consider Player A, the robot with the black wheels that travels in the clockwise direction, following the yellow line. If his color sensor sees black, it asks the left motor (connected to the left wheel) to rotate while stopping the right motor; if his color sensor sees yellow it asks the right motor (connected to the right wheel) to rotate while stopping the left motor. It basically uses a simple "zig-zag" method of line following where the robot is constantly turning back and forth as it sees either side of the color boundary. The robot is always either turning left or right, so it is never actually straight, even when the line is straight.
2. They continuously watch ahead using the ultrasonic sensors. If the sensor detect an object, in our case the other robot, within 10 cm asks the motors to slow down. This is done to prevent the robots to crash into each other when they are face to face. So, before they touch each other with the front bumper they will both reduce their speed.
3. They continuously check if the front bumper is being pressed. If it is so, they enter the *play_a_round* procedure which is perhaps the most important one. When this procedure is called the two robots carry out the following tasks:

a. Player A (the Master) opens a text file in which all the output results are going to be written.
b. Both palyers decide which strategy to use according to the initial setup of the variables “PLAYER”_COOPERATE, “PLAYER”_DEFECT, “PLAYER”_RANDOM, etc.. This variables work in the exact same way of the sliders in the NetLogo model.
c. Player A and Player B engage in a Bluetooth communication in which they exchange the following information:
    i. Their final decision, that is Defect (0) or Cooperate (1);

       The Master uses this information to calculate the outcome of the game, payoffs and to set the variable *move_rule* accordingly. The variable *move_rule* is then sent to the Slave. It contains the movement rules (who has to take the long way) and it is coded in the following way. If *move_rule* = 0, Player A goes back and takes the long way, while Player B waits 7 seconds and then goes forward in the short way. If *move_rule* = 1, we have the opposite situation. If *move_rule* = 2, both players go back and take the long way. The *move_rule* is set using the same instructions as the virtual simulation. For instance, if Player A cooperates and Player B defects, the former will go back and take the long way in sign of cooperation.

       The Slave instead uses the final decision of the Master (C or D) just for strategies such as Tit For

Tat who need to know the decision of the opponent in the previous round. In conclusion, all the computations are done by the Master and then are communicated to the Slave in terms of movement rules.

ii. Player B sends the strategy used, so that Player A (the Master) can print this information in the output file.

4. In the function *play_a_round* of the Master robot, the output file is written. It contains the same information we have in the output box of the NetLogo model: the strategy, the decision, the current payoff of each player for every round.

Now that the hardware, the real environment and the software for the robots have been introduced, we can move to the next section in which a real simulation will be undertaken.

7.4 Real world simulation: Robotic Iterated Prisoner's Dilemma

In this section, I will implement a robotic simulation experiment using the Lego NXT robots and the environment above presented. The simulation experiment that I am going to carry out in the real-world setup is the exact same one that has been undertaken in Chapter 6 with the NetLogo model. I did not replicate all the 25 simulations that I have run with the virtual model, but I reproduced just the following cases:

- Player A plays the strategy Defect with probability 75% and the remaining four accordingly;

- Player A plays the strategy Random with probability 75% and the remaining four accordingly;
- Player A plays the strategy Tit For Tat with probability 75% and the remaining four accordingly;
- Player A plays the strategy Win Stay, Lose Shift with probability 75% and the remaining accordingly.

The choice fell on these three simulation because they were the ones with the most interesting results and performing a robotic simulation allowed for a deeper analysis. The three simulations with the respective initial settings have been run for 10 games. The robotic simulation was very interesting to watch and made me realize things that I did not take into account in the virtual simulation that could make the model more realistic, but I will discuss about this in the following section in which a confrontation and a discussion about the two methodologies of modeling will be undertaken. During the robotic simulation experiments, I have shot some videos which are available at the following webpage: http://web.econ.unito.it/terna/tesi/grimaldi. The total payoff of each robotic player are displayed in the following table.

Slider modified	Value	Player A Total Payoff	Player B Total Payoff	Player A - Player B	Player A + Player B
static-defect*	75	32	12	20	44
static-random*	75	24	24	0	48
static-tit-for-tat*	75	29	24	5	53
static-win-stay-lose-shift**	75	28	29,7	-1,7	57,7

* One simulation of 10 games
** Average of 3 simulations of 10 games

Table 7.20 – Robotic simulations results

When Player A uses the Defect strategy with probability equal to 75%, over 10 games, he obtains a much higher payoff than Player B. This result leads to the same conclusions of the virtual simulation in NetLogo. What is more, always defecting leads to the high possible payoff for Player A in the short run.

When Player A uses the Random strategy with probability equal to 75%, over 10 games, he obtains the same payoff of Player B. This result leads to the same conclusions of the virtual simulation in NetLogo in which the games where 1000.

When Player A uses the TFT strategy with probability equal to 75%, over 10 games, he obtains an average payoff of 2.9, whereas Player B obtains a respective payoff of 2.4. Such result is consistent with the one seen in NetLogo after 1000 games where the average payoffs of the two players were very similar.

When Player A uses the WSLS strategy with probability equal to 75%, over 3 simulations of 10 games each (we wanted to be more precise for this simulation), Player B seems to prevail, even if with a small gap between the two. This result is opposite to the one obtained through the virtual simulation in which Player A "asymptotically" wins after 1000 games.

In conclusion, the result obtained by the two different (by methodology) simulations are almost all compatible, except that in the robotic simulation there is more emphasis to look at the short run. As in the virtual simulation, when we face a player (in our case a robot) endowed with five strategies (Cooperate, Defect, Random, Tit For Tat and Win Stay, Lose Shift), all used with the same probability, the strategy that will lead us to the greatest total payoff is the Defect strategy. Instead, the strategy Win Stay, Lose Shift could be a good

choice when facing an opponent for 1000 times, but it does not perform very well when played for very few interactions.

7.5 Confrontation between methodologies: virtual vs. robotic simulation

This section is the most important one in my thesis. Here, I will discuss about the two simulations under the methodological profile. I will give some considerations about the work done and I must admit that I was amazed to see that the contributions given by the real-world simulation were much greater than what I initially expected and were visible in fields that I was not even thinking about at the beginning of my research.

The first day I started my project, I was really interested in exploring an innovative branch of agent-based simulations. I wanted to introduce something different, pushing the innovation in the field of simulation a little forward.

Using real robots as agents for a simulation in the real world was not an easy task. It influenced the way the experiment has been thought and created in first place. For instance, the number of "participants" had to be very limited due to financial reasons. The way in which the robots interacted and communicated among each other had to be feasible and safe (to initiate a communication the two robots have to bump into each other, but before doing so they slowdown in order to avoid a disastrous collision). The real-world environment in which the simulation took place had to allow movement of real objects that demanded their own physical space. As a matter of fact, the length of the main path and of the long ways had to permit the passage of the two robot without leading them to a crash or a collision. This kind of problems instantly loses visibility when building a virtual model in which agents do not occupy

physical space and whose representations can overlap on each other. In addition, the environment colors have been chosen in order to provide the best possible performances of the color sensor. For instance, the black background absorbs better light reflections than the white background initially thought, with a positive effect on the color sensor performance. All this was certainly an eye opener about aspects of the model that can be easily neglected when coding a virtual simulation, even if sometimes their presence can actually be relevant to the final outcome of the simulations.

Furthermore, the simultaneous creation of the two simulations pointed out some mistakes that I was making when programming the virtual model. The most relevant error that the *bifocal* approach brought to surface was an analogy error. As a matter of fact, the way that the NetLogo code allowed the turtles to move along the blue or red path was a distortion of reality that did not become evident until the virtual simulation was confronted with the physical one. In the virtual model when the turtles take the long way, they sum/subtract 1 or 0.5 to their *xcor* and *ycor* (according to their position in the *view*) implicitly modifying their speed of move. In doing so, a turtle that follows the long way takes the same time of a turtle that follows the main way in order to complete a lap of the square. This is not what we see in reality where the deviation from the shorter path entails consequences in terms of time, effort and resources.

The process of creating both models was very engaging and the ability to see their simultaneous development brought a lot of emphasis on the environment design. Virtual models can easily ignore one fundamental process of physics, namely energy loss. Agents on the computer can freely move in the virtual world without experiencing any friction, unless the modeler decides to include it in the model. When dealing

with the physical world, modelers do not have that option (Blikstein & Wilensky, 2007). In our case, the problem of energy loss can be extended to the battery life of the two "actors" of the simulation. Such consideration could widen our way of thinking in the following way. Two players, who repeatedly interact with each other, motivated to take a short path in order to complete a fast lap of the route, could take into account in their decision-making process, the physical state of their body, namely fatigue or physical injuries (and therefore the consequences in terms of precision and rapidity of the movements). Creating just a virtual simulation might not have drawn attention to these aspects. In the real simulation instead it was visible and clear that after many laps the two robots were losing stamina.

Other three important characteristics were pointed out in the construction and analysis of the physical model: (i) "noisy" behavior of the players; (ii) time scale of human interaction and (iii) also some epistemological consideration.

During the physical simulation something not expected occurred more than once. When a robot decided to take the long way in sign of cooperation and, therefore went back to the green patch and turned right/left to take its respective longer route happened to miss to detect the red/blue line. Thus, it came back on the yellow line. It basically said to do one thing – cooperating by taking the long way and leaving the passage to the other robot – but instead it accomplished another action. The hardware failure that I happened to see more than once could be brought by a variety of reason (an electrical surge, bad light conditions, etc.), but that's not the point. Someone who is studying game theory for the first time could not have heard about noisy interactions (wrong interpretation of the opponents decision), but could have reached the same conclusion when seeing it happening in reality. Hardware failure

could have the following loose, but realistic interpretation. In nature a person or his body can fail to do something, and sometimes that can lead to undesired consequences. For instance, it could happen when other people misinterpret your bad action thinking it as voluntary, when it actually is not and begin a retaliation, undermining future cooperation. In this case, the fact that the simulation was carried out in the real world actually had a concrete and tangible impact on the study of the dilemma.

Another important contribution of the physical simulation was given in terms of time conception. When we virtually simulate a model we often look at the long run and we try to find asymptotical results but social interactions do not operate on long time scales .The real simulation "told" us to refocus our attention on few interactions between two player. How many time in life two opponent players get to interact 1000 times? Strategies that seemed best in the long run revealed themselves to be not so good when the number of meeting dropped to 10. For instance, we saw that the strategy Win Stay, Lose Shift could be a good choice when facing an opponent of the type described in Section 7.1 for 1000 times, but it suddenly loses strength when played for very few interactions.

The third important contribution brought up by the physical model is an epistemological one, that is, it refers to the basic assumptions of modeling. Even if the real simulation is thought to have some drawbacks (for instance the speed of the simulation, imprecision of movements, hardware failure, need of physical space), I believe that they are actually the strength of our approach. In addition, the environment and its constraints are "natural". The real-world environment allows to explore aspects that are very hard and tedious to include in a virtual simulation, such as asynchronous spatial

movements. The fact that the physical model is set in the real world can make us think that by construction, it can be said to have less initial assumptions. As Brooks (1990) states "the world is its own best model. It is always exactly up to date. It always contains every detail there is to be known".

An aspect of the physical simulation that I did not expect to be so powerful at the beginning of my work is the educational one. When I started this project I was sure that being able to watch the simulation, as a tangible experiment, could help a lot non-specialists to understand agent-based simulation. Instead, the pedagogical trait was much greater than I was initially presuming. The developing of a *bifocal* simulation allowed me to improve significantly my programming skills. When running a physical model in the real world, one can see at a glance how the robots are interacting. It is right in front of your eyes. Everything you program can instantly be tested and a real feedbacks can be obtained. There are no layers of abstraction to obfuscate the dynamics of the interactions between the agents and the "world" (Brooks, 1990).

In conclusion, I can state that the robotic simulation was certainly an added value to the simulation project. It was able to give us many insights on aspects otherwise neglected. Several points of view and perspectives are needed to make sense out of a problem, especially if the problem at issue has already been studied and analyzed in every detail exactly as the Prisoner's Dilemma.

7.6 Real-world applications

I would like to devote this section to explain how and when the simulation model, developed in my thesis project, can be used to help decision maker to face and solve real-world problems.

Imagine a situation as follows. Two competing bus lines have to take tourists at a ski resort in high altitude. In doing their business, they have to face the fact that the short road leading to the ski resort can allow the passage of only one bus at the time. We then would have to face a situation very similar to the one depicted in the NetLogo and in the robotic model presented in my thesis. The dilemma becomes even more clear when we talk about the profit of the two companies. Both firms would like to take the short way as it allows faster laps (thus more customers paying the ticket) and less fuel consumption, but if both firms take this decision they will find themselves stuck in the narrow road and they will have to put into reverse and go back to the beginning. Both players are tempted to defect in order to obtain the greatest benefit, but doing so they will find themselves in the worst possible scenario. We would have a Prisoner's Dilemma like interaction. The model presented in this thesis could help to analyze and study situation of the type offered above. The decision maker can hypothesize different types of opponents simply by setting the sliders of the dynamic player to different value. An assessment of the strategy or the mix of strategies to use, can be done running the model and looking at the total average payoffs of the players.

On the contrary of what one may think, the situation described above is not that unusual in reality. Watching a documentary called *Deadliest Roads* on the *National Geographic* channel, a situation very similar was showed. A bus line taking workers to a construction site at the top of a mountain in India had to face dump trucks bringing building materials travelling on the same road. The street was narrow and it allowed the passage of only one vehicle at the time. Furthermore, truck drivers were paid on the base of the number of loads delivered, and this was pushing them to take excessive risks in order to complete more laps at

the end of the day. In such a case, the NetLogo model could be used setting the sliders of the opponent player to more defective strategies. The examples discussed above are just few of the many situations that could arise in the real world in which our model could be applied.

8

Conclusions

8.1 General conclusions

My thesis project aimed at analyzing and comparing two different kinds of methodology, namely a virtual and a robotic agent-based simulation.
At present, robotic simulation for the social sciences is gaining more and more attention. This is due to the falling costs of robotic kits that nowadays can be found in most of the schools and universities. Just few year ago, it would have been unimaginable to see robots outside engineering laboratories. Not only they are becoming more

inexpensive, but they are also becoming easier to use and suitable to non-technical persons.

The simulation model that I developed for my dissertation, consists of two players that meet each other repeatedly through Prisoner's Dilemma like interactions. Each player is endowed with five strategies (Cooperate, Defect, Random, Tit For Tat, Win Stay Lose Shift) whose intensity of utilization can be set through sliders before running the model, allowing the decision-maker to create many initial scenarios similar to real problem that he/she is facing. In order to do so, I created an environment in which the two robots face the decision to either cooperate or defect and this entails consequences in terms of distance to travel. Then I replicated the same model in NetLogo for a comparison to seek differences, gains or drawbacks of the two methodologies.

As results showed, the robotic simulation was certainly an added value to the virtual one. In fact, I believe that the two methodologies are complementary. The physical simulation was able to give us many insights on aspects otherwise neglected. It allowed us to better calibrate the virtual model to the real environment by highlighting errors of analogy.

Furthermore, the imprecision of real world led us to some key insight about the Prisoner's Dilemma as well. For instance, robots' hardware failure in taking the right path could be interpreted as "noise" in the communication between the two players. This was something that was not explicitly programmed, but it emerged in more than one occasion and it could not have happened in the virtual model. In this case, the fact that the simulation was carried out in the real world actually had a concrete and tangible impact on the study of the dilemma.

Moreover, one should not underestimate the value of robotics in the pedagogical sphere. Several aspects contributed to my academic growth

during the creation of my project thesis. The most relevant was due to the fact that programming robotic agents allowed me to obtain real feedbacks of what I was doing without layers of abstraction to obfuscate the dynamics of the interactions between the agents and the "world".

My thesis lays simultaneously in three fields: robotics for simulation models, economics and game theory. In fact, the model created departing from the Prisoner's Dilemma can be easily applied to many real-world situations in which a confrontation between two players is present. The NetLogo model can be a valuable tool for decision-makers in assessing an effective strategy (or mix of strategies) to use when facing an opponent through Prisoner's Dilemma like interaction.

In conclusion, we can state that the robotic simulation provides a valuable contribution to the virtual simulation under several aspects and in our case offers another important perspective on the Prisoner's Dilemma.

References

Arthur, B. W., Durlauf, S. N., & Lane, D. (1997). *The Economy as an Evolving Complex System II.* Reading: Addison-Wesley.

Axelrod, R. (1984). *The Evolution of Cooperation.* New York: Basic Books, Inc.

Axelrod, R. (2203). Advancing the Art of Simulation in the Social Sciences. *Japanese Journal for Management Information Systems*, 1-19.

Axtell, R. L. (2000). Why Agents? On the Varied Motivations for Agent Computing in the Social Sciences. *Center on Social and Economc Dynamics*.

Belpaeme, T., & Birk, A. (1997). On the Watch. *ISATA Conference*.

Birk, A. (1996). Learning to Survive. *5th European Workshop on Learning Robots*.

Birk, A. (2001). Learning to Trust. *Vrije Universiteit Brussels, Artificial Intelligence Laboratory*, 133-134.

Birk, A., & Wiernik, J. (1996). Behavioral AI Experiments and Economics. *12th Eurpean Conference on Artificial Intelligence*.

Birk, A., & Wiernik, J. (2002). An N-Player Prisoner's Dilemma in a Robotic Ecosystem. *Robotics and Autonomous Systems*, 223-233.

Blikstein, P., & Wilensky, U. (2006). A Technological Platform for Trans-Media Scientific Exploration. *ICLS '06 Proceedings of the 7th international conference on Learning sciences*.

Blikstein, P., & Wilensky, U. (2007). Bifocal Modeling: a Framework for Combining Computer Modeling, Robotics and Real-World Sensing. *American Educational Research Association*, 1-16.

Box, G. E. (1987). *Empirical Model-Building and Response Surfaces.* New York: John Wiley & Sons, Inc.

Brooks, R. A. (1990). Elephants Don't Play Chess. *Robotics and Autonomous Systems*, 3-15.

Comim, F. (1999). The Santa Fe Approach to Complexity: a Marshallian Evaluation. *Structural Change and Economic Dynamics*, 25-43.

Cowan, G. A., Pines, D., & Meltzer, D. E. (1994). *Complexity: Metaphors, Models & Reality.* Reading: Addison-Wesley.

Deulofeu, J. (2011). *Dilemma del Prigioniero e Strategie Dominanti. La Teoria dei Giochi.* Milano: RBA.

Eckhart, A. (2008). *Explaining Altruism. A Simulation-Based Approach and its Limits.* Frankfurt: Ontos Verlag.

Edmonds, B. (2012, 12 28). *Agent-Based Social Simulation*. Retrieved from The University of Manchester: http://www.methods.manchester.ac.uk/methods/abss/

Epstein, J. M. (2008). Why Model? *Journal of Artifical Societies and Social Simulation, 11*(4), 1-12.

Fischer, E. (2012, 12 20). *Race and Ethnicity (2000)*. Retrieved from http://www.flickr.com/photos/walkingsf/sets/72157624812674967/with/4981417821/

Friedman, M. (1953). The Methodology of Positive Economics. In M. Friedman, *Essays in Positive Economics* (pp. 3-43). Chicago: The University of Chicago Press.

Gallino, L. (1987). *L'Attore Sociale. Biologia, Cultura e Intelligenza Artificiale.* Torino: Einaudi.

Gallino, L., Borgna, P., Bulsei, G.-L., & Grimaldi, R. (1992). *Teoria dell'Attore e Processi Decisionali. Modelli Intelligenti per la Valutazione dell'Impatto Socio-Ambientale.* Milano: FrancoAngeli s.r.l.

Gilbert, N. (2008). *Agent-Based Models.* London: Sage Publications.

Gilbert, N., & Terna, P. (2000). How to Build and Use Agent-based Models in Social Science. *Mind & Society, I*(1), 57-72.

Gilbert, N., & Troitzsch, K. G. (2005). *Simulation for the Social Scientist.* Buckingham: Open University Press.

Grimaldi, B., Gandolfo, M., & Barone, L. (2011). *Spatial Finiteness, Epidemics and Climate Shocks.* Torino.

Hansen, J. C. (2009). *Lego Mindstorm NXT Power Programming. Robotics in C.* Manitoba: Variant Press.

Hindriks, J., & Myles, G. D. (2006). *Intermediate Public Economics.* Cambridge: MIT Press.

Holcombe, R. G. (2004). Government: Unnecessary but Inevitable. *The Indpendent Review. VIII(3)*, 325-342.

Holland, J. H. (1992). *Adaptation in Natural Artificial Systems.* Cambridge: MIT Press.

Janssen, M. A. (2009). Understanding Artificial Anasazi. *Journal of Artificial Societies and Social Simulation, 12*(4), 1-13.

Kuang, C. (2010, 09 20). *Infographic of the Day: How Segregated is Your City?* Retrieved from Fast Company: http://www.fastcompany.com/1690097/infographic-day-how-segregated-your-city

Metta, G., Sandini, G., Vernon, D., Natale, L., & Nori, F. (2008). The iCub Humanoid Robot: an Open Platform for Research in Embodied Cognition.

Negri, N., Miceli, R., Frigessi, D., Allasino, E., Ciafaloni, F., Ortona, G., . . . Grimaldi, R. (1995). *Atteggiamenti e Comportamenti verso gli Immigrati in Alcuni Ambienti Istituzionali.* Torino: Rosenberg & Sellier.

Nowack, M., & Highfield, R. (2011). *SuperCooperators. Beyond the Survival of Fittest. Why Cooperation, not Competition, is the Key to Life.* Edinburgh: Canongate.

Ostrom, T. (1988). Computer simulation: the third symbol system. *Journal of Experimental Social Psychology, 24*, 381-392.

Page, S. (2012, February 1). *Model Thinking*. Retrieved October 20, 2012, from Coursera: https://www.coursera.org/

Parisi, D., & Petrosino, G. (2010). Robots that have emotions. *International Society of Adaptive Behavior*, 453-468.

Parker, D. (2012, 10 06). *Multi-Bot Line Sensor*. Retrieved from NXT Programs: http://www.nxtprograms.com/NXT2/multi-bot/line/steps.html

Parker, D. (2012, 10 06). *Multi-Bot Touch Sensors*. Retrieved from NXT Programs: http://www.nxtprograms.com/NXT2/multi-bot/touch/steps.html

Parker, D. (2012, 10 06). *Multi-Bot Ultrasonic Sensor*. Retrieved from NXT Programs: http://www.nxtprograms.com/NXT2/multi-bot/ultrasonic/steps.html

Parker, D. (2012, 10 06). *Multi-Bot Vehicle*. Retrieved from NXT Pograms: http://www.nxtprograms.com/NXT2/multi-bot/vehicle/steps.html

Pindyck, R. S., & Rubinfeld, D. L. (2006). *Microeconomia.* Bologna: Zanichelli.

Robotics, Brain and Cognitive Sciences dept. at the Istituto Italiano di Tecnologia. (2012, 12 14). *iCub*. Retrieved from iCub: http://www.icub.org

Schelling, T. (1978). *Micromotives and Macrobehavior.* New York: Norton.

Steels, L. (1994). A Case Study in the Behavior-Oriented Design of Autonomous Agents. In *From animals to animats 3 proceedings of the Third International Conference on Simulation of Adaptive Behavior* (pp. 445-452). Cambridge: MIT Press.

Terna, P. (2000). Economic Experiments with Swarm: a Neural Network Approach to the Self-Development of Consistency in Agents' Behavior. In F. Luna, & B. Stefansson, *Economic Simulation in Swarm: Agent-Based Modeling and Object Oriented Programming* (pp. 73-103). Dordrecht and London: Kluwer Academic.

Terna, P., Boero, R., Morini, M., & Sonnessa, M. (2011). *Modelli per la Complessità. La Simulazione ad Agenti in Economia.* Bologna: Mulino.

Tesfatsion, L. (1995). How Economists Can Get Alife. In *Proc. of The Economy as an Evolving Complex System.* Addison-Wesley.

Tisue, S., & Wilensky, U. (2004). NetLogo: A Simple Environment for Modeling Complexity. *Center for Connected Learning and Computer-Based Modeling*.

Wilensky, U. (1999). *NetLogo 2.0.0 User Manual.* Northwestern University: Center for Connected Learning and Computer-Based Modeling.

Appendix A – NetLogo Code

```
globals [
   move-rule
   long-way-s?
   long-way-d?
   result-static
   result-dynamic

   c-static-payoff
   d-static-payoff
   tft-static-payoff
   r-static-payoff
   wsls-static-payoff

   c-dynamic-payoff
   d-dynamic-payoff
   tft-dynamic-payoff
   r-dynamic-payoff
   wsls-dynamic-payoff

   static-payoff-current-round
   dynamic-payoff-current-round

   num-static-cooperate-games
   num-static-defect-games
   num-static-tft-games
   num-static-random-games
   num-static-wsls-games

   num-dynamic-defect-games
   num-dynamic-cooperate-games
   num-dynamic-tft-games
   num-dynamic-wsls-games
   num-dynamic-random-games

   num-cooperate-games
   num-defect-games
   num-tft-games
   num-random-games
   num-wsls-games
   first-move-rule?

   static-previous-round
   dynamic-previous-round

   static-payoff
   dynamic-payoff

   num-games


   ]

breed [statics static]
breed [dynamics dynamic]
breed [corners corner]

statics-own [
  quadrant
  first-half?
  ]

dynamics-own [
  quadrant
  first-half?
  ]

;;;;;;;;;;;;;;;;;;;;;;
;;;Setup Procedures;;;
;;;;;;;;;;;;;;;;;;;;;;
```

```
to setup
  clear-all
  black-background
  make-corners
  make-red-path
  make-blue-path
  make-yellow-path
  make-fake-green-corner
  statics-setup
  dynamics-setup
  set move-rule 0
  set long-way-s? false
  set long-way-d? false
  set first-move-rule? true

  set c-static-payoff 0
  set d-static-payoff 0
  set tft-static-payoff 0
  set r-static-payoff 0
  set wsls-static-payoff 0

  set static-payoff-current-round 0
  set dynamic-payoff-current-round 0

  set c-dynamic-payoff 0
  set d-dynamic-payoff 0
  set tft-dynamic-payoff 0
  set r-dynamic-payoff 0
  set wsls-dynamic-payoff 0

  set num-static-cooperate-games 0
  set num-static-defect-games 0
  set num-static-tft-games 0
  set num-static-random-games 0
  set num-static-wsls-games 0

  set num-dynamic-defect-games 0
  set num-dynamic-cooperate-games 0
  set num-dynamic-tft-games 0
  set num-dynamic-wsls-games 0
  set num-dynamic-random-games 0

  set num-cooperate-games 0
  set num-defect-games 0
  set num-tft-games 0
  set num-random-games 0
  set num-wsls-games 0
  set static-previous-round 0
  set dynamic-previous-round 0
  set static-payoff 0
  set dynamic-payoff 0

  set num-games 0



  check-sum
  reset-ticks
end

to default-setup
  clear-output
  set static-cooperate 20
  set static-defect 20
  set static-random 20
  set static-tit-for-tat 20
  set static-win-stay-lose-shift 20

  set lock-static-cooperate? false
  set lock-static-defect? false
  set lock-static-random? false
  set lock-static-tit-for-tat? false
  set lock-static-win-stay-lose-shift? false

  set wsls-slider-static 50
```

```
  set dynamic-cooperate 20
  set dynamic-defect 20
  set dynamic-random 20
  set dynamic-tit-for-tat 20
  set dynamic-win-stay-lose-shift 20

  set lock-dynamic-cooperate? false
  set lock-dynamic-defect? false
  set lock-dynamic-random? false
  set lock-dynamic-tit-for-tat? false
  set lock-dynamic-win-stay-lose-shift? false

  set wsls-slider-dynamic 50
end

to setup-sliders
  let sum-locked 0
  let to-distribute 0
  let num-sliders-free 0
  let final-number 0

  if lock-static-cooperate? = true [ set sum-locked sum-locked + static-
cooperate ]
  if lock-static-defect? = true [ set sum-locked sum-locked + static-defect
]
  if lock-static-random? = true [ set sum-locked sum-locked + static-random
]
  if lock-static-tit-for-tat? = true [ set sum-locked sum-locked + static-
tit-for-tat ]
  if lock-static-win-stay-lose-shift? = true [ set sum-locked sum-locked +
static-win-stay-lose-shift ]

  set to-distribute (100 - sum-locked)

  if lock-static-cooperate? = false [ set num-sliders-free num-sliders-free
+ 1  ]
  if lock-static-defect? = false [ set num-sliders-free num-sliders-free + 1
]
  if lock-static-random? = false [ set num-sliders-free num-sliders-free + 1
]
  if lock-static-tit-for-tat? = false [ set num-sliders-free num-sliders-
free + 1 ]
  if lock-static-win-stay-lose-shift? = false [set num-sliders-free num-
sliders-free + 1  ]

  set final-number to-distribute / num-sliders-free
  if final-number < 0 [ set final-number 0 ]

  if lock-static-cooperate? = false [ set static-cooperate final-number ]
  if lock-static-defect? = false [ set static-defect final-number ]
  if lock-static-random? = false [ set static-random final-number ]
  if lock-static-tit-for-tat? = false [ set static-tit-for-tat final-number
]
  if lock-static-win-stay-lose-shift? = false [ set static-win-stay-lose-
shift final-number ]


  set sum-locked 0
  set to-distribute 0
  set num-sliders-free 0
  set final-number 0

  if lock-dynamic-cooperate? = true [ set sum-locked sum-locked + dynamic-
cooperate ]
  if lock-dynamic-defect? = true [ set sum-locked sum-locked + dynamic-
defect ]
  if lock-dynamic-random? = true [ set sum-locked sum-locked + dynamic-
random ]
  if lock-dynamic-tit-for-tat? = true [ set sum-locked sum-locked + dynamic-
tit-for-tat ]
  if lock-dynamic-win-stay-lose-shift? = true [ set sum-locked sum-locked +
dynamic-win-stay-lose-shift ]

  set to-distribute (100 - sum-locked)
```

```
  if lock-dynamic-cooperate? = false [ set num-sliders-free num-sliders-free
+ 1  ]
  if lock-dynamic-defect? = false [ set num-sliders-free num-sliders-free +
1 ]
  if lock-dynamic-random? = false [ set num-sliders-free num-sliders-free +
1 ]
  if lock-dynamic-tit-for-tat? = false [ set num-sliders-free num-sliders-
free + 1 ]
  if lock-dynamic-win-stay-lose-shift? = false [ set num-sliders-free num-
sliders-free + 1  ]


  set final-number to-distribute / num-sliders-free
  if final-number < 0 [ set final-number 0 ]

  if lock-dynamic-cooperate? = false [ set dynamic-cooperate final-number ]
  if lock-dynamic-defect? = false [ set dynamic-defect final-number ]
  if lock-dynamic-random? = false [ set dynamic-random final-number ]
  if lock-dynamic-tit-for-tat? = false [ set dynamic-tit-for-tat final-
number ]
  if lock-dynamic-win-stay-lose-shift? = false [ set dynamic-win-stay-lose-
shift final-number ]
end

to check-sum
  ifelse ((static-cooperate + static-defect + static-random + static-tit-
for-tat + static-win-stay-lose-shift) = 100) and ((dynamic-cooperate +
dynamic-defect + dynamic-random + dynamic-tit-for-tat + dynamic-win-stay-
lose-shift) = 100) [
    output-print "You can start the simulation"
    ][
    output-print "Error: the sum is not 100\nSimulation will be
blocked\nCheck your sliders and start again"
    ]
end


to statics-setup
  set-default-shape statics "nxt"
  create-statics 1
  ask statics [
    set first-half? true
    set size 7
    set color red


    ifelse random 2 = 0 [
      ifelse random 2 = 0 [set xcor 20] [set xcor -20]
      set ycor (random 38 - 19)
      ][
      ifelse random 2 = 0 [set ycor 20] [set ycor -20]
      set xcor (random 38 - 19)
      ]


    ifelse (ycor < 20) and (ycor > -20) [
      ifelse xcor > 0  [
        facexy 20 -20
        ][
        facexy -20 20
        ]
      ][
      ifelse ycor = 20 [
        facexy 20 20
        ][
        facexy -20 -20
        ]
      ]
  ]
end

to dynamics-setup
  set-default-shape dynamics "nxt"
  create-dynamics 1
  ask dynamics [
    set first-half? true
```

```
    set size 7
    set color blue

    ifelse random 2 = 0 [
      ifelse random 2 = 0 [set xcor 20] [set xcor -20]
      set ycor (random 38 - 19)
      ][
      ifelse random 2 = 0 [set ycor 20] [set ycor -20]
      set xcor (random 38 - 19)
      ]


    ifelse (ycor < 20) and (ycor > -20) [
      ifelse xcor > 0  [
        facexy 20 20
        ][
        facexy -20 -20
        ]
      ][
      ifelse ycor = 20 [
        facexy -20 20
        ][
        facexy 20 -20
        ]
      ]
  ]
end


to go
  move-statics
  move-dynamics
  if num-games = 1000 [stop]
  tick
end


to move-s [new-x new-y]
  ask statics [
    set xcor (xcor + new-x)
    set ycor (ycor + new-y)
  ]
end



to move-statics
    ask statics [
      if long-way-s? = false [
        ifelse move-rule != 2 [
        ifelse move-rule = 1 or move-rule = 3[
          while [pcolor != green] [fd -1]
          set long-way-s? true
          ifelse move-rule = 3 [
            ifelse first-move-rule? = true [
              set first-move-rule? false

              ][
              set move-rule 0
              set first-move-rule? true
              ]
          ][
          set move-rule 0
          ]

        ][
        fd 1
        ]
        ][
        fd 0
        ]
      ]


      if xcor >= 0 and ycor > 0 [set quadrant 1]
      if xcor < 0 and ycor >= 0 [set quadrant 2]
```

```
      if xcor <= 0 and ycor < 0 [set quadrant 3]
      if xcor > 0 and ycor <= 0 [set quadrant 4]

    if long-way-s? = true [
      ifelse first-half? = true [
      if quadrant = 1 [
        facexy  30   0
        move-s 0.5 -1
        ]
      if quadrant = 2 [
        facexy   0  30
        move-s 1 0.5
        ]
      if quadrant = 3 [
        facexy -30   0
        move-s -0.5 1
        ]
      if quadrant = 4 [
        facexy   0 -30
        move-s -1 -0.5
        ]

      if pcolor = yellow [set first-half? false]
      ] [

      if quadrant = 1 [
        facexy 20 20
        move-s 1 -0.5
        ]
      if quadrant = 2 [
        facexy -20 20
        move-s 0.5 1
        ]
      if quadrant = 3 [
        facexy -20 -20
        move-s -1 0.5
        ]
      if quadrant = 4 [
        facexy 20 -20
        move-s -0.5 -1
        ]
      ]
    ]


    if (pcolor = green)  [
      ifelse long-way-s? = false [rt 90] [
        if quadrant = 1 [facexy 20 -20]
        if quadrant = 2 [facexy 20 20]
        if quadrant = 3 [facexy -20 20]
        if quadrant = 4 [facexy -20 -20]
      ]
      set first-half? true
      set long-way-s? false
      ]
    ]
end

to play-a-round
  set result-static random 100
  set result-dynamic random 100
  ifelse result-static < static-cooperate [ set result-static 1 ]
     [ ifelse result-static < (static-cooperate + static-defect) [ set
result-static 0 ]
          [ ifelse result-static < (static-cooperate + static-defect +
static-random) [ set result-static 3 ]
                [ ifelse result-static < ( static-cooperate + static-defect +
static-random + static-tit-for-tat) [ set result-static 2 ]
                     [ set result-static 4 ]
                ]
          ]
     ]

  ifelse result-dynamic < dynamic-cooperate [ set result-dynamic 1 ]
     [ ifelse result-dynamic < (dynamic-cooperate + dynamic-defect) [ set
result-dynamic 0 ]
```

```
          [ ifelse result-dynamic < (dynamic-cooperate + dynamic-defect +
dynamic-random) [ set result-dynamic 3 ]
              [ ifelse result-dynamic < (dynamic-cooperate + dynamic-defect
+ dynamic-random + dynamic-tit-for-tat) [ set result-dynamic 2 ]
                [ set result-dynamic 4 ]
              ]
          ]
     ]


  set num-games num-games + 1
  let static-strategy 0
  let dynamic-strategy 0
  if result-static = 0 [ set static-strategy defect 0]
  if result-static = 1 [ set static-strategy cooperate 0]
  if result-static = 2 [ set static-strategy tit-for-tat 0 ]
  if result-static = 3 [ set static-strategy random-strategy 0]
  if result-static = 4 [ set static-strategy win-stay-lose-shift 0]

  if result-dynamic = 0 [ set dynamic-strategy defect 1]
  if result-dynamic = 1 [ set dynamic-strategy cooperate 1]
  if result-dynamic = 2 [ set dynamic-strategy tit-for-tat 1]
  if result-dynamic = 3 [ set dynamic-strategy random-strategy 1]
  if result-dynamic = 4 [ set dynamic-strategy win-stay-lose-shift 1]

  calculate-payoff static-strategy dynamic-strategy result-static result-
dynamic


  set-move-rule static-strategy dynamic-strategy

  let translation-static ""
  if result-static = 0 [ set translation-static "Defect     "]
  if result-static = 1 [ set translation-static "Cooperate  "]
  if result-static = 2 [ set translation-static "Tit-for-tat"]
  if result-static = 3 [ set translation-static "Random     "]
  if result-static = 4 [ set translation-static "WSLS       "]

  let translation-dynamic ""
  if result-dynamic = 0 [ set translation-dynamic "Defect     "]
  if result-dynamic = 1 [ set translation-dynamic "Cooperate  "]
  if result-dynamic = 2 [ set translation-dynamic "Tit-for-tat"]
  if result-dynamic = 3 [ set translation-dynamic "Random     "]
  if result-dynamic = 4 [ set translation-dynamic "WSLS       "]

  let translate-static-outcome ""
  if static-strategy = 0 [ set translate-static-outcome " C"]
  if static-strategy = 1 [ set translate-static-outcome " D"]

  let translate-dynamic-outcome ""
  if dynamic-strategy = 0 [ set translate-dynamic-outcome " C"]
  if dynamic-strategy = 1 [ set translate-dynamic-outcome " D"]

  output-type "GAME #" output-type num-games output-type "    PLAYER A: "
output-type translation-static output-type "  DECISION:" output-type
translate-static-outcome output-type "  PAYOFF: " output-type static-payoff-
current-round output-type "      PLAYER B: " output-type translation-dynamic
output-type "  DECISION:" output-type translate-dynamic-outcome output-type
"  PAYOFF: " output-type dynamic-payoff-current-round output-print ""

end

to-report cooperate  [player]
  ifelse player = 0 [
    set num-static-cooperate-games (num-static-cooperate-games + 1)
  ][
  set num-dynamic-cooperate-games (num-dynamic-cooperate-games + 1)
  ]
  report 0
end

to-report defect [player]
  ifelse player = 0 [
    set num-static-defect-games (num-static-defect-games + 1)
  ][
  set num-dynamic-defect-games (num-dynamic-defect-games + 1)
```

```
  ]
  report 1
end

to-report random-strategy [player]
  ifelse player = 0 [
    set num-static-random-games (num-static-random-games + 1)
  ][
  set num-dynamic-random-games (num-dynamic-random-games + 1)
  ]
  ifelse random-float 1 < 0.5 [
    report 0
    ][
    report 1
    ]
end

to-report tit-for-tat [ player ]
  ifelse player = 0 [
    set num-static-tft-games (num-static-tft-games + 1)
  ][
  set num-dynamic-tft-games (num-dynamic-tft-games + 1)
  ]

  let strategy 0
  ifelse player = 0 [
    set strategy dynamic-previous-round
    ][
    set strategy static-previous-round
    ]
  report strategy
end

to-report win-stay-lose-shift [ player ] ; player 0 = static     player 1 =
dynamic

  let strategy 0
  ifelse player = 0 [
    set num-static-wsls-games (num-static-wsls-games + 1 )
  ][
  set num-dynamic-wsls-games (num-dynamic-wsls-games + 1)
  ]
  ifelse (dynamic-previous-round = 0) and (static-previous-round = 0) [
    set strategy 0
    ][
    ifelse (dynamic-previous-round = 1) and (static-previous-round = 1) [
           ifelse player = 0
           [ ifelse random 101 < wsls-slider-static [
               set strategy 0
               ][
               set strategy 1
               ]
            ][
            ifelse random 101 < wsls-slider-dynamic [
              set strategy 0
              ][
              set strategy 1
              ]
              ]
            ][
            set strategy 1
            ]
       ]
   report strategy
end


to calculate-payoff [ _static-strategy _dynamic-strategy _result-static
_result-dynamic ]

  ifelse _static-strategy = 0 [
    ifelse _dynamic-strategy = 0 [
      set-payoffs _result-static 4 0
      set-payoffs _result-dynamic 4 1
      set static-payoff-current-round 4
      set dynamic-payoff-current-round 4
```

```
      ][
      set-payoffs _result-static 0 0
      set-payoffs _result-dynamic 5 1
      set static-payoff-current-round 0
      set dynamic-payoff-current-round 5
      ]
    ][
    ifelse _dynamic-strategy = 0 [
        set-payoffs _result-static 5 0
        set-payoffs _result-dynamic 0 1
        set static-payoff-current-round 5
        set dynamic-payoff-current-round 0
        ][
        set-payoffs _result-static 2 0
        set-payoffs _result-dynamic 2 1
        set static-payoff-current-round 2
        set dynamic-payoff-current-round 2
      ]
  ]
end


to set-payoffs [strategy value player]
  if strategy = 0 [
    ifelse player = 0 [
      set d-static-payoff (d-static-payoff + value)
    ][
      set d-dynamic-payoff (d-dynamic-payoff + value)
      ]
  ]
  if strategy = 1 [
    ifelse player = 0 [
      set c-static-payoff (c-static-payoff + value)
    ][
    set c-dynamic-payoff (c-dynamic-payoff + value)
    ]
  ]
  if strategy = 2 [
    ifelse player = 0 [
      set tft-static-payoff (tft-static-payoff + value)
    ][
    set tft-dynamic-payoff (tft-dynamic-payoff + value)
    ]
  ]
  if strategy = 3 [
    ifelse player = 0 [
      set r-static-payoff (r-static-payoff + value)
    ][
    set r-dynamic-payoff (r-dynamic-payoff + value)
    ]
  ]
  if strategy = 4 [
    ifelse player = 0 [
      set wsls-static-payoff (wsls-static-payoff + value)
    ][
    set wsls-dynamic-payoff (wsls-dynamic-payoff + value)
    ]
  ]
end


to set-move-rule [strategy1 strategy2]
  ifelse strategy1 = 0 and strategy2 = 0 [
    ifelse random-float 1 < 0.5 [
      set move-rule 1
      set static-previous-round 0
      set dynamic-previous-round 0
      set static-payoff static-payoff + 4
      set dynamic-payoff dynamic-payoff + 4
      ] [
      set move-rule 2
      set static-previous-round 0
      set dynamic-previous-round 0
      set static-payoff static-payoff + 4
      set dynamic-payoff dynamic-payoff + 4
      ]
```

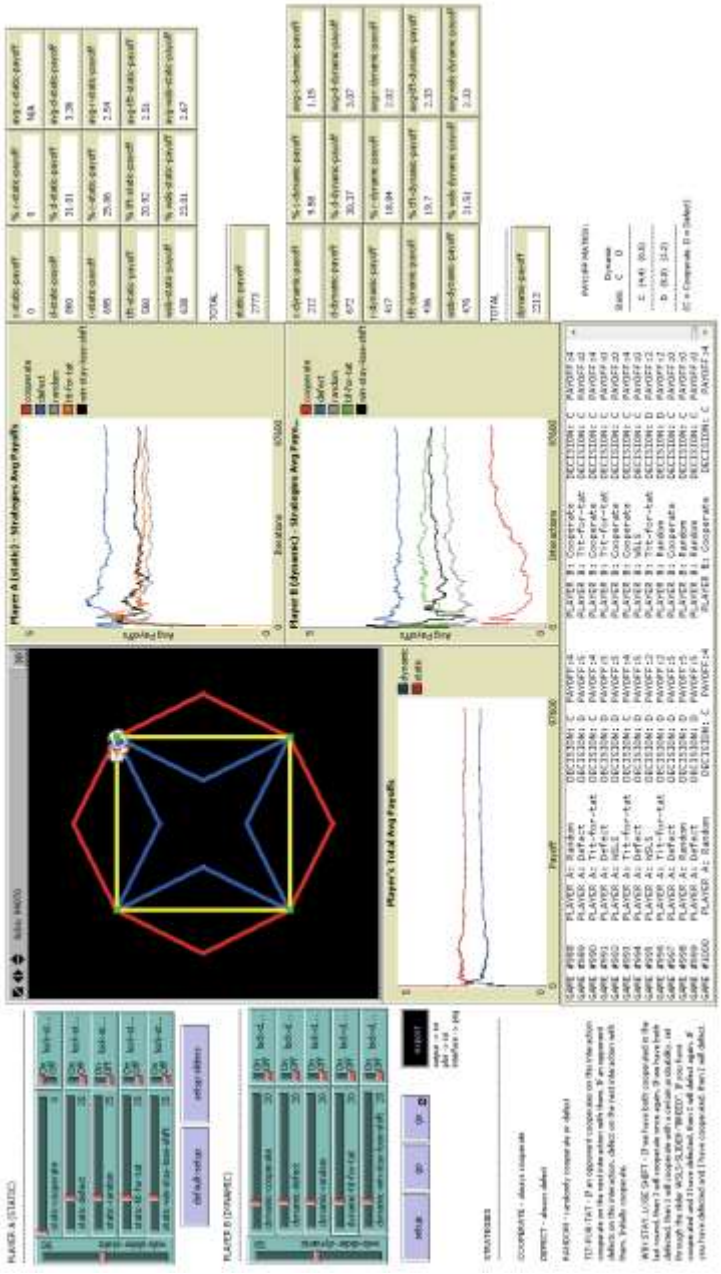
```
  ][

  ifelse strategy1 = strategy2 [
    set move-rule 3
    set static-previous-round 1
    set dynamic-previous-round 1
    set static-payoff static-payoff + 2
    set dynamic-payoff dynamic-payoff + 2
    ][
    ifelse strategy1 = 0 [
      set move-rule 1
      set static-previous-round 0
      set dynamic-previous-round 1
      set static-payoff static-payoff + 0
      set dynamic-payoff dynamic-payoff + 5
    ] [
      set move-rule 2
      set static-previous-round 1
      set dynamic-previous-round 0
      set static-payoff static-payoff + 5
      set dynamic-payoff dynamic-payoff + 0
      ]
    ]
  ]
end



to move-d [new-x new-y]
  ask dynamics [
    set xcor (xcor + new-x)
    set ycor (ycor + new-y)
  ]
end


to move-dynamics
    ask dynamics [
      if ((count statics in-radius 1 = 1) and (long-way-d? = false) and
(long-way-s? = false)) [play-a-round]
      if long-way-d? = false [
        ifelse move-rule != 1 [
        ifelse move-rule = 2 or move-rule = 3 [
          while [pcolor != green] [fd -1]
          set long-way-d? true
           ifelse move-rule = 3 [
            ifelse first-move-rule? = true [
              set first-move-rule? false


              ][
              set move-rule 0
              set first-move-rule? true
              ]
          ][
          set move-rule 0
          ]
        ][

        fd 1
        ]
      ][
      fd 0
      ]
      ]


      if xcor > 0 and ycor >= 0 [set quadrant 1]
      if xcor <= 0 and ycor > 0 [set quadrant 2]
      if xcor < 0 and ycor <= 0 [set quadrant 3]
      if xcor >= 0 and ycor < 0 [set quadrant 4]

     if long-way-d? = true [
      ifelse first-half? = true [
      if quadrant = 1 [
        facexy   0    10
```

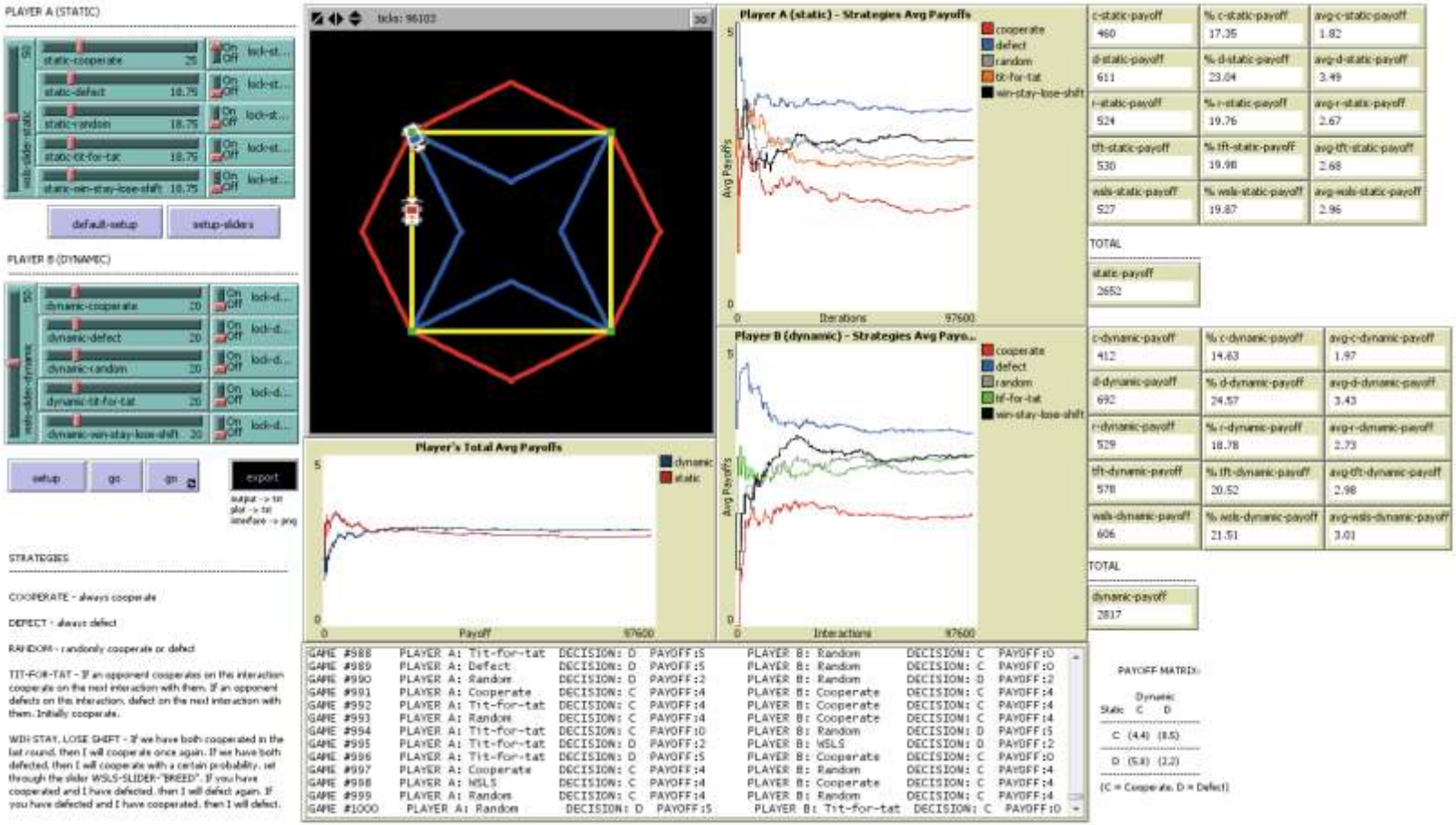
```
        move-d -1 -0.5
        ]
      if quadrant = 2 [
        facexy  -10    0
        move-d 0.5 -1
        ]
      if quadrant = 3 [
        facexy   0   -10
        move-d 1 0.5
        ]
      if quadrant = 4 [
        facexy   10    0
        move-d -0.5 1
        ]

      if pcolor = yellow [
        set first-half? false

        ]
      ] [
      if quadrant = 1 [
        facexy 20 20
        move-d 0.5 1
        ]
      if quadrant = 2 [
        facexy -20 20
        move-d -1 0.5
        ]
      if quadrant = 3 [
        facexy -20 -20
        move-d -0.5 -1
        ]
      if quadrant = 4 [
        facexy 20 -20
        move-d 1 -0.5
        ]
      ]
    ]

    if (pcolor = green)  [
      ifelse long-way-d? = false [lt 90] [
        if quadrant = 1 [facexy -20  20]
        if quadrant = 2 [facexy -20 -20]
        if quadrant = 3 [facexy  20 -20]
        if quadrant = 4 [facexy  20  20]
      ]
      set first-half? true
      set long-way-d? false
      ]

    ]
end

to export
   let newname ""

   if (static-cooperate) = (static-defect) and (static-cooperate) = (static-
random) and (static-cooperate) = (static-tit-for-tat)
[set newname "wsls______"]
   if (static-cooperate) = (static-defect) and (static-cooperate) = (static-
random) and (static-cooperate) = (static-win-stay-lose-shift)
[set newname "tft_______"]
   if (static-cooperate) = (static-defect) and (static-cooperate) = (static-
win-stay-lose-shift) and (static-cooperate) = (static-tit-for-tat)
[set newname "random____"]
   if (static-cooperate) = (static-random) and (static-cooperate) = (static-
win-stay-lose-shift) and (static-cooperate) = (static-tit-for-tat)
[set newname "defect_____"]
   if (static-random) = (static-defect) and (static-random) = (static-win-
stay-lose-shift) and (static-random) = (static-tit-for-tat)
[set newname "cooperate_"]
   if ((static-cooperate) = (static-defect)) and ((static-cooperate) =
(static-tit-for-tat)) and ((static-cooperate) = (static-win-stay-lose-shift)
and ((static-cooperate) = (static-random))) [set newname "static_____"]
```

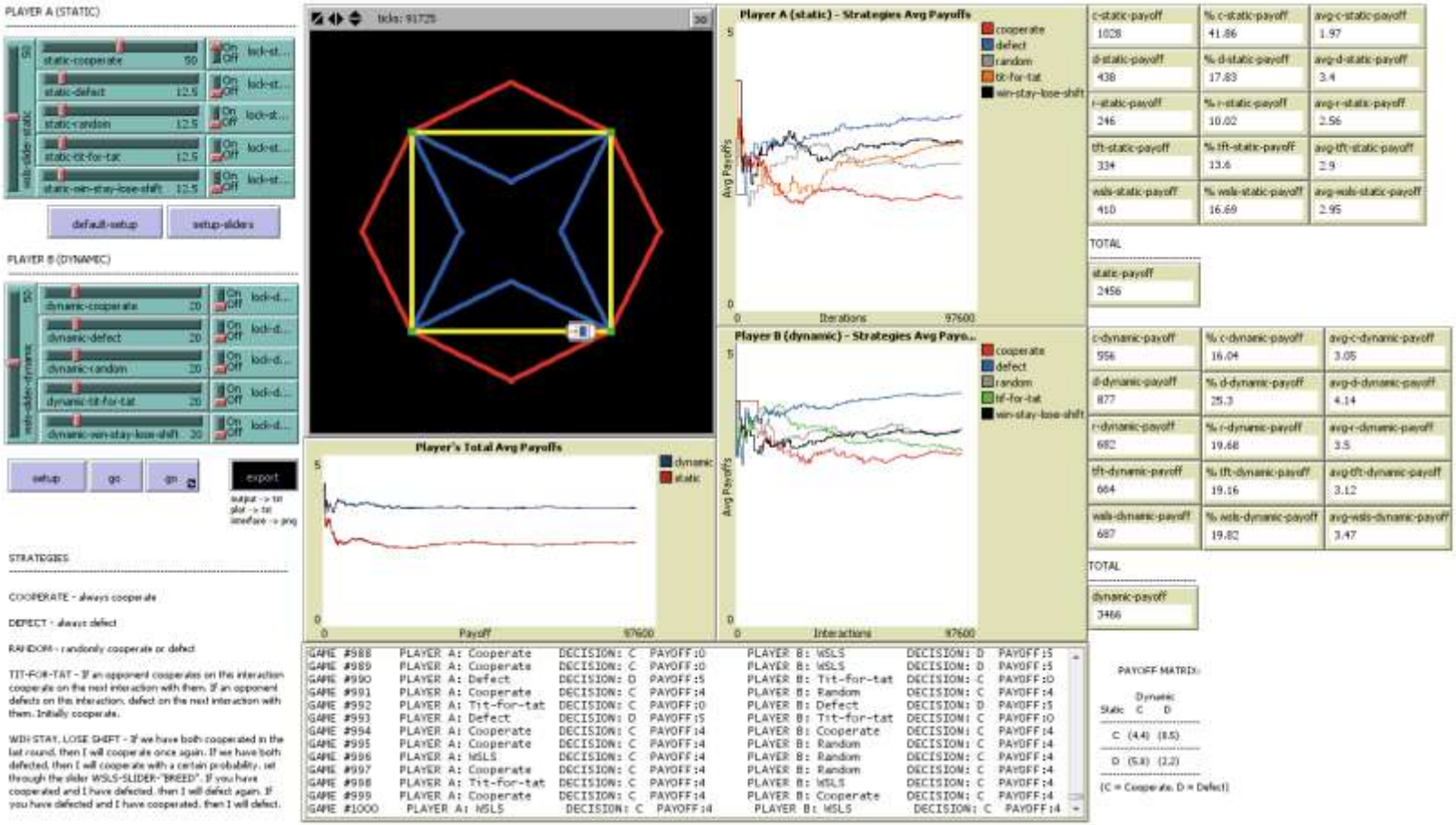
```
   export-output (word  newname (static-cooperate)"_"(static-
defect)"_"(static-random)"_"(static-tit-for-tat)"_"(static-win-stay-lose-
shift) "_output.txt")
   export-all-plots (word newname (static-cooperate)"_"(static-
defect)"_"(static-random)"_"(static-tit-for-tat)"_"(static-win-stay-lose-
shift) "_plot.txt")
   export-interface (word newname (static-cooperate)"_"(static-
defect)"_"(static-random)"_"(static-tit-for-tat)"_"(static-win-stay-lose-
shift) "_interface.png")
end

to black-background
  ask patches [set pcolor black]
end

to make-corners
  ask patch -20 -20 [set pcolor green]
  ask patch  20  20 [set pcolor green]
  ask patch -20  20 [set pcolor green]
  ask patch  20 -20 [set pcolor green]

  ask patch   0  30 [set pcolor yellow]
  ask patch  30   0 [set pcolor yellow]
  ask patch   0 -30 [set pcolor yellow]
  ask patch -30   0 [set pcolor yellow]

  ask patch   0  10 [set pcolor yellow]
  ask patch  10   0 [set pcolor yellow]
  ask patch   0 -10 [set pcolor yellow]
  ask patch -10   0 [set pcolor yellow]
end

to make-red-path
  crt 1
  ask turtles [
    set color red
    set size 1
    setxy -20 20
    pen-down
    set pen-size 5
    facexy 0 30
    fd 22.36067977
    facexy 20 20
    fd 22.36067977
    facexy 30 0
    fd 22.36067977
    facexy 20 -20
    fd 22.360679771
    facexy 0 -30
    fd 22.36067977
    facexy -20 -20
    fd 22.36067977
    facexy -30 0
    fd 22.36067977
    facexy -20 20
    fd 22.36067977
    die
  ]
end

to make-blue-path
  crt 1
  ask turtles [
    set color blue
    set size 1
    setxy -20 20
    pen-down
    set pen-size 5
    facexy 0 10
    fd 22.36067977
    facexy 20 20
    fd 22.36067977
    facexy 10 0
    fd 22.36067977
    facexy 20 -20
    fd 22.360679771
```

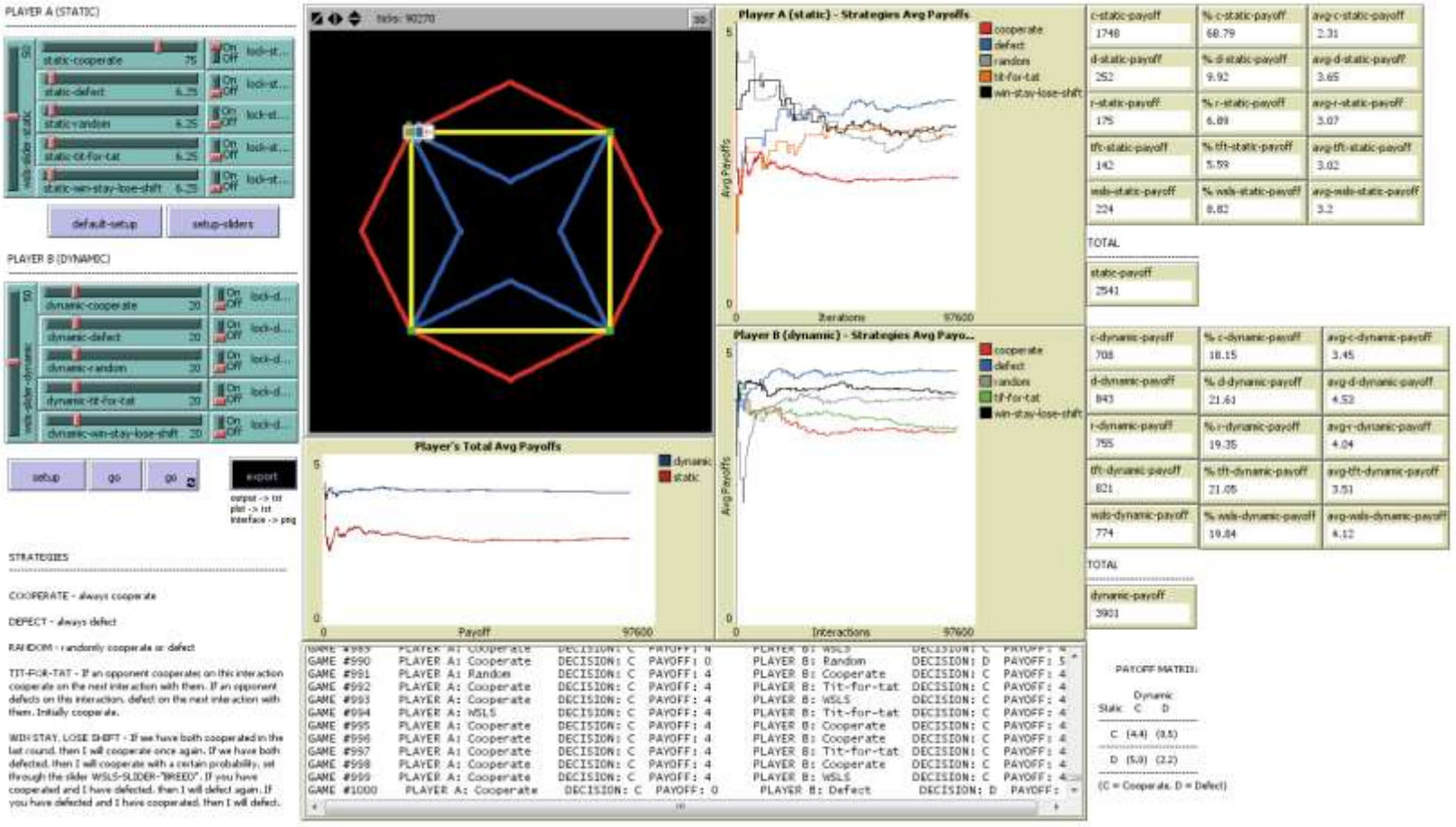
```
    facexy 0 -10
    fd 22.36067977
    facexy -20 -20
    fd 22.36067977
    facexy -10 0
    fd 22.36067977
    facexy -20 20
    fd 22.36067977
    die
  ]
end

to make-yellow-path
  crt 1
  ask turtles [
    set color yellow
    set size 1
    setxy -20 20
    pen-down
    set pen-size 5
    facexy 20 20
    fd 40
    facexy 20 -20
    fd 40
    facexy -20 -20
    fd 40
    facexy -20 20
    fd 40
    die
   ]
end

to make-fake-green-corner
  set-default-shape corners "square"
  create-corners 1[
  set xcor -20
  set ycor -20
  set color green
  set size 2
  ]
  create-corners 1 [
  set xcor 20
  set ycor 20
  set color green
  set size 2
  ]
  create-corners 1 [
  set xcor -20
  set ycor 20
  set color green
  set size 2
  ]
  create-corners 1 [
  set xcor 20
  set ycor -20
  set color green
  set size 2
  ]
end
```

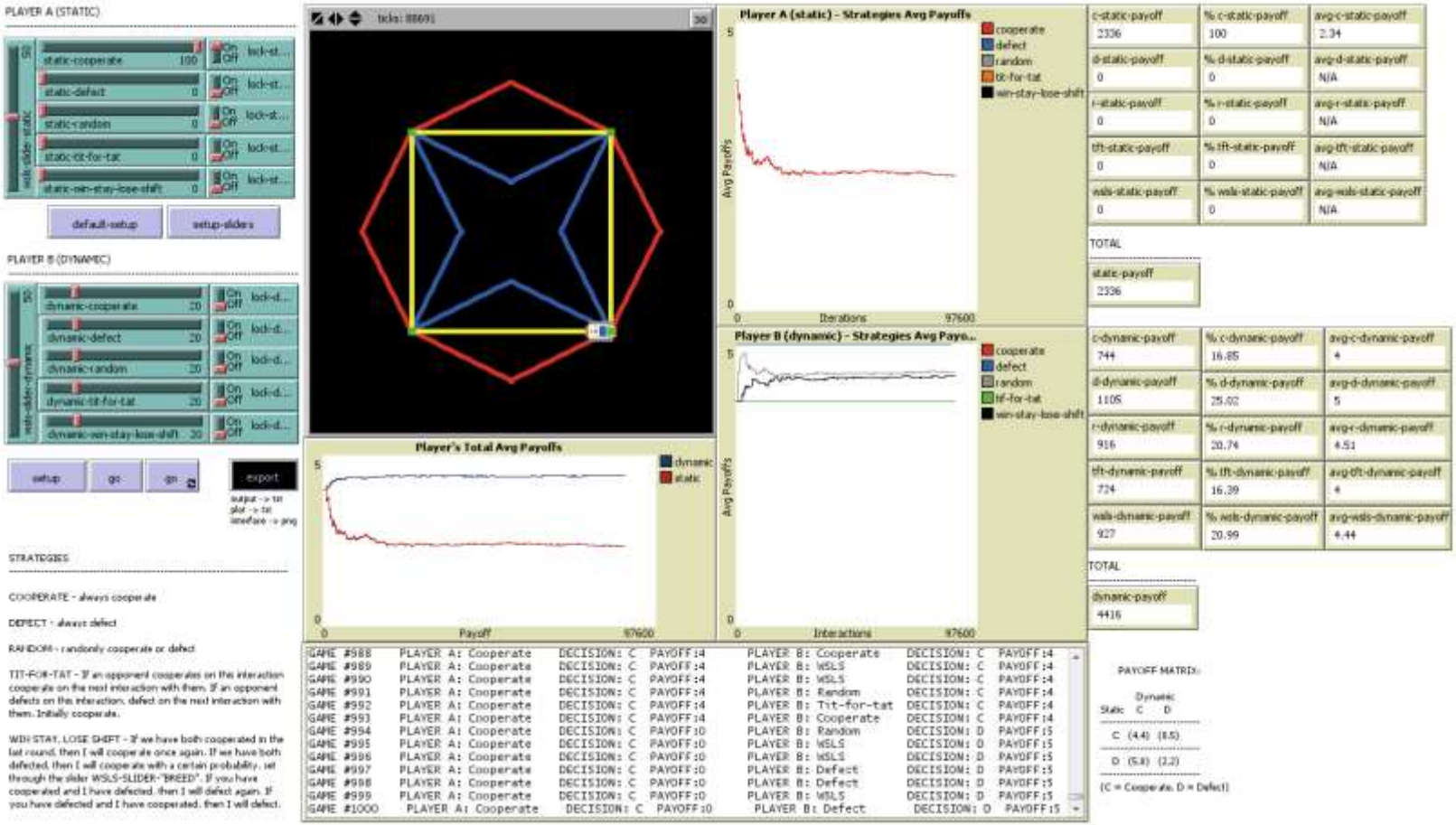
Appendix B – Simulation Interfaces

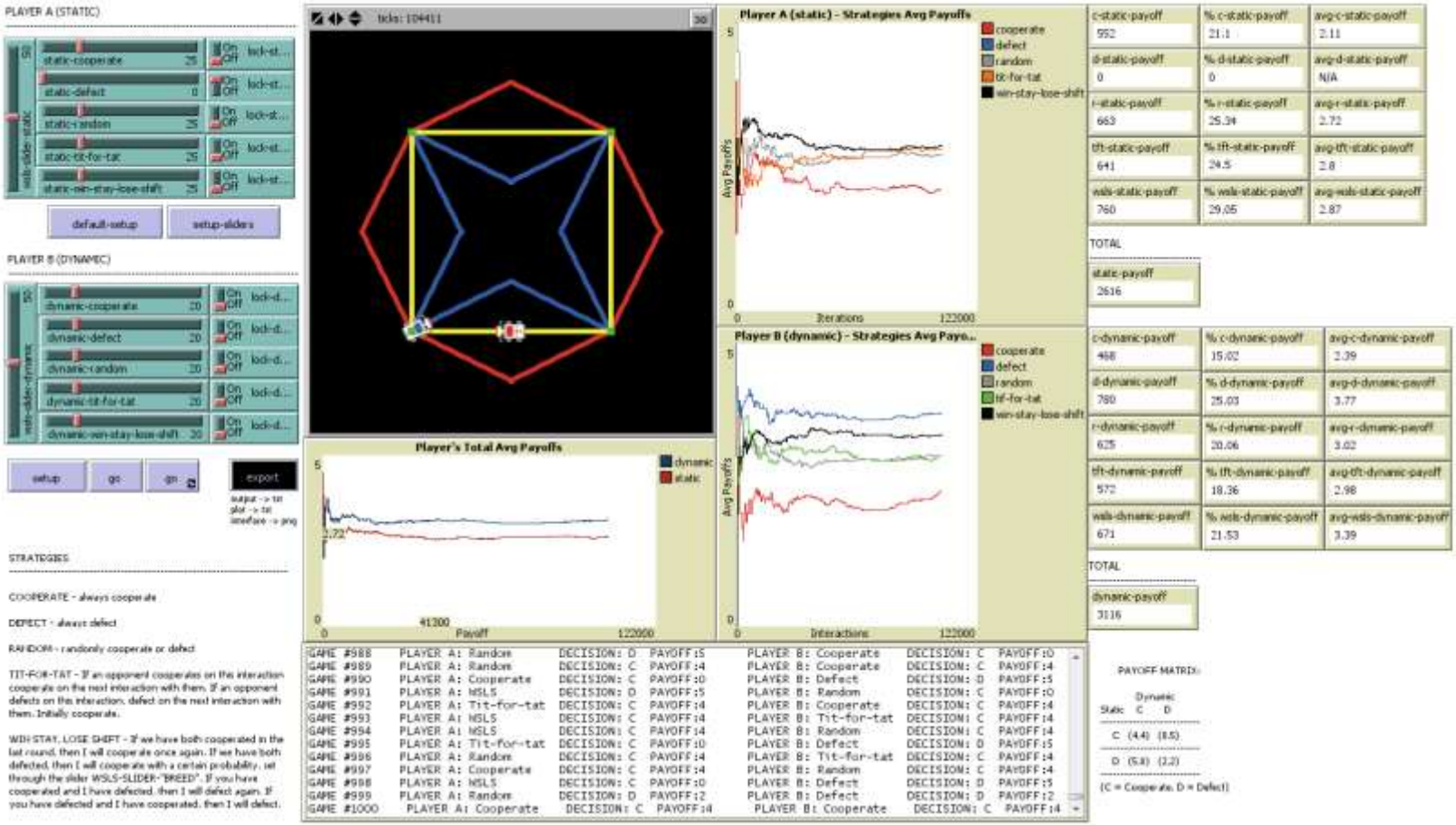


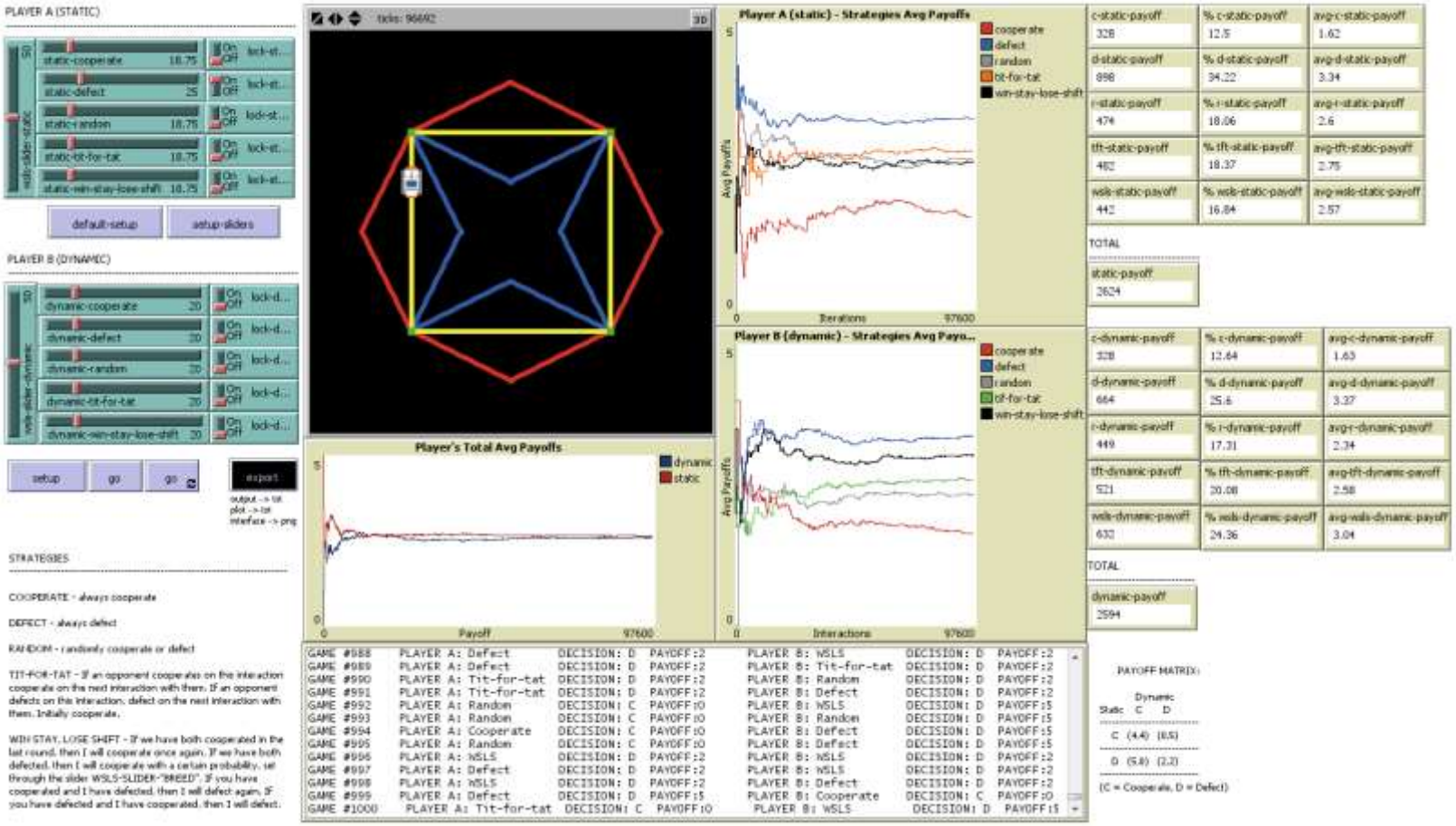

PLAYER A (STATIC)
static-cooperate 25
static-defect 18.75
static-random 18.75
static-tit-for-tat 18.75
static-win-stay-lose-shift 18.75
default-setup
setup-sliders
PLAYER B (DYNAMIC)
dynamic-cooperate 20
dynamic-defect 20
dynamic-random 20
dynamic-tit-for-tat 20
dynamic-win-stay-lose-shift 20
setup
go
export
STRATEGIES
COOPERATE - always cooperate
DEFECT - always defect
RANDOM - randomly cooperate or defect
TIT-FOR-TAT - If an opponent cooperates on the interaction cooperate on the next interaction with them. If an opponent defects on the interaction, defect on the next interaction with them. Initially cooperate.
WIN-STAY, LOSE SHIFT - If we have both cooperated in the last round, then I will cooperate once again. If we have both defected, then I will cooperate with a certain probability, set through the slider WSLS-SLIDER-"BREED". If you have cooperated and I have defected, then I will defect again. If you have defected and I have cooperated, then I will defect.
ticks: 96103
Player A (static) - Strategies Avg Payoffs
cooperate
defect
random
tit-for-tat
win-stay-lose-shift
Avg Payoffs
Iterations
97600
Player B (dynamic) - Strategies Avg Payo...
Player's Total Avg Payoffs
dynamic
static
Payoff
Interactions
c-static-payoff 460
% c-static-payoff 17.35
avg-c-static-payoff 1.82
d-static-payoff 611
% d-static-payoff 23.04
avg-d-static-payoff 3.49
r-static-payoff 524
% r-static-payoff 19.76
avg-r-static-payoff 2.67
tft-static-payoff 530
% tft-static-payoff 19.98
avg-tft-static-payoff 2.68
wsls-static-payoff 527
% wsls-static-payoff 19.87
avg-wsls-static-payoff 2.96
TOTAL
static-payoff 2652
c-dynamic-payoff 412
% c-dynamic-payoff 14.63
avg-c-dynamic-payoff 1.97
d-dynamic-payoff 692
% d-dynamic-payoff 24.57
avg-d-dynamic-payoff 3.43
r-dynamic-payoff 529
% r-dynamic-payoff 18.78
avg-r-dynamic-payoff 2.73
tft-dynamic-payoff 578
% tft-dynamic-payoff 20.52
avg-tft-dynamic-payoff 2.98
wsls-dynamic-payoff 606
% wsls-dynamic-payoff 21.51
avg-wsls-dynamic-payoff 3.01
TOTAL
dynamic-payoff 2817
GAME #988 PLAYER A: Tit-for-tat DECISION: D PAYOFF:5 PLAYER B: Random DECISION: C PAYOFF:0
GAME #989 PLAYER A: Defect DECISION: D PAYOFF:5 PLAYER B: Random DECISION: C PAYOFF:0
GAME #990 PLAYER A: Random DECISION: D PAYOFF:2 PLAYER B: Random DECISION: D PAYOFF:2
GAME #991 PLAYER A: Cooperate DECISION: C PAYOFF:4 PLAYER B: Cooperate DECISION: C PAYOFF:4
GAME #992 PLAYER A: Tit-for-tat DECISION: C PAYOFF:4 PLAYER B: Cooperate DECISION: C PAYOFF:4
GAME #993 PLAYER A: Random DECISION: C PAYOFF:4 PLAYER B: Cooperate DECISION: C PAYOFF:4
GAME #994 PLAYER A: Tit-for-tat DECISION: C PAYOFF:0 PLAYER B: Random DECISION: D PAYOFF:5
GAME #995 PLAYER A: Tit-for-tat DECISION: D PAYOFF:2 PLAYER B: WSLS DECISION: D PAYOFF:2
GAME #996 PLAYER A: Tit-for-tat DECISION: D PAYOFF:5 PLAYER B: Cooperate DECISION: C PAYOFF:0
GAME #997 PLAYER A: Cooperate DECISION: C PAYOFF:4 PLAYER B: Random DECISION: C PAYOFF:4
GAME #998 PLAYER A: WSLS DECISION: C PAYOFF:4 PLAYER B: Cooperate DECISION: C PAYOFF:4
GAME #999 PLAYER A: Random DECISION: C PAYOFF:4 PLAYER B: Random DECISION: C PAYOFF:4
GAME #1000 PLAYER A: Random DECISION: D PAYOFF:5 PLAYER B: Tit-for-tat DECISION: C PAYOFF:0
PAYOFF MATRIX:
Dynamic
Static C D
C (4,4) (0,5)
D (5,0) (2,2)
(C = Cooperate, D = Defect)

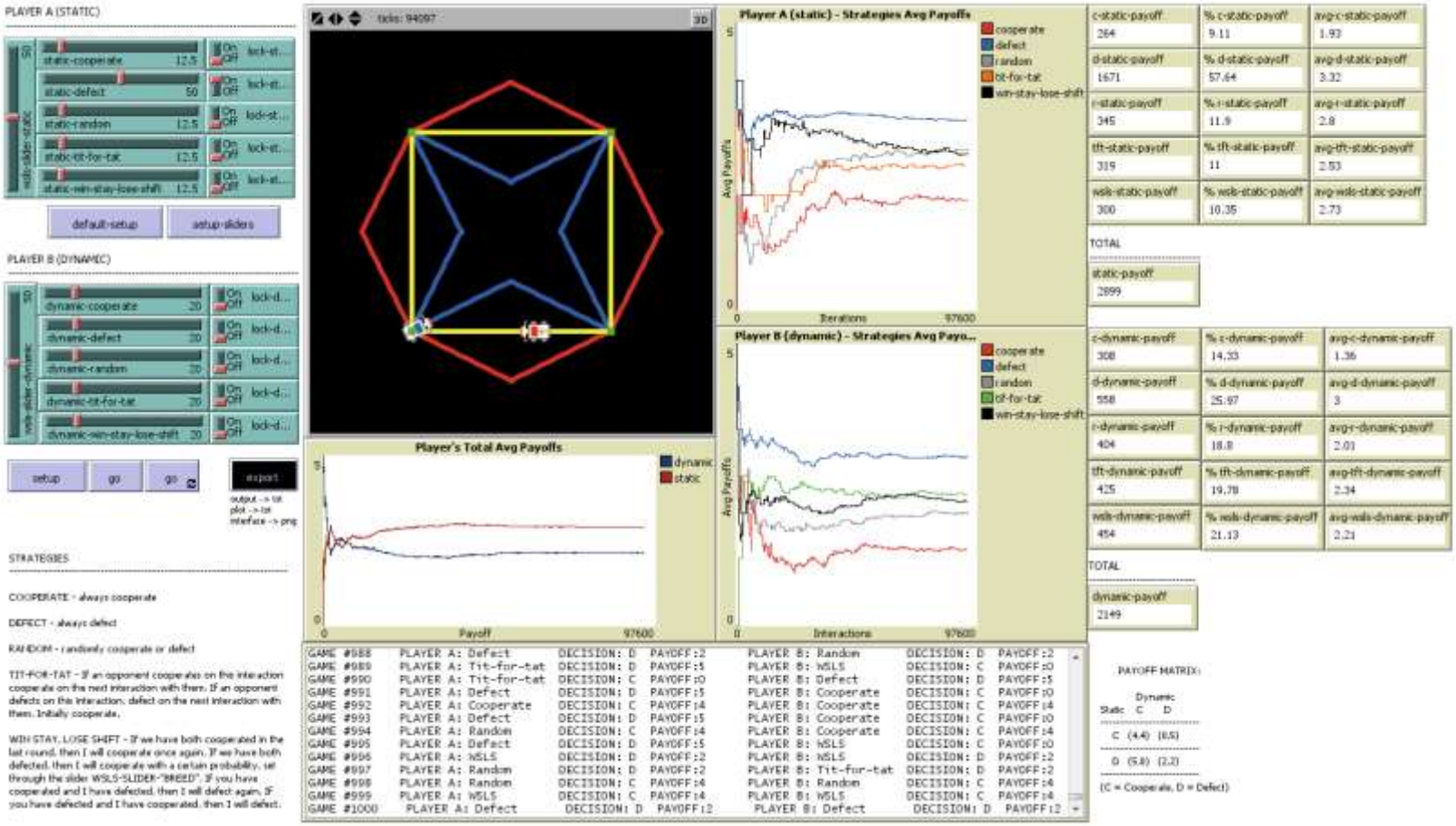

PLAYER A (STATIC)
static-cooperate 50
static-defect 12.5
static-random 12.5
static-tit-for-tat 12.5
static-win-stay-lose-shift 12.5
default-setup
setup-sliders
PLAYER B (DYNAMIC)
dynamic-cooperate 20
dynamic-defect 20
dynamic-random 20
dynamic-tit-for-tat 20
dynamic-win-stay-lose-shift 20
setup
go
export
STRATEGIES
COOPERATE - always cooperate
DEFECT - always defect
RANDOM - randomly cooperate or defect
TIT-FOR-TAT - If an opponent cooperates on the interaction cooperate on the next interaction with them. If an opponent defects on the interaction, defect on the next interaction with them. Initially cooperate.
WIN-STAY, LOSE-SHIFT - If we have both cooperated in the last round, then I will cooperate once again. If we have both defected, then I will cooperate with a certain probability, set through the slider WSLS-SLIDER-"BREED". If you have cooperated and I have defected, then I will defect again. If you have defected and I have cooperated, then I will defect.
ticks: 91725
Player's Total Avg Payoffs
dynamic
static
Payoff
Player A (static) - Strategies Avg Payoffs
cooperate
defect
random
tit-for-tat
win-stay-lose-shift
Avg Payoffs
Iterations
Player B (dynamic) - Strategies Avg Payo...
Interactions
c-static-payoff 1028
% c-static-payoff 41.86
avg-c-static-payoff 1.97
d-static-payoff 438
% d-static-payoff 17.83
avg-d-static-payoff 3.4
r-static-payoff 246
% r-static-payoff 10.02
avg-r-static-payoff 2.56
tft-static-payoff 334
% tft-static-payoff 13.6
avg-tft-static-payoff 2.9
wsls-static-payoff 410
% wsls-static-payoff 16.69
avg-wsls-static-payoff 2.95
TOTAL
static-payoff 2456
c-dynamic-payoff 956
% c-dynamic-payoff 16.04
avg-c-dynamic-payoff 3.05
d-dynamic-payoff 877
% d-dynamic-payoff 25.3
avg-d-dynamic-payoff 4.14
r-dynamic-payoff 682
% r-dynamic-payoff 19.68
avg-r-dynamic-payoff 3.5
tft-dynamic-payoff 664
% tft-dynamic-payoff 19.16
avg-tft-dynamic-payoff 3.12
wsls-dynamic-payoff 687
% wsls-dynamic-payoff 19.82
avg-wsls-dynamic-payoff 3.47
TOTAL
dynamic-payoff 3466
PAYOFF MATRIX:
Dynamic
Static C D
C (4,4) (8,5)
D (5,8) (2,2)
(C = Cooperate, D = Defect)
GAME #988 PLAYER A: Cooperate DECISION: C PAYOFF:10 PLAYER B: WSLS DECISION: D PAYOFF:5
GAME #989 PLAYER A: Cooperate DECISION: C PAYOFF:10 PLAYER B: WSLS DECISION: D PAYOFF:5
GAME #990 PLAYER A: Defect DECISION: D PAYOFF:5 PLAYER B: Tit-for-tat DECISION: C PAYOFF:10
GAME #991 PLAYER A: Cooperate DECISION: C PAYOFF:4 PLAYER B: Random DECISION: C PAYOFF:4
GAME #992 PLAYER A: Tit-for-tat DECISION: C PAYOFF:10 PLAYER B: Defect DECISION: D PAYOFF:5
GAME #993 PLAYER A: Defect DECISION: D PAYOFF:5 PLAYER B: Tit-for-tat DECISION: C PAYOFF:10
GAME #994 PLAYER A: Cooperate DECISION: C PAYOFF:4 PLAYER B: Cooperate DECISION: C PAYOFF:4
GAME #995 PLAYER A: Cooperate DECISION: C PAYOFF:4 PLAYER B: Random DECISION: C PAYOFF:4
GAME #996 PLAYER A: WSLS DECISION: C PAYOFF:4 PLAYER B: Random DECISION: C PAYOFF:4
GAME #997 PLAYER A: Cooperate DECISION: C PAYOFF:4 PLAYER B: Random DECISION: C PAYOFF:4
GAME #998 PLAYER A: Tit-for-tat DECISION: C PAYOFF:4 PLAYER B: WSLS DECISION: C PAYOFF:4
GAME #999 PLAYER A: Cooperate DECISION: C PAYOFF:4 PLAYER B: Cooperate DECISION: C PAYOFF:4
GAME #1000 PLAYER A: WSLS DECISION: C PAYOFF:4 PLAYER B: WSLS DECISION: C PAYOFF:4

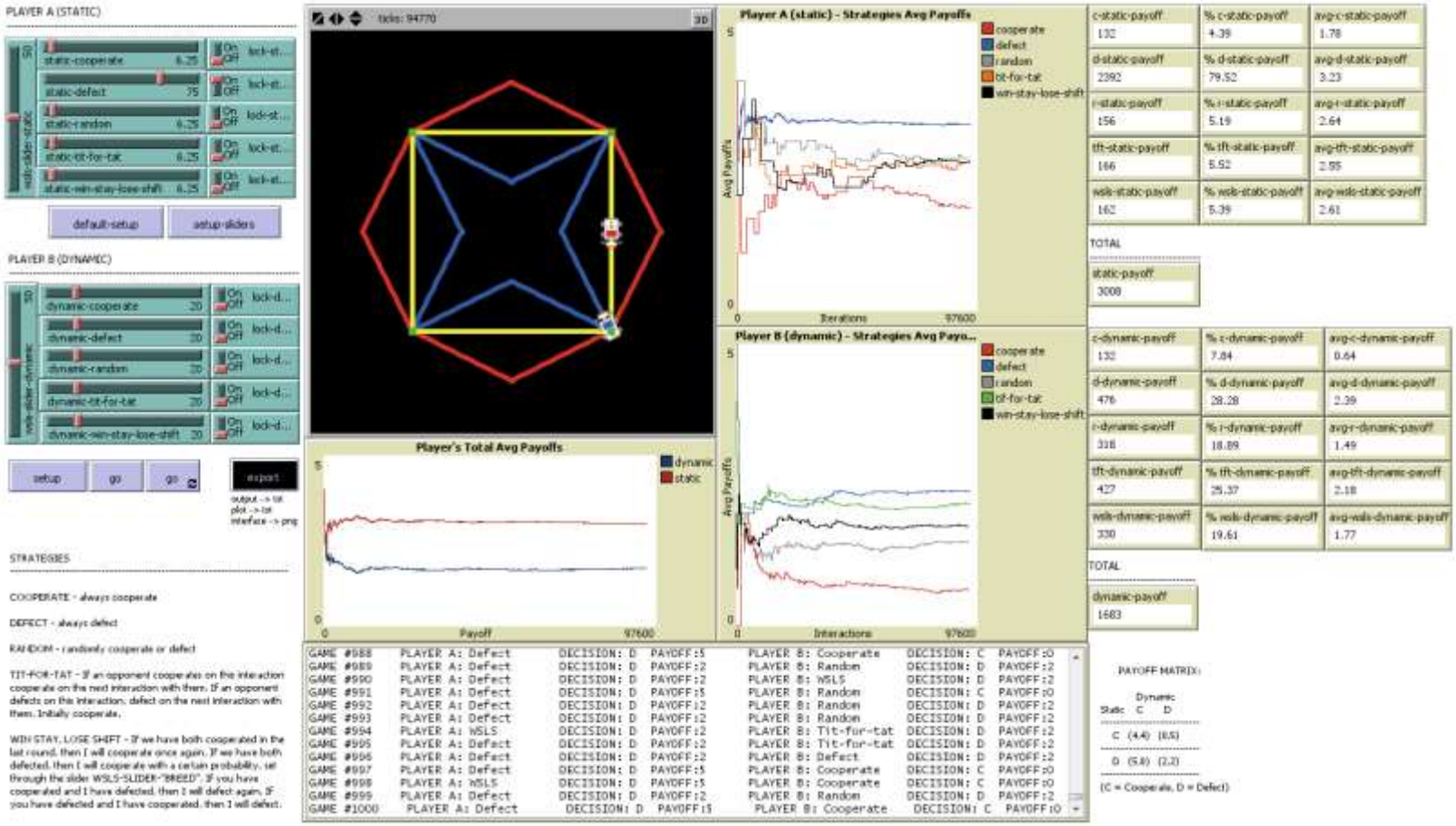

PLAYER A (STATIC)
static-cooperate
static-defect
static-random
static-tit-for-tat
static-win-stay-lose-shift
default-setup
setup-sliders
PLAYER B (DYNAMIC)
dynamic-cooperate
dynamic-defect
dynamic-random
dynamic-tit-for-tat
dynamic-win-stay-lose-shift
setup
go
export
STRATEGIES
COOPERATE - always cooperate
DEFECT - always defect
RANDOM - randomly cooperate or defect
TIT-FOR-TAT - If an opponent cooperates on this interaction cooperate on the next interaction with them. If an opponent defects on this interaction, defect on the next interaction with them. Initially cooperate.
WIN-STAY, LOSE SHIFT - If we have both cooperated in the last round, then I will cooperate once again. If we have both defected, then I will cooperate with a certain probability, set through the slider WSLS-SLIDER-"BREED". If you have cooperated and I have defected, then I will defect again. If you have defected and I have cooperated, then I will defect.
Player A (static) - Strategies Avg Payoffs
Player B (dynamic) - Strategies Avg Payo...
Player's Total Avg Payoffs
Iterations
Payoff
cooperate
defect
random
tit-for-tat
win-stay-lose-shift
dynamic
static
c-static-payoff
% c-static-payoff
avg-c-static-payoff
d-static-payoff
% d-static-payoff
avg-d-static-payoff
r-static-payoff
% r-static-payoff
avg-r-static-payoff
tft-static-payoff
% tft-static-payoff
avg-tft-static-payoff
wsls-static-payoff
% wsls-static-payoff
avg-wsls-static-payoff
TOTAL
static-payoff
c-dynamic-payoff
% c-dynamic-payoff
avg-c-dynamic-payoff
d-dynamic-payoff
% d-dynamic-payoff
avg-d-dynamic-payoff
r-dynamic-payoff
% r-dynamic-payoff
avg-r-dynamic-payoff
tft-dynamic-payoff
% tft-dynamic-payoff
avg-tft-dynamic-payoff
wsls-dynamic-payoff
% wsls-dynamic-payoff
avg-wsls-dynamic-payoff
TOTAL
dynamic-payoff
PAYOFF MATRIX:
(C = Cooperate, D = Defect)

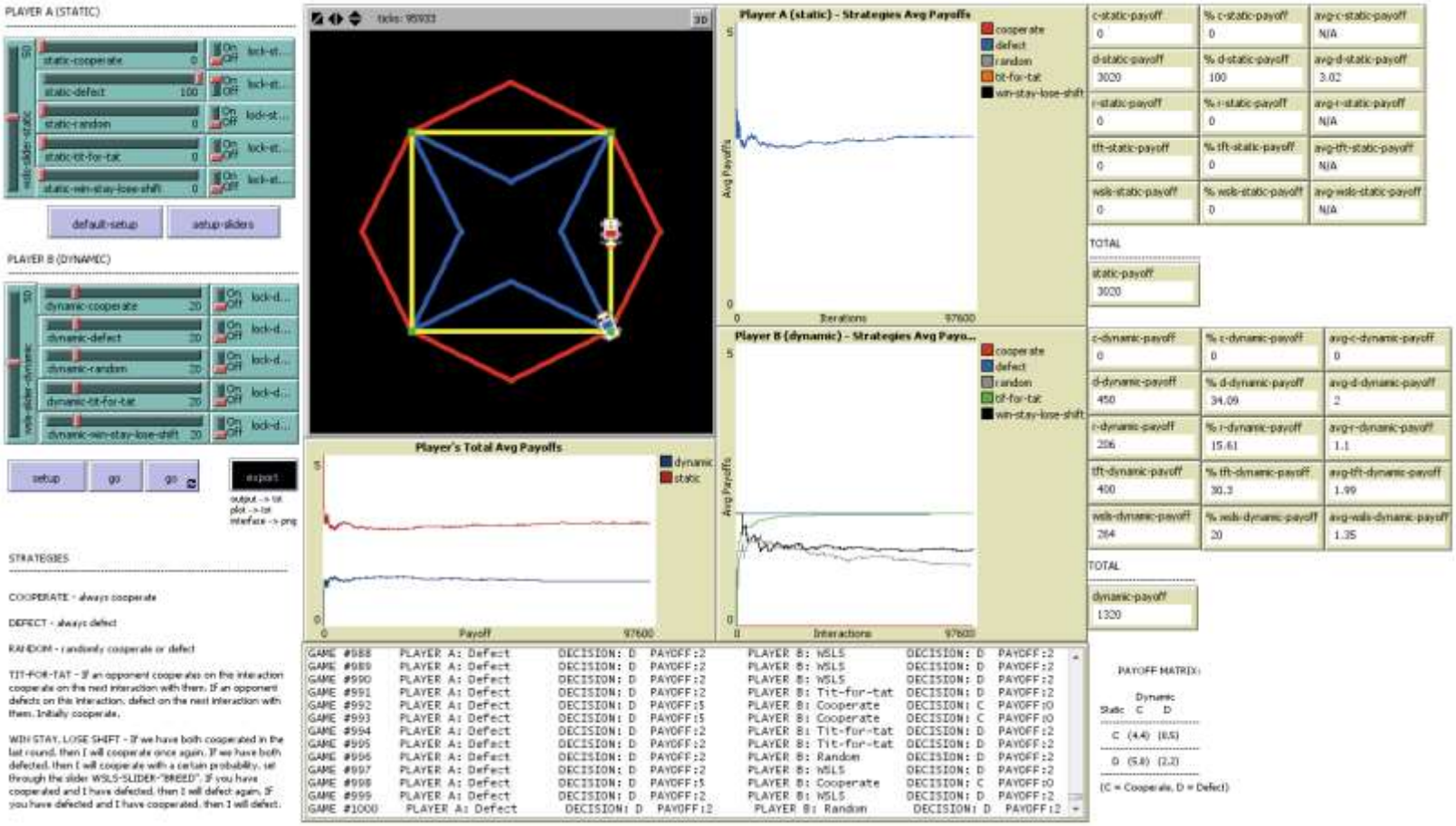

PLAYER A (STATIC)
static-cooperate
static-defect
static-random
static-tit-for-tat
static-win-stay-lose-shift
default-setup
setup-sliders
PLAYER B (DYNAMIC)
dynamic-cooperate
dynamic-defect
dynamic-random
dynamic-tit-for-tat
dynamic-win-stay-lose-shift
setup
go
export
STRATEGIES
COOPERATE - always cooperate
DEFECT - always defect
RANDOM - randomly cooperate or defect
TIT-FOR-TAT - If an opponent cooperates on the interaction cooperate on the next interaction with them. If an opponent defects on the interaction, defect on the next interaction with them. Initially cooperate.
WIN-STAY, LOSE-SHIFT - If we have both cooperated in the last round, then I will cooperate once again. If we have both defected, then I will cooperate with a certain probability, set through the slider WSLS-SLIDER-"BREED". If you have cooperated and I have defected, then I will defect again. If you have defected and I have cooperated, then I will defect.
Player A (static) - Strategies Avg Payoffs
Avg Payoffs
Iterations
cooperate
defect
random
tit-for-tat
win-stay-lose-shift
Player B (dynamic) - Strategies Avg Payo...
Player's Total Avg Payoffs
Payoff
dynamic
static
c-static-payoff
% c-static-payoff
avg-c-static-payoff
d-static-payoff
% d-static-payoff
avg-d-static-payoff
r-static-payoff
% r-static-payoff
avg-r-static-payoff
tft-static-payoff
% tft-static-payoff
avg-tft-static-payoff
wsls-static-payoff
% wsls-static-payoff
avg-wsls-static-payoff
TOTAL
static-payoff
c-dynamic-payoff
% c-dynamic-payoff
avg-c-dynamic-payoff
d-dynamic-payoff
% d-dynamic-payoff
avg-d-dynamic-payoff
r-dynamic-payoff
% r-dynamic-payoff
avg-r-dynamic-payoff
tft-dynamic-payoff
% tft-dynamic-payoff
avg-tft-dynamic-payoff
wsls-dynamic-payoff
% wsls-dynamic-payoff
avg-wsls-dynamic-payoff
TOTAL
dynamic-payoff
PAYOFF MATRIX:
Dynamic
Static C D
C (4,4) (8,5)
D (5,8) (2,2)
(C = Cooperate, D = Defect)

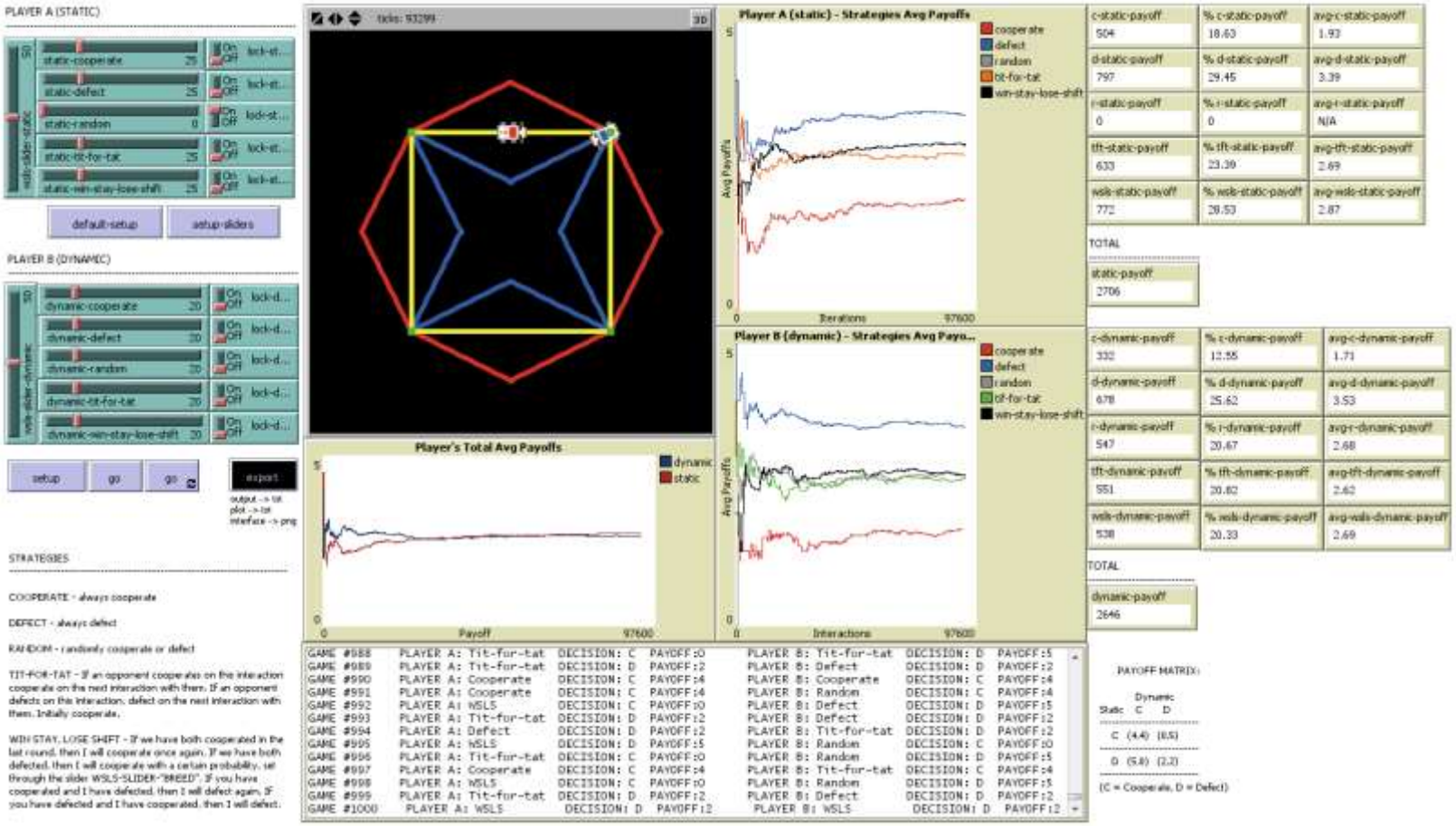

PLAYER A (STATIC)
static-cooperate
static-defect
static-random
static-tit-for-tat
static-win-stay-lose-shift
default-setup
setup-sliders
PLAYER B (DYNAMIC)
dynamic-cooperate
dynamic-defect
dynamic-random
dynamic-tit-for-tat
dynamic-win-stay-lose-shift
setup
go
export
STRATEGIES
COOPERATE - always cooperate
DEFECT - always defect
RANDOM - randomly cooperate or defect
TIT-FOR-TAT - If an opponent cooperates on the interaction cooperate on the next interaction with them. If an opponent defects on the interaction, defect on the next interaction with them. Initially cooperate.
WIN-STAY, LOSE-SHIFT - If we have both cooperated in the last round, then I will cooperate once again. If we have both defected, then I will cooperate with a certain probability, set through the slider WSLS-SLIDER-"BREED". If you have cooperated and I have defected, then I will defect again. If you have defected and I have cooperated, then I will defect.
ticks: 104411
Player A (static) - Strategies Avg Payoffs
cooperate
defect
random
tit-for-tat
win-stay-lose-shift
Avg Payoffs
Iterations
Player B (dynamic) - Strategies Avg Payo...
Player's Total Avg Payoffs
dynamic
static
Payoff
c-static-payoff
% c-static-payoff
avg-c-static-payoff
d-static-payoff
% d-static-payoff
avg-d-static-payoff
r-static-payoff
% r-static-payoff
avg-r-static-payoff
tft-static-payoff
% tft-static-payoff
avg-tft-static-payoff
wsls-static-payoff
% wsls-static-payoff
avg-wsls-static-payoff
TOTAL
static-payoff
c-dynamic-payoff
% c-dynamic-payoff
avg-c-dynamic-payoff
d-dynamic-payoff
% d-dynamic-payoff
avg-d-dynamic-payoff
r-dynamic-payoff
% r-dynamic-payoff
avg-r-dynamic-payoff
tft-dynamic-payoff
% tft-dynamic-payoff
avg-tft-dynamic-payoff
wsls-dynamic-payoff
% wsls-dynamic-payoff
avg-wsls-dynamic-payoff
TOTAL
dynamic-payoff
PAYOFF MATRIX:
Dynamic
Static C D
C (4,4) (0,5)
D (5,0) (2,2)
(C = Cooperate, D = Defect)

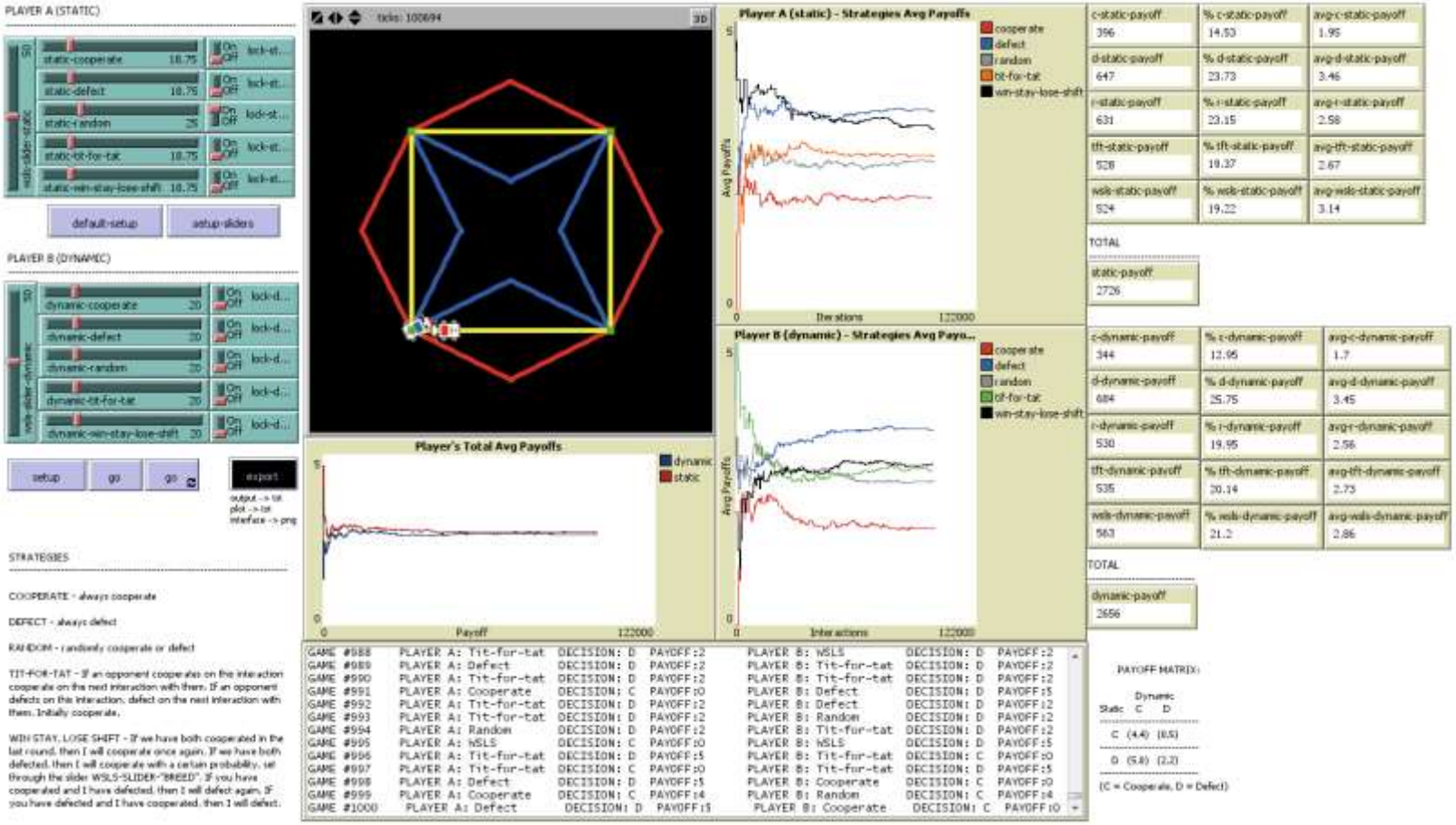

PLAYER A (STATIC)
static-cooperate 18.75
static-defect 25
static-random 18.75
static-tit-for-tat 18.75
static-win-stay-lose-shift 18.75
default-setup
setup-sliders
PLAYER B (DYNAMIC)
dynamic-cooperate 20
dynamic-defect 20
dynamic-random 20
dynamic-tit-for-tat 20
dynamic-win-stay-lose-shift 20
setup
go
go
export
STRATEGIES
COOPERATE - always cooperate
DEFECT - always defect
RANDOM - randomly cooperate or defect
TIT-FOR-TAT - If an opponent cooperates on the interaction cooperate on the next interaction with them. If an opponent defects on this interaction, defect on the next interaction with them. Initially cooperate.
WIN-STAY, LOSE-SHIFT - If we have both cooperated in the last round, then I will cooperate once again. If we have both defected, then I will cooperate with a certain probability, set through the slider WSLS-SLIDER-"BREED". If you have cooperated and I have defected, then I will defect again. If you have defected and I have cooperated, then I will defect.
Player A (static) - Strategies Avg Payoffs
cooperate
defect
random
tit-for-tat
win-stay-lose-shift
Avg Payoffs
Iterations
Player B (dynamic) - Strategies Avg Payo...
Player's Total Avg Payoffs
dynamic
static
Payoff
c-static-payoff 328
% c-static-payoff 12.5
avg-c-static-payoff 1.62
d-static-payoff 898
% d-static-payoff 34.22
avg-d-static-payoff 3.34
r-static-payoff 474
% r-static-payoff 18.06
avg-r-static-payoff 2.6
tft-static-payoff 482
% tft-static-payoff 18.37
avg-tft-static-payoff 2.75
wsls-static-payoff 442
% wsls-static-payoff 16.84
avg-wsls-static-payoff 2.57
TOTAL
static-payoff 2624
c-dynamic-payoff 328
% c-dynamic-payoff 12.64
avg-c-dynamic-payoff 1.63
d-dynamic-payoff 664
% d-dynamic-payoff 25.6
avg-d-dynamic-payoff 3.27
r-dynamic-payoff 449
% r-dynamic-payoff 17.31
avg-r-dynamic-payoff 2.24
tft-dynamic-payoff 521
% tft-dynamic-payoff 20.08
avg-tft-dynamic-payoff 2.58
wsls-dynamic-payoff 632
% wsls-dynamic-payoff 24.36
avg-wsls-dynamic-payoff 3.04
TOTAL
dynamic-payoff 2594
PAYOFF MATRIX:
Dynamic
Static C D
C (4,4) (0,5)
D (5,0) (2,2)
(C = Cooperate, D = Defect)
GAME #988 PLAYER A: Defect DECISION: D PAYOFF:2 PLAYER B: WSLS DECISION: D PAYOFF:2
GAME #989 PLAYER A: Defect DECISION: D PAYOFF:2 PLAYER B: Tit-for-tat DECISION: D PAYOFF:2
GAME #990 PLAYER A: Tit-for-tat DECISION: D PAYOFF:2 PLAYER B: Random DECISION: D PAYOFF:2
GAME #991 PLAYER A: Tit-for-tat DECISION: D PAYOFF:2 PLAYER B: Defect DECISION: D PAYOFF:2
GAME #992 PLAYER A: Random DECISION: C PAYOFF:0 PLAYER B: WSLS DECISION: D PAYOFF:5
GAME #993 PLAYER A: Random DECISION: C PAYOFF:0 PLAYER B: Random DECISION: D PAYOFF:5
GAME #994 PLAYER A: Cooperate DECISION: C PAYOFF:0 PLAYER B: Defect DECISION: D PAYOFF:5
GAME #995 PLAYER A: Random DECISION: C PAYOFF:0 PLAYER B: Defect DECISION: D PAYOFF:5
GAME #996 PLAYER A: WSLS DECISION: D PAYOFF:2 PLAYER B: WSLS DECISION: D PAYOFF:2
GAME #997 PLAYER A: Defect DECISION: D PAYOFF:2 PLAYER B: WSLS DECISION: D PAYOFF:2
GAME #998 PLAYER A: WSLS DECISION: D PAYOFF:2 PLAYER B: Defect DECISION: D PAYOFF:2
GAME #999 PLAYER A: Defect DECISION: D PAYOFF:5 PLAYER B: Cooperate DECISION: C PAYOFF:0
GAME #1000 PLAYER A: Tit-for-tat DECISION: C PAYOFF:0 PLAYER B: WSLS DECISION: D PAYOFF:5

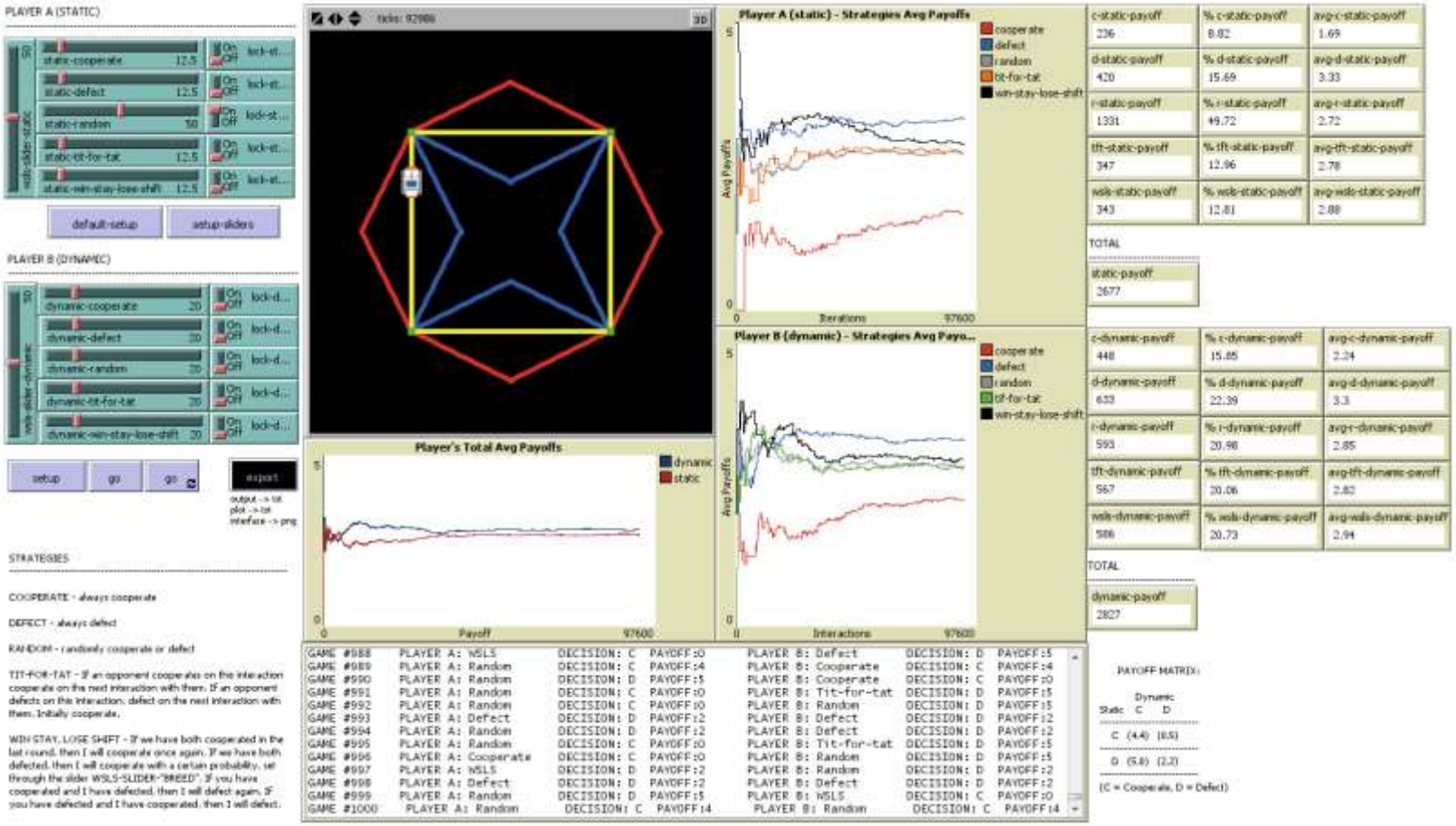

PLAYER A (STATIC)
static-cooperate 12.5
static-defect 50
static-random 12.5
static-tit-for-tat 12.5
static-win-stay-lose-shift 12.5
default-setup
setup-sliders
PLAYER B (DYNAMIC)
dynamic-cooperate 20
dynamic-defect 20
dynamic-random 20
dynamic-tit-for-tat 20
dynamic-win-stay-lose-shift 20
setup
go
export
STRATEGIES
COOPERATE - always cooperate
DEFECT - always defect
RANDOM - randomly cooperate or defect
TIT-FOR-TAT - If an opponent cooperates on the interaction cooperate on the next interaction with them. If an opponent defects on this interaction, defect on the next interaction with them. Initially cooperate.
WIN-STAY, LOSE-SHIFT - If we have both cooperated in the last round, then I will cooperate once again. If we have both defected, then I will cooperate with a certain probability, set through the slider WSLS-SLIDER-"BREED". If you have cooperated and I have defected, then I will defect again. If you have defected and I have cooperated, then I will defect.
Player A (static) - Strategies Avg Payoffs
Player B (dynamic) - Strategies Avg Payo...
Player's Total Avg Payoffs
cooperate
defect
random
tit-for-tat
win-stay-lose-shift
dynamic
static
Iterations
Payoff
c-static-payoff 264
% c-static-payoff 9.11
avg-c-static-payoff 1.93
d-static-payoff 1671
% d-static-payoff 57.64
avg-d-static-payoff 3.32
r-static-payoff 345
% r-static-payoff 11.9
avg-r-static-payoff 2.8
tft-static-payoff 319
% tft-static-payoff 11
avg-tft-static-payoff 2.53
wsls-static-payoff 300
% wsls-static-payoff 10.35
avg-wsls-static-payoff 2.73
TOTAL
static-payoff 2899
c-dynamic-payoff 308
% c-dynamic-payoff 14.33
avg-c-dynamic-payoff 1.36
d-dynamic-payoff 558
% d-dynamic-payoff 25.97
avg-d-dynamic-payoff 3
r-dynamic-payoff 404
% r-dynamic-payoff 18.8
avg-r-dynamic-payoff 2.01
tft-dynamic-payoff 425
% tft-dynamic-payoff 19.78
avg-tft-dynamic-payoff 2.34
wsls-dynamic-payoff 454
% wsls-dynamic-payoff 21.13
avg-wsls-dynamic-payoff 2.21
TOTAL
dynamic-payoff 2149
PAYOFF MATRIX:
Dynamic
Static C D
C (4,4) (0,5)
D (5,0) (2,2)
(C = Cooperate, D = Defect)
GAME #988 PLAYER A: Defect DECISION: D PAYOFF:2 PLAYER B: Random DECISION: D PAYOFF:2
GAME #989 PLAYER A: Tit-for-tat DECISION: D PAYOFF:5 PLAYER B: WSLS DECISION: C PAYOFF:0
GAME #990 PLAYER A: Tit-for-tat DECISION: C PAYOFF:0 PLAYER B: Defect DECISION: D PAYOFF:5
GAME #991 PLAYER A: Defect DECISION: D PAYOFF:5 PLAYER B: Cooperate DECISION: C PAYOFF:0
GAME #992 PLAYER A: Cooperate DECISION: C PAYOFF:4 PLAYER B: Cooperate DECISION: C PAYOFF:4
GAME #993 PLAYER A: Defect DECISION: D PAYOFF:5 PLAYER B: Cooperate DECISION: C PAYOFF:0
GAME #994 PLAYER A: Random DECISION: C PAYOFF:4 PLAYER B: Cooperate DECISION: C PAYOFF:4
GAME #995 PLAYER A: Defect DECISION: D PAYOFF:5 PLAYER B: WSLS DECISION: C PAYOFF:0
GAME #996 PLAYER A: WSLS DECISION: D PAYOFF:2 PLAYER B: WSLS DECISION: D PAYOFF:2
GAME #997 PLAYER A: Random DECISION: D PAYOFF:2 PLAYER B: Tit-for-tat DECISION: D PAYOFF:2
GAME #998 PLAYER A: Random DECISION: C PAYOFF:4 PLAYER B: Random DECISION: C PAYOFF:4
GAME #999 PLAYER A: WSLS DECISION: C PAYOFF:4 PLAYER B: WSLS DECISION: C PAYOFF:4
GAME #1000 PLAYER A: Defect DECISION: D PAYOFF:2 PLAYER B: Defect DECISION: D PAYOFF:2

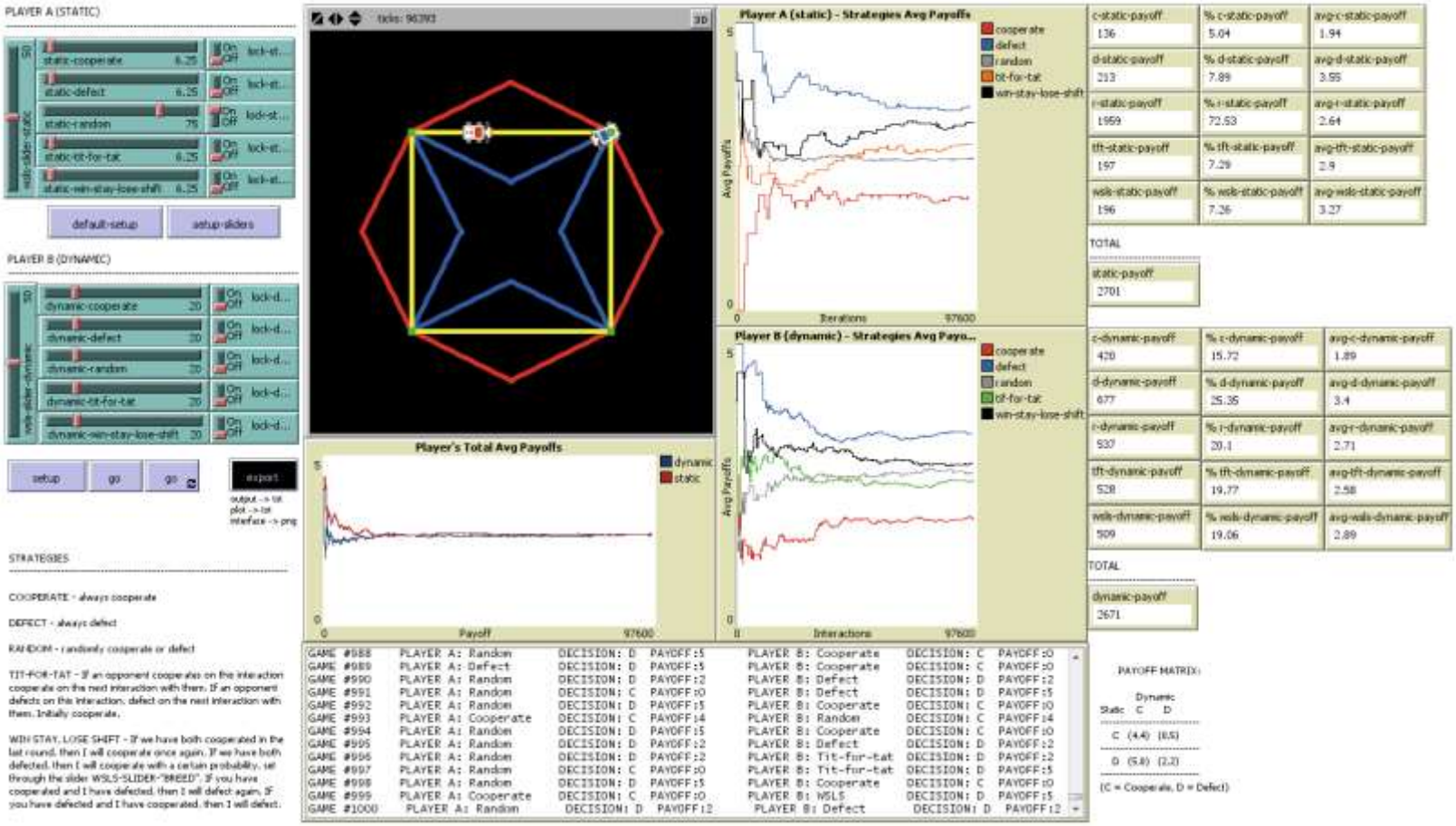

PLAYER A (STATIC)
static-cooperate
static-defect
static-random
static-tit-for-tat
static-win-stay-lose-shift
default-setup
setup-sliders
PLAYER B (DYNAMIC)
dynamic-cooperate
dynamic-defect
dynamic-random
dynamic-tit-for-tat
dynamic-win-stay-lose-shift
setup
go
export
STRATEGIES
COOPERATE - always cooperate
DEFECT - always defect
RANDOM - randomly cooperate or defect
TIT-FOR-TAT - If an opponent cooperates on the interaction cooperate on the next interaction with them. If an opponent defects on this interaction, defect on the next interaction with them. Initially cooperate.
WIN-STAY, LOSE-SHIFT - If we have both cooperated in the last round, then I will cooperate once again. If we have both defected, then I will cooperate with a certain probability, set through the slider WSLS-SLIDER-"BREED". If you have cooperated and I have defected, then I will defect again. If you have defected and I have cooperated, then I will defect.
ticks: 94770
Player A (static) - Strategies Avg Payoffs
cooperate
defect
random
tit-for-tat
win-stay-lose-shift
Avg Payoffs
Iterations
Player's Total Avg Payoffs
dynamic
static
Payoff
Player B (dynamic) - Strategies Avg Payo...
c-static-payoff 132
% c-static-payoff 4.39
avg-c-static-payoff 1.78
d-static-payoff 2392
% d-static-payoff 79.52
avg-d-static-payoff 3.23
r-static-payoff 156
% r-static-payoff 5.19
avg-r-static-payoff 2.64
tft-static-payoff 166
% tft-static-payoff 5.52
avg-tft-static-payoff 2.55
wsls-static-payoff 162
% wsls-static-payoff 5.39
avg-wsls-static-payoff 2.61
TOTAL
static-payoff 3008
c-dynamic-payoff 132
% c-dynamic-payoff 7.84
avg-c-dynamic-payoff 0.64
d-dynamic-payoff 476
% d-dynamic-payoff 28.28
avg-d-dynamic-payoff 2.39
r-dynamic-payoff 318
% r-dynamic-payoff 18.89
avg-r-dynamic-payoff 1.49
tft-dynamic-payoff 427
% tft-dynamic-payoff 25.37
avg-tft-dynamic-payoff 2.18
wsls-dynamic-payoff 330
% wsls-dynamic-payoff 19.61
avg-wsls-dynamic-payoff 1.77
TOTAL
dynamic-payoff 1683
GAME #988 PLAYER A: Defect DECISION: D PAYOFF:5 PLAYER B: Cooperate DECISION: C PAYOFF:0
GAME #989 PLAYER A: Defect DECISION: D PAYOFF:2 PLAYER B: Random DECISION: D PAYOFF:2
GAME #990 PLAYER A: Defect DECISION: D PAYOFF:2 PLAYER B: WSLS DECISION: D PAYOFF:2
GAME #991 PLAYER A: Defect DECISION: D PAYOFF:5 PLAYER B: Random DECISION: C PAYOFF:0
GAME #992 PLAYER A: Defect DECISION: D PAYOFF:2 PLAYER B: Random DECISION: D PAYOFF:2
GAME #993 PLAYER A: Defect DECISION: D PAYOFF:2 PLAYER B: Random DECISION: D PAYOFF:2
GAME #994 PLAYER A: WSLS DECISION: D PAYOFF:2 PLAYER B: Tit-for-tat DECISION: D PAYOFF:2
GAME #995 PLAYER A: Defect DECISION: D PAYOFF:2 PLAYER B: Tit-for-tat DECISION: D PAYOFF:2
GAME #996 PLAYER A: Defect DECISION: D PAYOFF:2 PLAYER B: Defect DECISION: D PAYOFF:2
GAME #997 PLAYER A: Defect DECISION: D PAYOFF:5 PLAYER B: Cooperate DECISION: C PAYOFF:0
GAME #998 PLAYER A: WSLS DECISION: D PAYOFF:5 PLAYER B: Cooperate DECISION: C PAYOFF:0
GAME #999 PLAYER A: Defect DECISION: D PAYOFF:2 PLAYER B: Random DECISION: D PAYOFF:2
GAME #1000 PLAYER A: Defect DECISION: D PAYOFF:5 PLAYER B: Cooperate DECISION: C PAYOFF:0
PAYOFF MATRIX:
Dynamic
Static C D
C (4,4) (0,5)
D (5,0) (2,2)
(C = Cooperate, D = Defect)

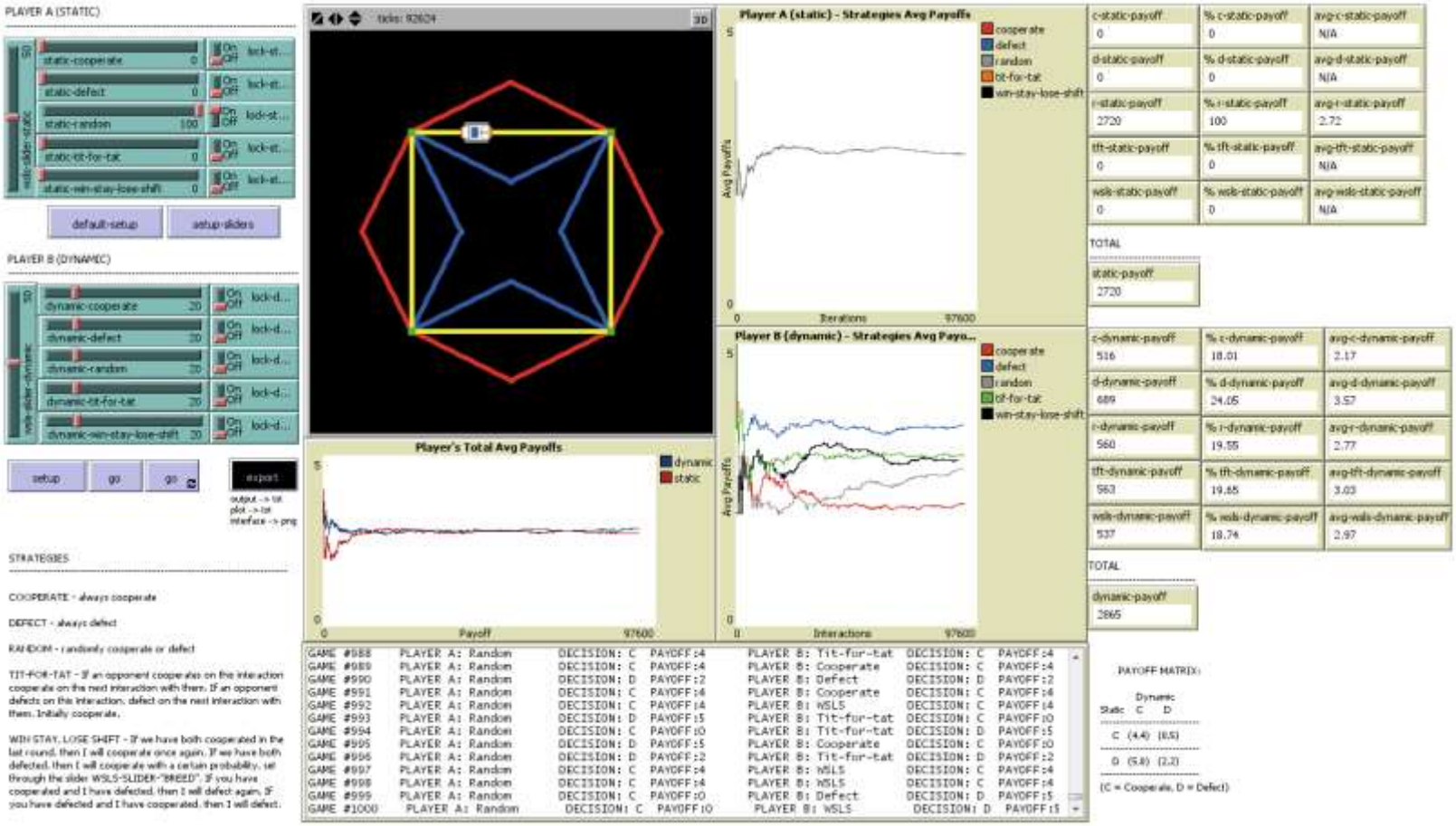

PLAYER A (STATIC)
static-cooperate
static-defect
static-random
static-tit-for-tat
static-win-stay-lose-shift
default-setup
setup-sliders
PLAYER B (DYNAMIC)
dynamic-cooperate
dynamic-defect
dynamic-random
dynamic-tit-for-tat
dynamic-win-stay-lose-shift
setup
go
export
STRATEGIES
COOPERATE - always cooperate
DEFECT - always defect
RANDOM - randomly cooperate or defect
TIT-FOR-TAT - If an opponent cooperates on the interaction cooperate on the next interaction with them. If an opponent defects on this interaction, defect on the next interaction with them. Initially cooperate.
WIN-STAY, LOSE-SHIFT - If we have both cooperated in the last round, then I will cooperate once again. If we have both defected, then I will cooperate with a certain probability, set through the slider WSLS-SLIDER-"BREED". If you have cooperated and I have defected, then I will defect again. If you have defected and I have cooperated, then I will defect.
Player A (static) - Strategies Avg Payoffs
Player B (dynamic) - Strategies Avg Payo...
Player's Total Avg Payoffs
Avg Payoffs
Iterations
Payoff
cooperate
defect
random
tit-for-tat
win-stay-lose-shift
dynamic
static
c-static-payoff
% c-static-payoff
avg-c-static-payoff
d-static-payoff
% d-static-payoff
avg-d-static-payoff
r-static-payoff
% r-static-payoff
avg-r-static-payoff
tft-static-payoff
% tft-static-payoff
avg-tft-static-payoff
wsls-static-payoff
% wsls-static-payoff
avg-wsls-static-payoff
TOTAL
static-payoff
c-dynamic-payoff
% c-dynamic-payoff
avg-c-dynamic-payoff
d-dynamic-payoff
% d-dynamic-payoff
avg-d-dynamic-payoff
r-dynamic-payoff
% r-dynamic-payoff
avg-r-dynamic-payoff
tft-dynamic-payoff
% tft-dynamic-payoff
avg-tft-dynamic-payoff
wsls-dynamic-payoff
% wsls-dynamic-payoff
avg-wsls-dynamic-payoff
TOTAL
dynamic-payoff
PAYOFF MATRIX:
Dynamic
Static C D
C (4,4) (0,5)
D (5,0) (2,2)
(C = Cooperate, D = Defect)

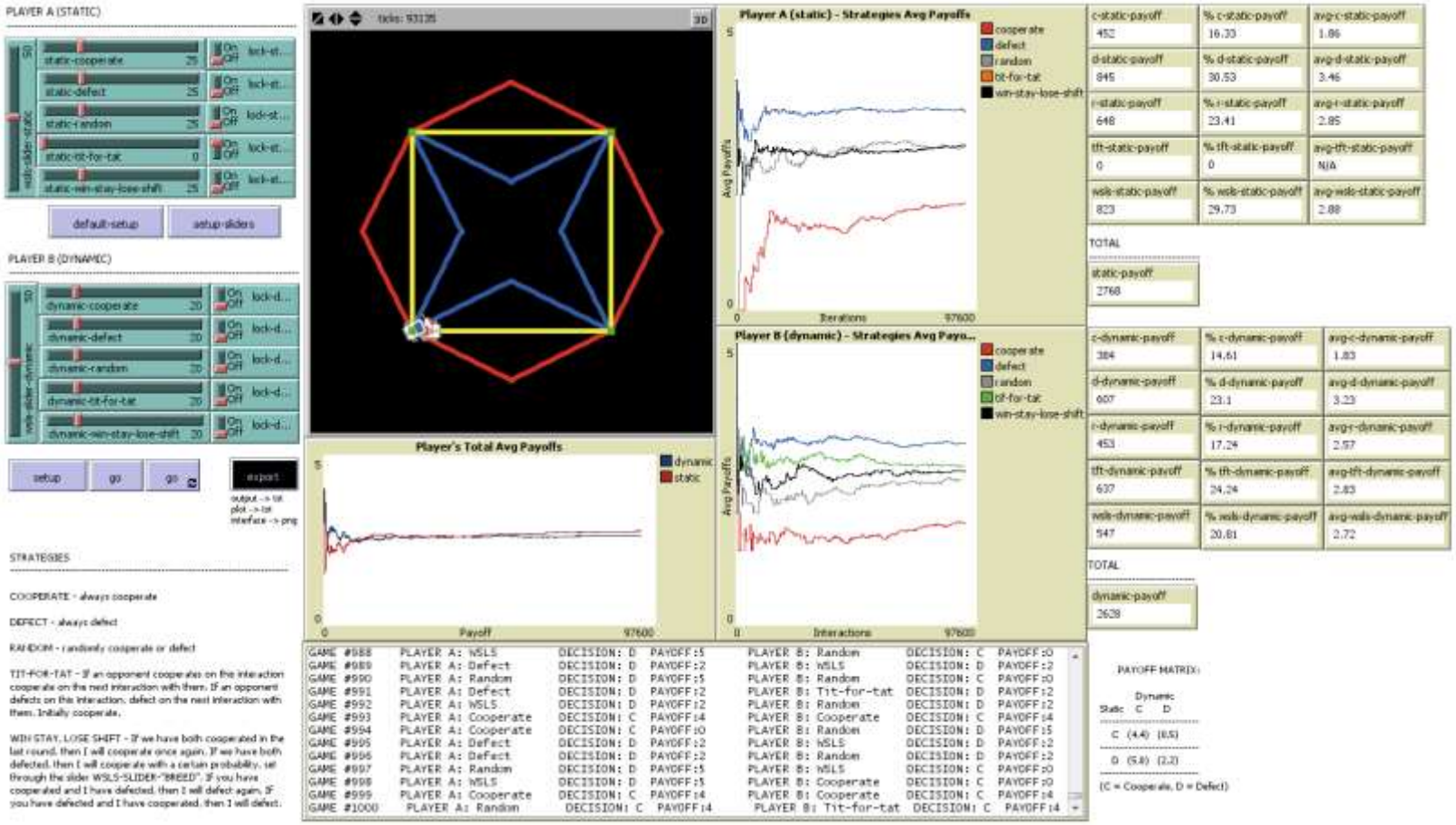

PLAYER A (STATIC)
static-cooperate
static-defect
static-random
static-tit-for-tat
static-win-stay-lose-shift
default-setup
setup-sliders
PLAYER B (DYNAMIC)
dynamic-cooperate
dynamic-defect
dynamic-random
dynamic-tit-for-tat
dynamic-win-stay-lose-shift
setup
go
export
STRATEGIES
COOPERATE - always cooperate
DEFECT - always defect
RANDOM - randomly cooperate or defect
TIT-FOR-TAT - IF an opponent cooperates on the interaction cooperate on the next interaction with them. IF an opponent defects on this interaction, defect on the next interaction with them. Initially cooperate.
WIN-STAY, LOSE-SHIFT - IF we have both cooperated in the last round, then I will cooperate once again. IF we have both defected, then I will cooperate with a certain probability, set through the slider WSLS-SLIDER-"BREED". IF you have cooperated and I have defected, then I will defect again. IF you have defected and I have cooperated, then I will defect.
Player A (static) - Strategies Avg Payoffs
Player B (dynamic) - Strategies Avg Payo...
Player's Total Avg Payoffs
cooperate
defect
random
tit-for-tat
win-stay-lose-shift
dynamic
static
Avg Payoffs
Iterations
Payoff
c-static-payoff
% c-static-payoff
avg-c-static-payoff
d-static-payoff
% d-static-payoff
avg-d-static-payoff
r-static-payoff
% r-static-payoff
avg-r-static-payoff
tft-static-payoff
% tft-static-payoff
avg-tft-static-payoff
wsls-static-payoff
% wsls-static-payoff
avg-wsls-static-payoff
TOTAL
static-payoff
c-dynamic-payoff
% c-dynamic-payoff
avg-c-dynamic-payoff
d-dynamic-payoff
% d-dynamic-payoff
avg-d-dynamic-payoff
r-dynamic-payoff
% r-dynamic-payoff
avg-r-dynamic-payoff
tft-dynamic-payoff
% tft-dynamic-payoff
avg-tft-dynamic-payoff
wsls-dynamic-payoff
% wsls-dynamic-payoff
avg-wsls-dynamic-payoff
TOTAL
dynamic-payoff
PAYOFF MATRIX:
Dynamic
Static C D
C (4,4) (0,5)
D (5,0) (2,2)
(C = Cooperate, D = Defect)

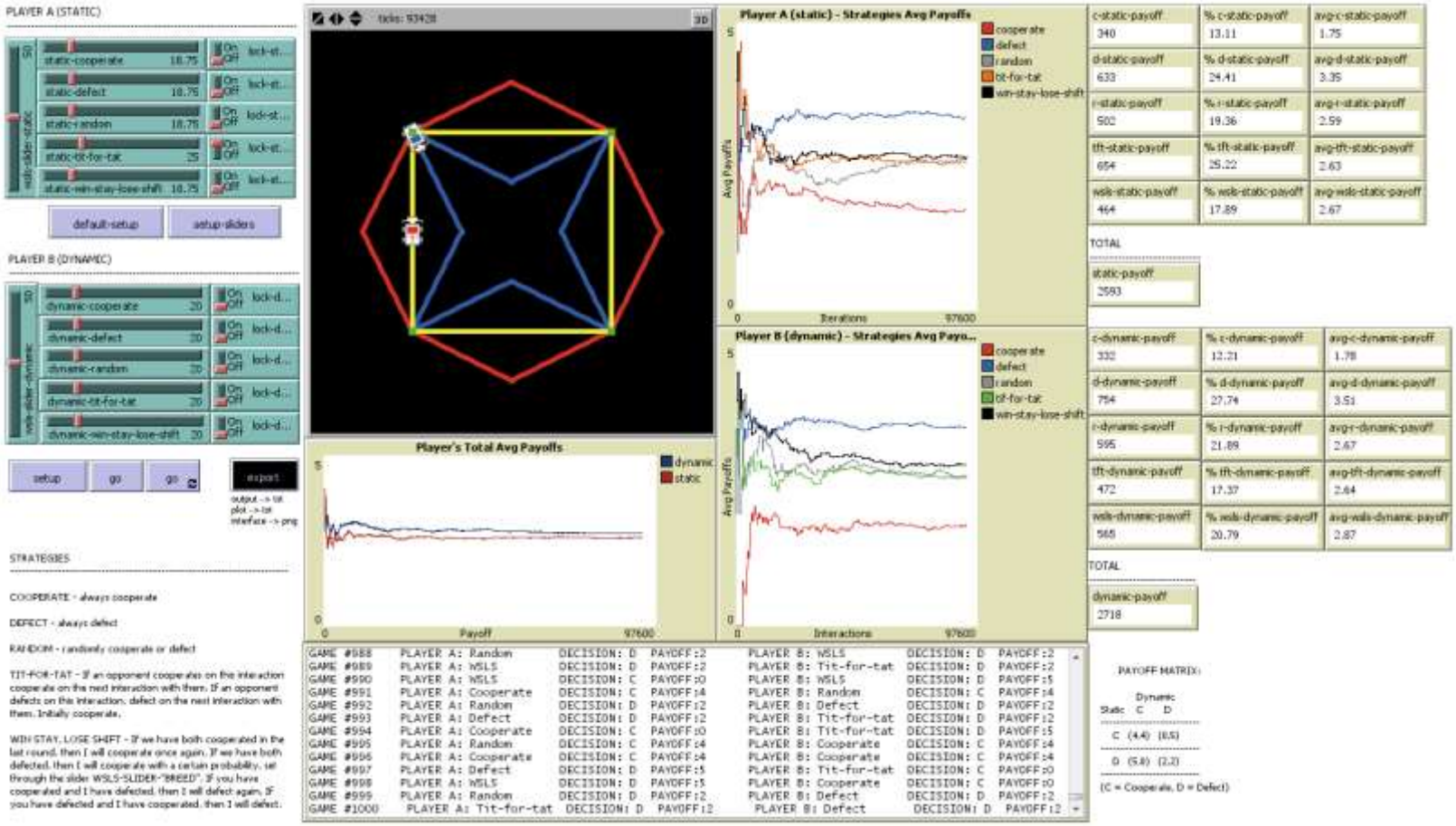

PLAYER A (STATIC)
static-cooperate
static-defect
static-random
static-tit-for-tat
static-win-stay-lose-shift
default-setup
setup-sliders
PLAYER B (DYNAMIC)
dynamic-cooperate
dynamic-defect
dynamic-random
dynamic-tit-for-tat
dynamic-win-stay-lose-shift
setup
go
export
STRATEGIES
COOPERATE - always cooperate
DEFECT - always defect
RANDOM - randomly cooperate or defect
TIT-FOR-TAT - If an opponent cooperates on the interaction cooperate on the next interaction with them. If an opponent defects on this interaction, defect on the next interaction with them. Initially cooperate.
WIN-STAY, LOSE-SHIFT - If we have both cooperated in the last round, then I will cooperate once again. If we have both defected, then I will cooperate with a certain probability, set through the slider WSLS-SLIDER-"BREED". If you have cooperated and I have defected, then I will defect again. If you have defected and I have cooperated, then I will defect.
Player A (static) - Strategies Avg Payoffs
Player B (dynamic) - Strategies Avg Payo...
Player's Total Avg Payoffs
cooperate
defect
random
tit-for-tat
win-stay-lose-shift
dynamic
static
Iterations
Payoff
c-static-payoff
% c-static-payoff
avg-c-static-payoff
d-static-payoff
% d-static-payoff
avg-d-static-payoff
r-static-payoff
% r-static-payoff
avg-r-static-payoff
tft-static-payoff
% tft-static-payoff
avg-tft-static-payoff
wsls-static-payoff
% wsls-static-payoff
avg-wsls-static-payoff
TOTAL
static-payoff
c-dynamic-payoff
% c-dynamic-payoff
avg-c-dynamic-payoff
d-dynamic-payoff
% d-dynamic-payoff
avg-d-dynamic-payoff
r-dynamic-payoff
% r-dynamic-payoff
avg-r-dynamic-payoff
tft-dynamic-payoff
% tft-dynamic-payoff
avg-tft-dynamic-payoff
wsls-dynamic-payoff
% wsls-dynamic-payoff
avg-wsls-dynamic-payoff
TOTAL
dynamic-payoff
PAYOFF MATRIX:
(C = Cooperate, D = Defect)

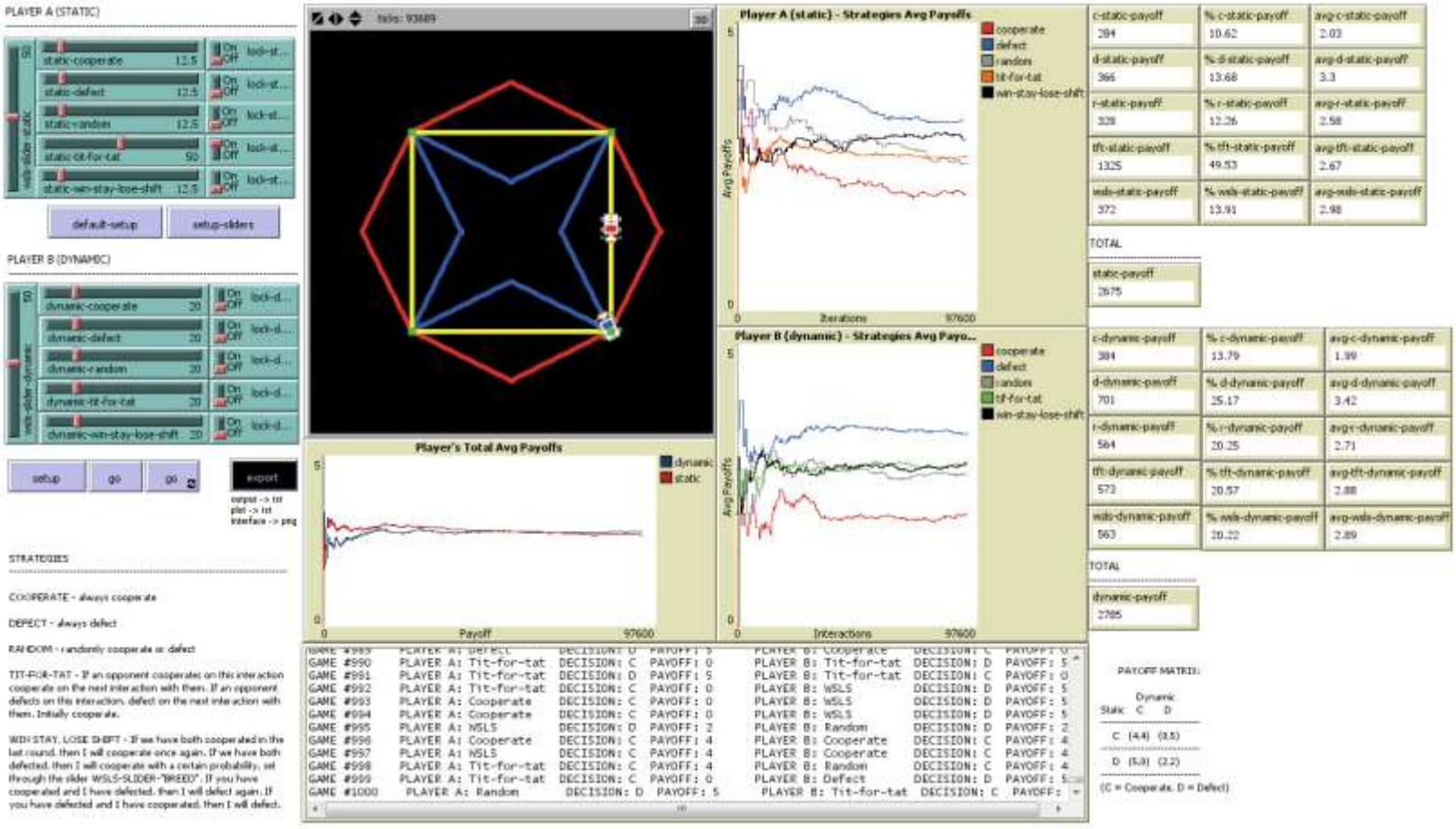

PLAYER A (STATIC)
static-cooperate 12.5
static-defect 12.5
static-random 50
static-tit-for-tat 12.5
static-win-stay-lose-shift 12.5
default-setup
setup-sliders
PLAYER B (DYNAMIC)
dynamic-cooperate 20
dynamic-defect 20
dynamic-random 20
dynamic-tit-for-tat 20
dynamic-win-stay-lose-shift 20
setup
go
go
export
output -> txt
plot -> txt
interface -> png
STRATEGIES
COOPERATE - always cooperate
DEFECT - always defect
RANDOM - randomly cooperate or defect
TIT-FOR-TAT - If an opponent cooperates on the interaction cooperate on the next interaction with them. If an opponent defects on this interaction, defect on the next interaction with them. Initially cooperate.
WIN-STAY, LOSE-SHIFT - If we have both cooperated in the last round, then I will cooperate once again. If we have both defected, then I will cooperate with a certain probability, set through the slider WSLS-SLIDER-"BREED". If you have cooperated and I have defected, then I will defect again. If you have defected and I have cooperated, then I will defect.
ticks: 92986
Player A (static) - Strategies Avg Payoffs
cooperate
defect
random
tit-for-tat
win-stay-lose-shift
Avg Payoffs
Iterations
97600
Player B (dynamic) - Strategies Avg Payo...
Player's Total Avg Payoffs
dynamic
static
Payoff
c-static-payoff 236
% c-static-payoff 8.82
avg-c-static-payoff 1.69
d-static-payoff 420
% d-static-payoff 15.69
avg-d-static-payoff 3.33
r-static-payoff 1331
% r-static-payoff 49.72
avg-r-static-payoff 2.72
tft-static-payoff 347
% tft-static-payoff 12.96
avg-tft-static-payoff 2.78
wsls-static-payoff 343
% wsls-static-payoff 12.81
avg-wsls-static-payoff 2.88
TOTAL
static-payoff 2677
c-dynamic-payoff 448
% c-dynamic-payoff 15.85
avg-c-dynamic-payoff 2.24
d-dynamic-payoff 633
% d-dynamic-payoff 22.39
avg-d-dynamic-payoff 3.3
r-dynamic-payoff 593
% r-dynamic-payoff 20.98
avg-r-dynamic-payoff 2.85
tft-dynamic-payoff 567
% tft-dynamic-payoff 20.06
avg-tft-dynamic-payoff 2.82
wsls-dynamic-payoff 586
% wsls-dynamic-payoff 20.73
avg-wsls-dynamic-payoff 2.94
TOTAL
dynamic-payoff 2827
GAME #988 PLAYER A: WSLS DECISION: C PAYOFF:0 PLAYER B: Defect DECISION: D PAYOFF:5
GAME #989 PLAYER A: Random DECISION: C PAYOFF:4 PLAYER B: Cooperate DECISION: C PAYOFF:4
GAME #990 PLAYER A: Random DECISION: D PAYOFF:5 PLAYER B: Cooperate DECISION: C PAYOFF:0
GAME #991 PLAYER A: Random DECISION: C PAYOFF:0 PLAYER B: Tit-for-tat DECISION: D PAYOFF:5
GAME #992 PLAYER A: Random DECISION: C PAYOFF:0 PLAYER B: Random DECISION: D PAYOFF:5
GAME #993 PLAYER A: Defect DECISION: D PAYOFF:2 PLAYER B: Defect DECISION: D PAYOFF:2
GAME #994 PLAYER A: Random DECISION: D PAYOFF:2 PLAYER B: Defect DECISION: D PAYOFF:2
GAME #995 PLAYER A: Random DECISION: C PAYOFF:0 PLAYER B: Tit-for-tat DECISION: D PAYOFF:5
GAME #996 PLAYER A: Cooperate DECISION: C PAYOFF:0 PLAYER B: Random DECISION: D PAYOFF:5
GAME #997 PLAYER A: WSLS DECISION: D PAYOFF:2 PLAYER B: Random DECISION: D PAYOFF:2
GAME #998 PLAYER A: Defect DECISION: D PAYOFF:2 PLAYER B: Defect DECISION: D PAYOFF:2
GAME #999 PLAYER A: Random DECISION: D PAYOFF:5 PLAYER B: WSLS DECISION: C PAYOFF:0
GAME #1000 PLAYER A: Random DECISION: C PAYOFF:4 PLAYER B: Random DECISION: C PAYOFF:4
PAYOFF MATRIX:
Dynamic
Static C D
C (4,4) (0,5)
D (5,0) (2,2)
(C = Cooperate, D = Defect)

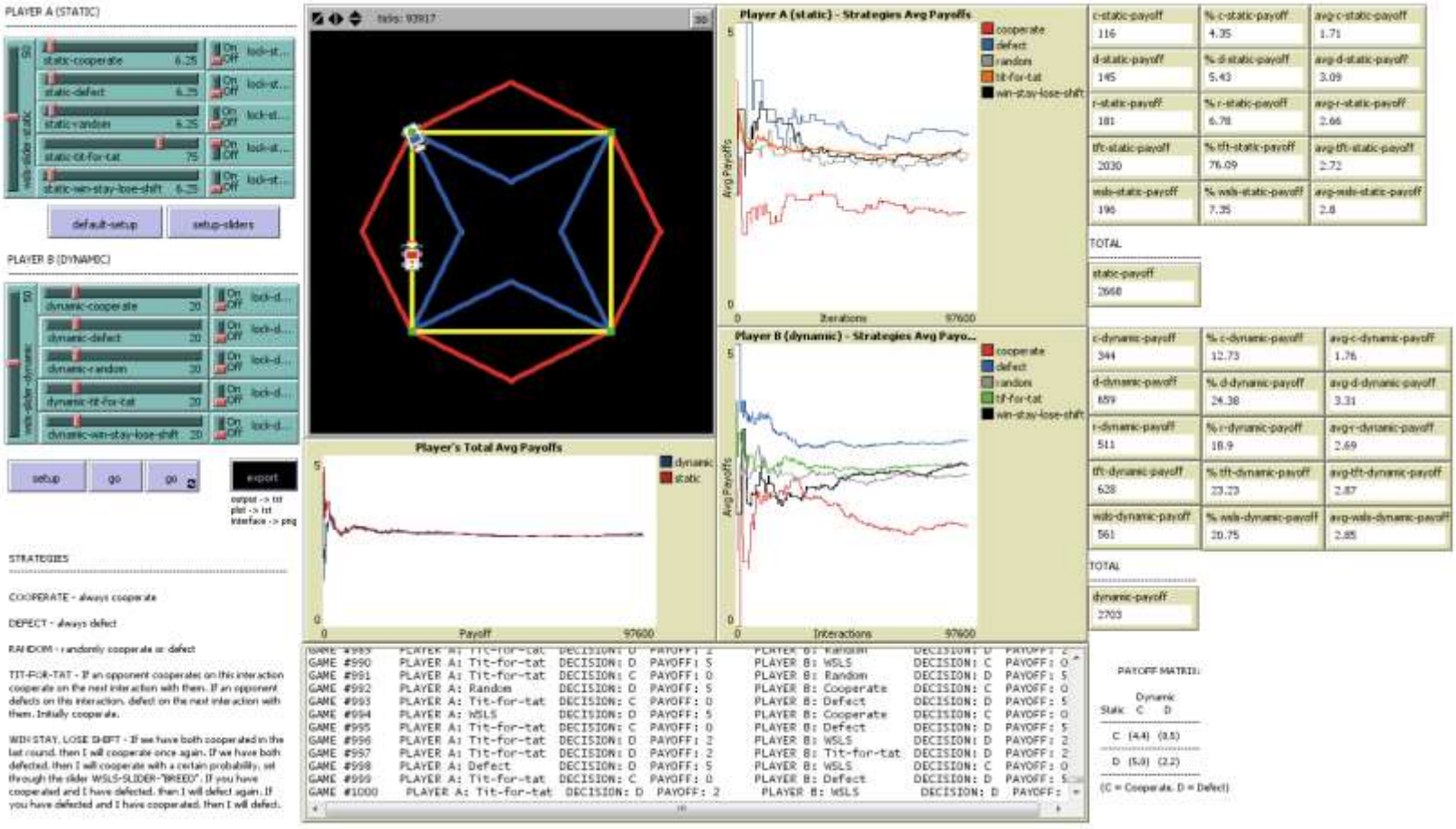

PLAYER A (STATIC)
static-cooperate
static-defect
static-random
static-tit-for-tat
static-win-stay-lose-shift
default-setup
setup-sliders
PLAYER B (DYNAMIC)
dynamic-cooperate
dynamic-defect
dynamic-random
dynamic-tit-for-tat
dynamic-win-stay-lose-shift
setup
go
export
STRATEGIES
COOPERATE - always cooperate
DEFECT - always defect
RANDOM - randomly cooperate or defect
TIT-FOR-TAT - If an opponent cooperates on the interaction cooperate on the next interaction with them. If an opponent defects on this interaction, defect on the next interaction with them. Initially cooperate.
WIN-STAY, LOSE-SHIFT - If we have both cooperated in the last round, then I will cooperate once again. If we have both defected, then I will cooperate with a certain probability, set through the slider WSLS-SLIDER-"BREED". If you have cooperated and I have defected, then I will defect again. If you have defected and I have cooperated, then I will defect.
Player A (static) - Strategies Avg Payoffs
Player B (dynamic) - Strategies Avg Payo...
Player's Total Avg Payoffs
cooperate
defect
random
tit-for-tat
win-stay-lose-shift
dynamic
static
Avg Payoffs
Iterations
Payoff
c-static-payoff
% c-static-payoff
avg-c-static-payoff
d-static-payoff
% d-static-payoff
avg-d-static-payoff
r-static-payoff
% r-static-payoff
avg-r-static-payoff
tft-static-payoff
% tft-static-payoff
avg-tft-static-payoff
wsls-static-payoff
% wsls-static-payoff
avg-wsls-static-payoff
TOTAL
static-payoff
c-dynamic-payoff
% c-dynamic-payoff
avg-c-dynamic-payoff
d-dynamic-payoff
% d-dynamic-payoff
avg-d-dynamic-payoff
r-dynamic-payoff
% r-dynamic-payoff
avg-r-dynamic-payoff
tft-dynamic-payoff
% tft-dynamic-payoff
avg-tft-dynamic-payoff
wsls-dynamic-payoff
% wsls-dynamic-payoff
avg-wsls-dynamic-payoff
TOTAL
dynamic-payoff
PAYOFF MATRIX:
Dynamic
Static C D
C (4,4) (0,5)
D (5,0) (2,2)
(C = Cooperate, D = Defect)

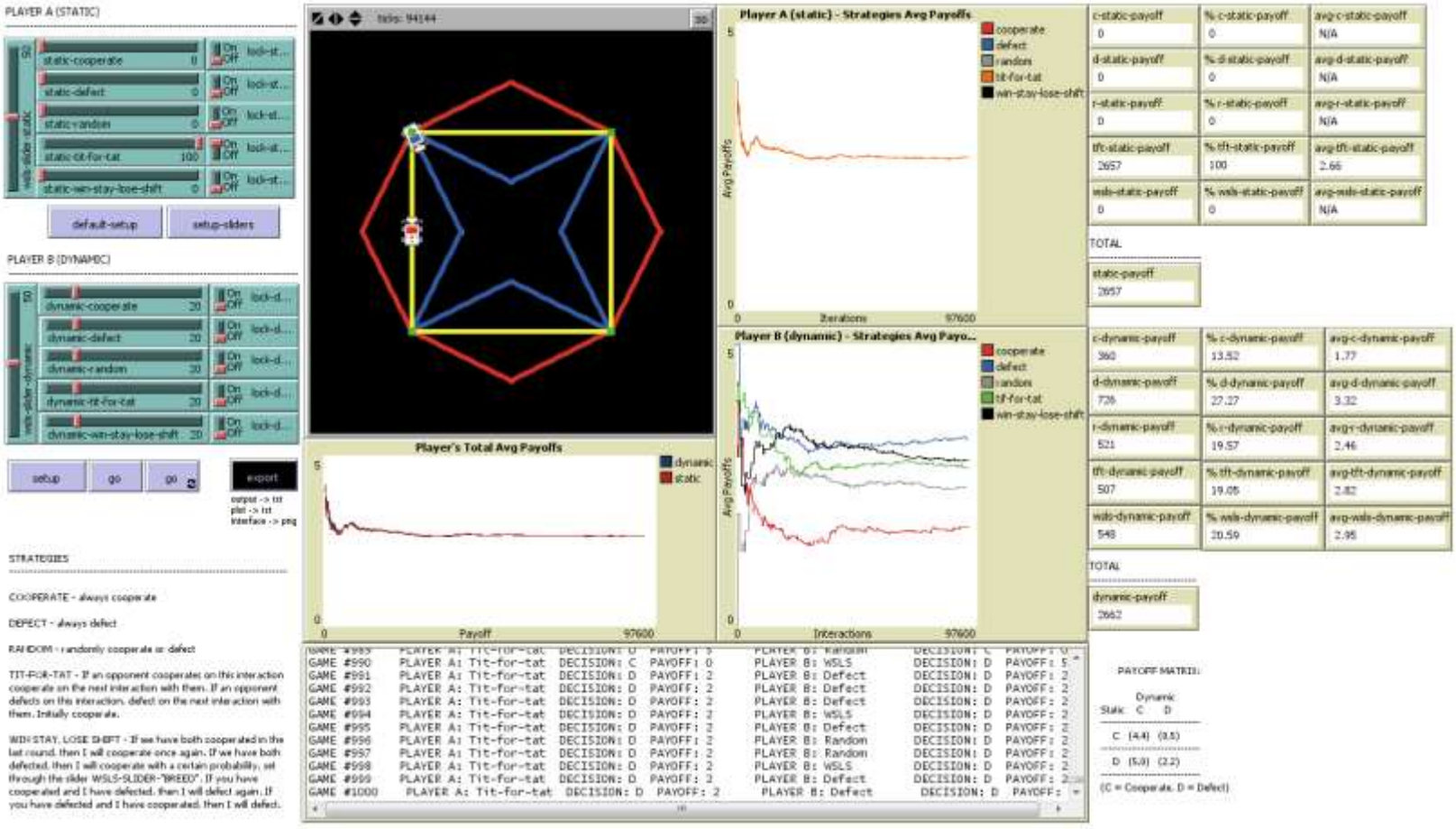

PLAYER A (STATIC)
static-cooperate
static-defect
static-random
static-tit-for-tat
static-win-stay-lose-shift
default-setup
setup-sliders
PLAYER B (DYNAMIC)
dynamic-cooperate
dynamic-defect
dynamic-random
dynamic-tit-for-tat
dynamic-win-stay-lose-shift
setup
go
export
STRATEGIES
COOPERATE - always cooperate
DEFECT - always defect
RANDOM - randomly cooperate or defect
TIT-FOR-TAT - If an opponent cooperates on the interaction cooperate on the next interaction with them. If an opponent defects on this interaction, defect on the next interaction with them. Initially cooperate.
WIN-STAY, LOSE-SHIFT - If we have both cooperated in the last round, then I will cooperate once again. If we have both defected, then I will cooperate with a certain probability, set through the slider WSLS-SLIDER-"BREED". If you have cooperated and I have defected, then I will defect again. If you have defected and I have cooperated, then I will defect.
Player A (static) - Strategies Avg Payoffs
cooperate
defect
random
tit-for-tat
win-stay-lose-shift
Avg Payoffs
Iterations
Player B (dynamic) - Strategies Avg Payo...
Player's Total Avg Payoffs
dynamic
static
Payoff
c-static-payoff
% c-static-payoff
avg-c-static-payoff
d-static-payoff
% d-static-payoff
avg-d-static-payoff
r-static-payoff
% r-static-payoff
avg-r-static-payoff
tft-static-payoff
% tft-static-payoff
avg-tft-static-payoff
wsls-static-payoff
% wsls-static-payoff
avg-wsls-static-payoff
TOTAL
static-payoff
c-dynamic-payoff
% c-dynamic-payoff
avg-c-dynamic-payoff
d-dynamic-payoff
% d-dynamic-payoff
avg-d-dynamic-payoff
r-dynamic-payoff
% r-dynamic-payoff
avg-r-dynamic-payoff
tft-dynamic-payoff
% tft-dynamic-payoff
avg-tft-dynamic-payoff
wsls-dynamic-payoff
% wsls-dynamic-payoff
avg-wsls-dynamic-payoff
TOTAL
dynamic-payoff
PAYOFF MATRIX:
Dynamic
Static C D
C (4,4) (0,5)
D (5,0) (2,2)
(C = Cooperate, D = Defect)

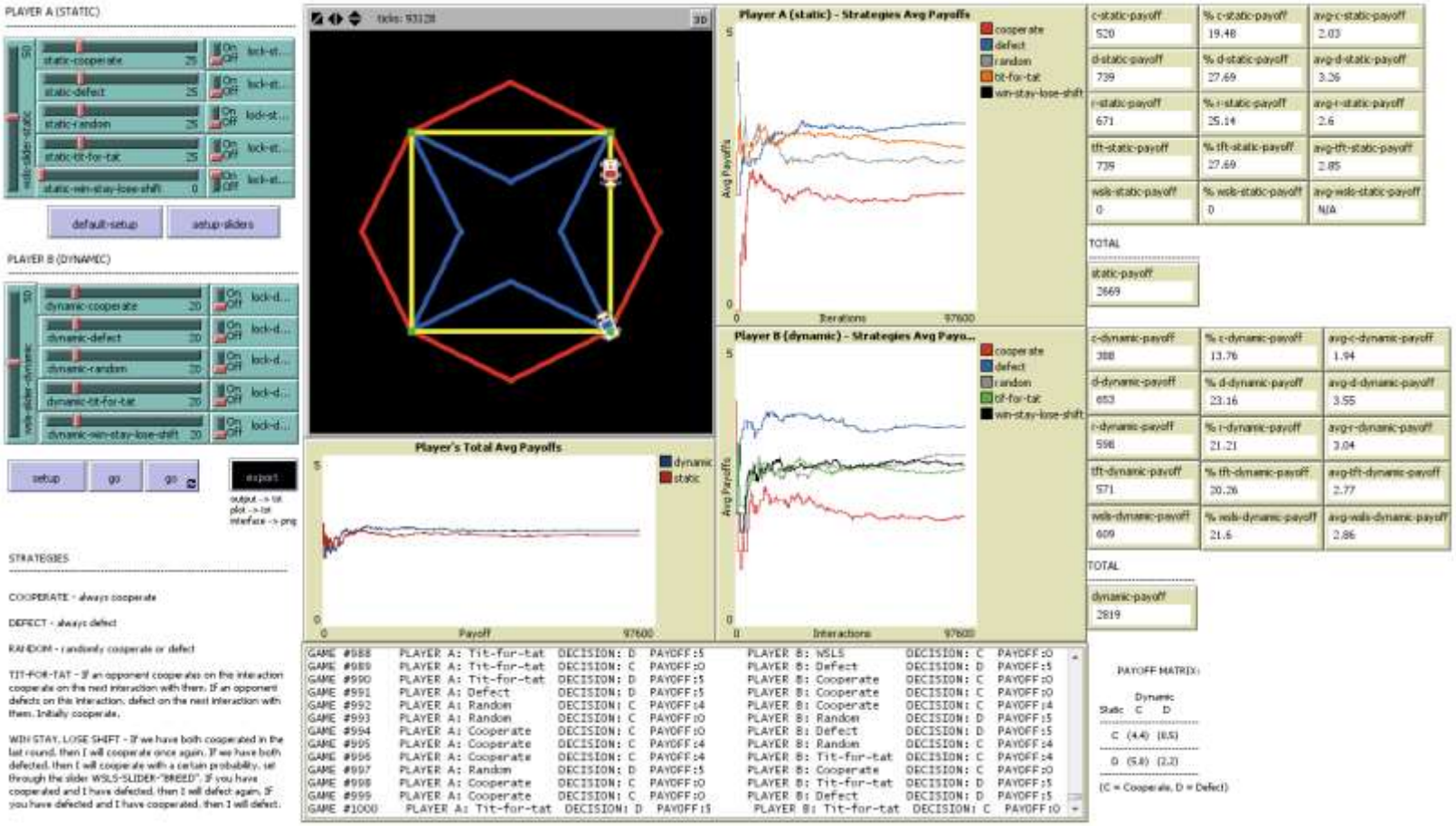

PLAYER A (STATIC)
static-cooperate
static-defect
static-random
static-tit-for-tat
static-win-stay-lose-shift
default-setup
setup-sliders
PLAYER B (DYNAMIC)
dynamic-cooperate
dynamic-defect
dynamic-random
dynamic-tit-for-tat
dynamic-win-stay-lose-shift
setup
go
export
STRATEGIES
COOPERATE - always cooperate
DEFECT - always defect
RANDOM - randomly cooperate or defect
TIT-FOR-TAT - If an opponent cooperates on the interaction cooperate on the next interaction with them. If an opponent defects on this interaction, defect on the next interaction with them. Initially cooperate.
WIN-STAY, LOSE-SHIFT - If we have both cooperated in the last round, then I will cooperate once again. If we have both defected, then I will cooperate with a certain probability, set through the slider WSLS-SLIDER-"BREED". If you have cooperated and I have defected, then I will defect again. If you have defected and I have cooperated, then I will defect.
Player A (static) - Strategies Avg Payoffs
Player B (dynamic) - Strategies Avg Payo...
Player's Total Avg Payoffs
cooperate
defect
random
tit-for-tat
win-stay-lose-shift
dynamic
static
Avg Payoffs
Iterations
Payoff
c-static-payoff
% c-static-payoff
avg-c-static-payoff
d-static-payoff
% d-static-payoff
avg-d-static-payoff
r-static-payoff
% r-static-payoff
avg-r-static-payoff
tft-static-payoff
% tft-static-payoff
avg-tft-static-payoff
wsls-static-payoff
% wsls-static-payoff
avg-wsls-static-payoff
TOTAL
static-payoff
c-dynamic-payoff
% c-dynamic-payoff
avg-c-dynamic-payoff
d-dynamic-payoff
% d-dynamic-payoff
avg-d-dynamic-payoff
r-dynamic-payoff
% r-dynamic-payoff
avg-r-dynamic-payoff
tft-dynamic-payoff
% tft-dynamic-payoff
avg-tft-dynamic-payoff
wsls-dynamic-payoff
% wsls-dynamic-payoff
avg-wsls-dynamic-payoff
TOTAL
dynamic-payoff
PAYOFF MATRIX:
Dynamic
Static C D
C (4,4) (0,5)
D (5,0) (2,2)
(C = Cooperate, D = Defect)

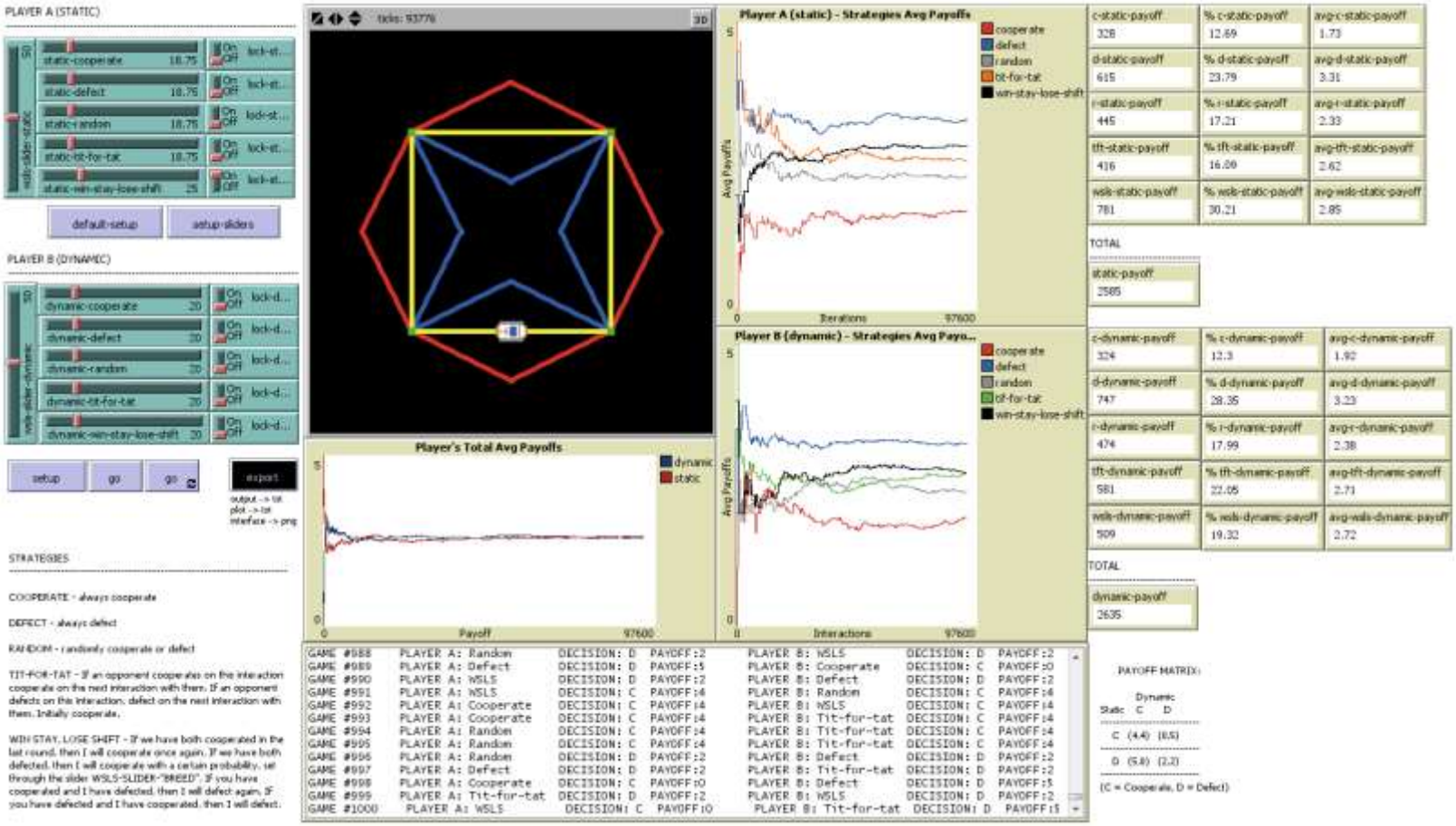

PLAYER A (STATIC)
static-cooperate
static-defect
static-random
static-tit-for-tat
static-win-stay-lose-shift
default-setup
setup-sliders
PLAYER B (DYNAMIC)
dynamic-cooperate
dynamic-defect
dynamic-random
dynamic-tit-for-tat
dynamic-win-stay-lose-shift
setup
go
export
STRATEGIES
COOPERATE - always cooperate
DEFECT - always defect
RANDOM - randomly cooperate or defect
TIT-FOR-TAT - If an opponent cooperates on the interaction cooperate on the next interaction with them. If an opponent defects on this interaction, defect on the next interaction with them. Initially cooperate.
WIN-STAY, LOSE-SHIFT - If we have both cooperated in the last round, then I will cooperate once again. If we have both defected, then I will cooperate with a certain probability, set through the slider WSLS-SLIDER-"BREED". If you have cooperated and I have defected, then I will defect again. If you have defected and I have cooperated, then I will defect.
Player A (static) - Strategies Avg Payoffs
Player B (dynamic) - Strategies Avg Payo...
Player's Total Avg Payoffs
cooperate
defect
random
tit-for-tat
win-stay-lose-shift
dynamic
static
Avg Payoffs
Iterations
Payoff
c-static-payoff
% c-static-payoff
avg-c-static-payoff
d-static-payoff
% d-static-payoff
avg-d-static-payoff
r-static-payoff
% r-static-payoff
avg-r-static-payoff
tft-static-payoff
% tft-static-payoff
avg-tft-static-payoff
wsls-static-payoff
% wsls-static-payoff
avg-wsls-static-payoff
TOTAL
static-payoff
c-dynamic-payoff
% c-dynamic-payoff
avg-c-dynamic-payoff
d-dynamic-payoff
% d-dynamic-payoff
avg-d-dynamic-payoff
r-dynamic-payoff
% r-dynamic-payoff
avg-r-dynamic-payoff
tft-dynamic-payoff
% tft-dynamic-payoff
avg-tft-dynamic-payoff
wsls-dynamic-payoff
% wsls-dynamic-payoff
avg-wsls-dynamic-payoff
TOTAL
dynamic-payoff
PAYOFF MATRIX:
Dynamic
Static C D
C (4,4) (0,5)
D (5,0) (2,2)
(C = Cooperate, D = Defect)

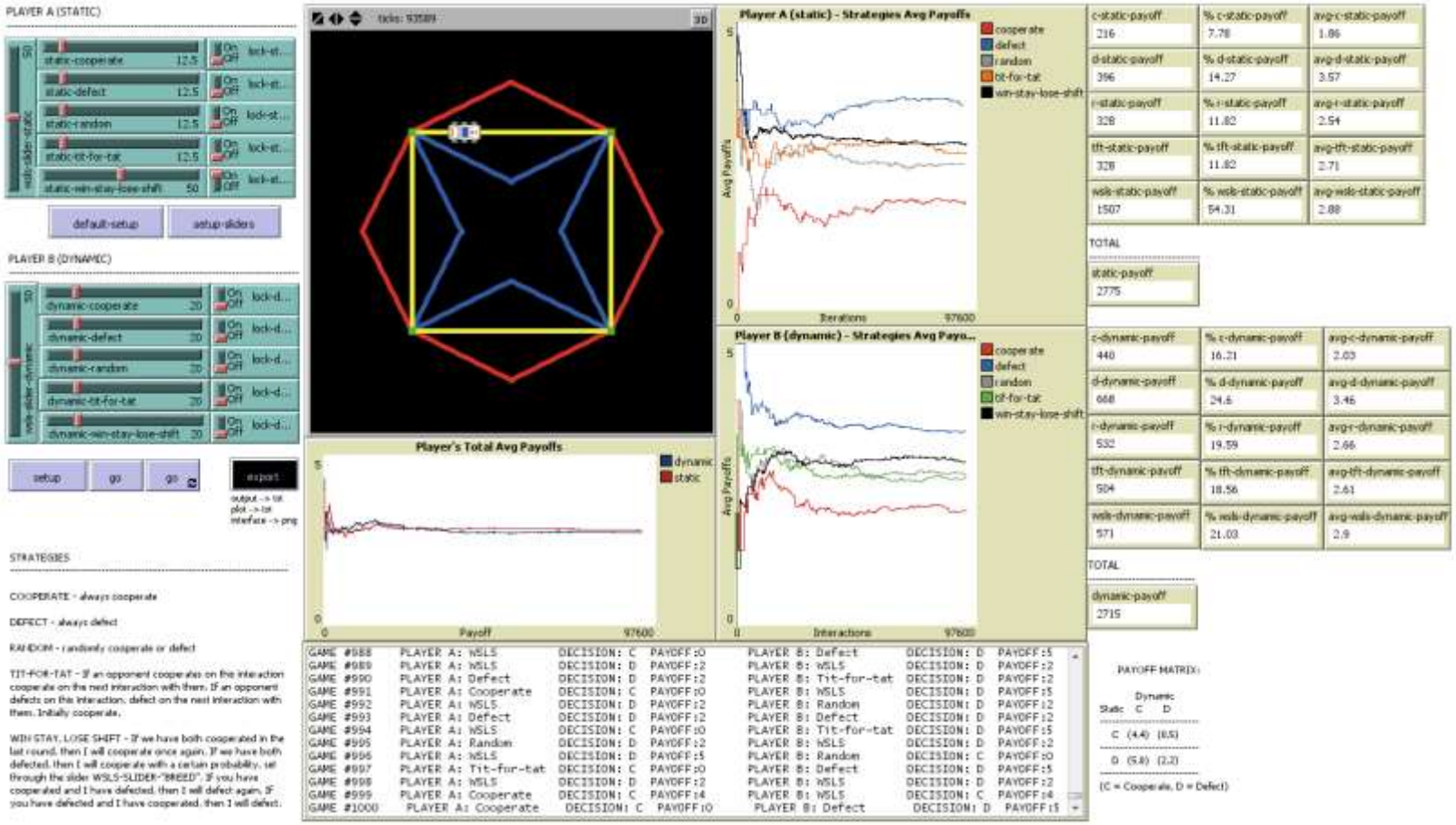

PLAYER A (STATIC)
static-cooperate 12.5
static-defect 12.5
static-random 12.5
static-tit-for-tat 50
static-win-stay-lose-shift 12.5
default-setup
setup-sliders
PLAYER B (DYNAMIC)
dynamic-cooperate 20
dynamic-defect 20
dynamic-random 20
dynamic-tit-for-tat 20
dynamic-win-stay-lose-shift 20
setup
go
export
STRATEGIES
COOPERATE - always cooperate
DEFECT - always defect
RANDOM - randomly cooperate or defect
TIT-FOR-TAT - If an opponent cooperates on this interaction cooperate on the next interaction with them. If an opponent defects on this interaction, defect on the next interaction with them. Initially cooperate.
WIN-STAY, LOSE SHIFT - If we have both cooperated in the last round, then I will cooperate once again. If we have both defected, then I will cooperate with a certain probability, set through the slider WSLS-SLIDER-"BREED". If you have cooperated and I have defected, then I will defect again. If you have defected and I have cooperated, then I will defect.
ticks: 93689
Player A (static) - Strategies Avg Payoffs
cooperate
defect
random
tit-for-tat
win-stay-lose-shift
Avg Payoffs
Iterations
0
97600
Player B (dynamic) - Strategies Avg Payo...
Player's Total Avg Payoffs
dynamic
static
Payoff
Interactions
c-static-payoff 284
% c-static-payoff 10.62
avg-c-static-payoff 2.03
d-static-payoff 366
% d-static-payoff 13.68
avg-d-static-payoff 3.3
r-static-payoff 328
% r-static-payoff 12.26
avg-r-static-payoff 2.58
tft-static-payoff 1325
% tft-static-payoff 49.53
avg-tft-static-payoff 2.67
wsls-static-payoff 372
% wsls-static-payoff 13.91
avg-wsls-static-payoff 2.98
TOTAL
static-payoff 2675
c-dynamic-payoff 384
% c-dynamic-payoff 13.79
avg-c-dynamic-payoff 1.99
d-dynamic-payoff 701
% d-dynamic-payoff 25.17
avg-d-dynamic-payoff 3.42
r-dynamic-payoff 564
% r-dynamic-payoff 20.25
avg-r-dynamic-payoff 2.71
tft-dynamic-payoff 573
% tft-dynamic-payoff 20.57
avg-tft-dynamic-payoff 2.88
wsls-dynamic-payoff 563
% wsls-dynamic-payoff 20.22
avg-wsls-dynamic-payoff 2.89
TOTAL
dynamic-payoff 2785
GAME #990 PLAYER A: Tit-for-tat DECISION: C PAYOFF: 0 PLAYER B: Tit-for-tat DECISION: D PAYOFF: 5
GAME #991 PLAYER A: Tit-for-tat DECISION: D PAYOFF: 5 PLAYER B: Tit-for-tat DECISION: C PAYOFF: 0
GAME #992 PLAYER A: Tit-for-tat DECISION: C PAYOFF: 0 PLAYER B: WSLS DECISION: D PAYOFF: 5
GAME #993 PLAYER A: Cooperate DECISION: C PAYOFF: 0 PLAYER B: WSLS DECISION: D PAYOFF: 5
GAME #994 PLAYER A: Cooperate DECISION: C PAYOFF: 0 PLAYER B: WSLS DECISION: D PAYOFF: 5
GAME #995 PLAYER A: WSLS DECISION: D PAYOFF: 2 PLAYER B: Random DECISION: D PAYOFF: 2
GAME #996 PLAYER A: Cooperate DECISION: C PAYOFF: 4 PLAYER B: Cooperate DECISION: C PAYOFF: 4
GAME #997 PLAYER A: WSLS DECISION: C PAYOFF: 4 PLAYER B: Cooperate DECISION: C PAYOFF: 4
GAME #998 PLAYER A: Tit-for-tat DECISION: C PAYOFF: 4 PLAYER B: Random DECISION: C PAYOFF: 4
GAME #999 PLAYER A: Tit-for-tat DECISION: C PAYOFF: 0 PLAYER B: Defect DECISION: D PAYOFF: 5
GAME #1000 PLAYER A: Random DECISION: D PAYOFF: 5 PLAYER B: Tit-for-tat DECISION: C PAYOFF: 0
PAYOFF MATRIX:
Dynamic
Static C D
C (4,4) (0,5)
D (5,0) (2,2)
(C = Cooperate, D = Defect)

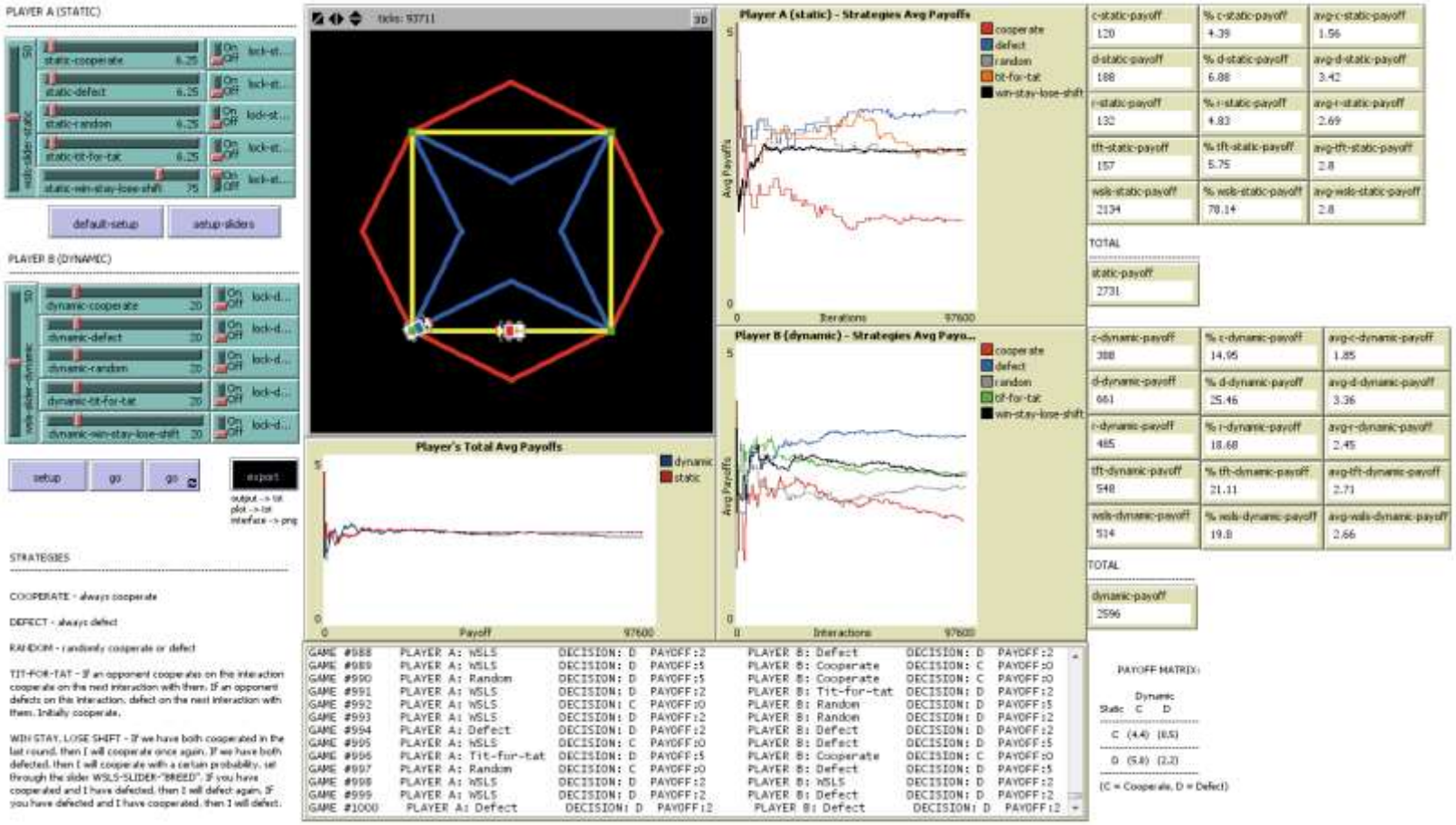

PLAYER A (STATIC)
static-cooperate 6.25
static-defect 6.25
static-random 6.25
static-tit-for-tat 75
static-win-stay-lose-shift 6.25
default-setup
setup-sliders
PLAYER B (DYNAMIC)
dynamic-cooperate 20
dynamic-defect 20
dynamic-random 20
dynamic-tit-for-tat 20
dynamic-win-stay-lose-shift 20
setup
go
go
export
output -> txt
plot -> txt
interface -> png
STRATEGIES
COOPERATE - always cooperate
DEFECT - always defect
RANDOM - randomly cooperate or defect
TIT-FOR-TAT - If an opponent cooperates on this interaction cooperate on the next interaction with them. If an opponent defects on this interaction, defect on the next interaction with them. Initially cooperate.
WIN-STAY, LOSE SHIFT - If we have both cooperated in the last round, then I will cooperate once again. If we have both defected, then I will cooperate with a certain probability, set through the slider WSLS-SLIDER-"BREED". If you have cooperated and I have defected, then I will defect again. If you have defected and I have cooperated, then I will defect.
ticks: 93917
Player A (static) - Strategies Avg Payoffs
cooperate
defect
random
tit-for-tat
win-stay-lose-shift
Avg Payoffs
Iterations
0
97600
Player B (dynamic) - Strategies Avg Payo...
Player's Total Avg Payoffs
dynamic
static
Payoff
Interactions
c-static-payoff 116
% c-static-payoff 4.35
avg-c-static-payoff 1.71
d-static-payoff 145
% d-static-payoff 5.43
avg-d-static-payoff 3.09
r-static-payoff 181
% r-static-payoff 6.78
avg-r-static-payoff 2.66
tft-static-payoff 2030
% tft-static-payoff 76.09
avg-tft-static-payoff 2.72
wsls-static-payoff 196
% wsls-static-payoff 7.35
avg-wsls-static-payoff 2.8
TOTAL
static-payoff 2668
c-dynamic-payoff 344
% c-dynamic-payoff 12.73
avg-c-dynamic-payoff 1.76
d-dynamic-payoff 659
% d-dynamic-payoff 24.38
avg-d-dynamic-payoff 3.31
r-dynamic-payoff 511
% r-dynamic-payoff 18.9
avg-r-dynamic-payoff 2.69
tft-dynamic-payoff 628
% tft-dynamic-payoff 23.23
avg-tft-dynamic-payoff 2.87
wsls-dynamic-payoff 561
% wsls-dynamic-payoff 20.75
avg-wsls-dynamic-payoff 2.85
TOTAL
dynamic-payoff 2703
PAYOFF MATRIX:
Dynamic
Static C D
C (4,4) (0,5)
D (5,0) (2,2)
(C = Cooperate, D = Defect)
GAME #990 PLAYER A: Tit-for-tat DECISION: D PAYOFF: 5 PLAYER B: WSLS DECISION: C PAYOFF: 0
GAME #991 PLAYER A: Tit-for-tat DECISION: C PAYOFF: 0 PLAYER B: Random DECISION: D PAYOFF: 5
GAME #992 PLAYER A: Random DECISION: D PAYOFF: 5 PLAYER B: Cooperate DECISION: C PAYOFF: 0
GAME #993 PLAYER A: Tit-for-tat DECISION: C PAYOFF: 0 PLAYER B: Defect DECISION: D PAYOFF: 5
GAME #994 PLAYER A: WSLS DECISION: D PAYOFF: 5 PLAYER B: Cooperate DECISION: C PAYOFF: 0
GAME #995 PLAYER A: Tit-for-tat DECISION: C PAYOFF: 0 PLAYER B: Defect DECISION: D PAYOFF: 5
GAME #996 PLAYER A: Tit-for-tat DECISION: D PAYOFF: 2 PLAYER B: WSLS DECISION: D PAYOFF: 2
GAME #997 PLAYER A: Tit-for-tat DECISION: D PAYOFF: 2 PLAYER B: Tit-for-tat DECISION: D PAYOFF: 2
GAME #998 PLAYER A: Defect DECISION: D PAYOFF: 5 PLAYER B: WSLS DECISION: C PAYOFF: 0
GAME #999 PLAYER A: Tit-for-tat DECISION: C PAYOFF: 0 PLAYER B: Defect DECISION: D PAYOFF: 5
GAME #1000 PLAYER A: Tit-for-tat DECISION: D PAYOFF: 2 PLAYER B: WSLS DECISION: D PAYOFF:

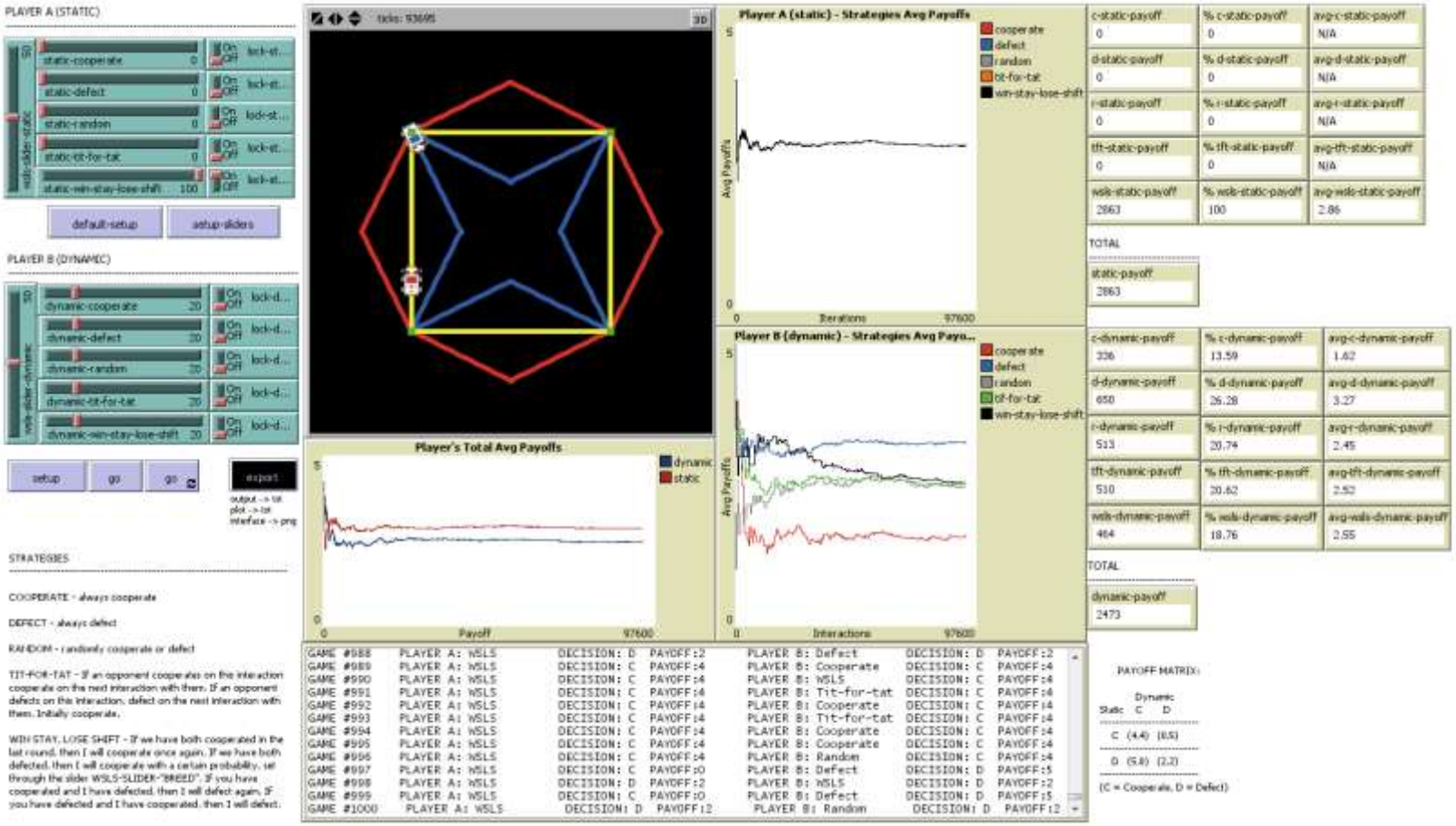

PLAYER A (STATIC)
static-cooperate
static-defect
static-random
static-tit-for-tat
static-win-stay-lose-shift
default-setup
setup-sliders
PLAYER B (DYNAMIC)
dynamic-cooperate
dynamic-defect
dynamic-random
dynamic-tit-for-tat
dynamic-win-stay-lose-shift
setup
go
export
STRATEGIES
COOPERATE - always cooperate
DEFECT - always defect
RANDOM - randomly cooperate or defect
TIT-FOR-TAT - If an opponent cooperates on this interaction cooperate on the next interaction with them. If an opponent defects on this interaction, defect on the next interaction with them. Initially cooperate.
WIN-STAY, LOSE SHIFT - If we have both cooperated in the last round, then I will cooperate once again. If we have both defected, then I will cooperate with a certain probability, set through the slider WSLS-SLIDER-"BREED". If you have cooperated and I have defected, then I will defect again. If you have defected and I have cooperated, then I will defect.
Player A (static) - Strategies Avg Payoffs
Player B (dynamic) - Strategies Avg Payo...
Player's Total Avg Payoffs
cooperate
defect
random
tit-for-tat
win-stay-lose-shift
dynamic
static
Avg Payoffs
Iterations
Payoff
Interactions
c-static-payoff
% c-static-payoff
avg-c-static-payoff
d-static-payoff
% d-static-payoff
avg-d-static-payoff
r-static-payoff
% r-static-payoff
avg-r-static-payoff
tft-static-payoff
% tft-static-payoff
avg-tft-static-payoff
wsls-static-payoff
% wsls-static-payoff
avg-wsls-static-payoff
TOTAL
static-payoff
c-dynamic-payoff
% c-dynamic-payoff
avg-c-dynamic-payoff
d-dynamic-payoff
% d-dynamic-payoff
avg-d-dynamic-payoff
r-dynamic-payoff
% r-dynamic-payoff
avg-r-dynamic-payoff
tft-dynamic-payoff
% tft-dynamic-payoff
avg-tft-dynamic-payoff
wsls-dynamic-payoff
% wsls-dynamic-payoff
avg-wsls-dynamic-payoff
TOTAL
dynamic-payoff
PAYOFF MATRIX:
(C = Cooperate, D = Defect)


PLAYER A (STATIC)
static-cooperate
static-defect
static-random
static-tit-for-tat
static-win-stay-lose-shift
default-setup
setup-sliders
PLAYER B (DYNAMIC)
dynamic-cooperate
dynamic-defect
dynamic-random
dynamic-tit-for-tat
dynamic-win-stay-lose-shift
setup
go
export
STRATEGIES
COOPERATE - always cooperate
DEFECT - always defect
RANDOM - randomly cooperate or defect
TIT-FOR-TAT - IF an opponent cooperates on the interaction cooperate on the next interaction with them. IF an opponent defects on this interaction, defect on the next interaction with them. Initially cooperate.
WIN-STAY, LOSE-SHIFT - IF we have both cooperated in the last round, then I will cooperate once again. IF we have both defected, then I will cooperate with a certain probability, set through the slider WSLS-SLIDER-"BREED". IF you have cooperated and I have defected, then I will defect again. IF you have defected and I have cooperated, then I will defect.
Player A (static) - Strategies Avg Payoffs
Player B (dynamic) - Strategies Avg Payo...
Player's Total Avg Payoffs
cooperate
defect
random
tit-for-tat
win-stay-lose-shift
dynamic
static
Avg Payoffs
Iterations
Payoff
c-static-payoff 520
% c-static-payoff 19.48
avg-c-static-payoff 2.03
d-static-payoff 739
% d-static-payoff 27.69
avg-d-static-payoff 3.26
r-static-payoff 671
% r-static-payoff 25.14
avg-r-static-payoff 2.6
tft-static-payoff 739
% tft-static-payoff 27.69
avg-tft-static-payoff 2.85
wsls-static-payoff 0
% wsls-static-payoff 0
avg-wsls-static-payoff N/A
TOTAL
static-payoff 2669
c-dynamic-payoff 388
% c-dynamic-payoff 13.76
avg-c-dynamic-payoff 1.94
d-dynamic-payoff 653
% d-dynamic-payoff 23.16
avg-d-dynamic-payoff 3.55
r-dynamic-payoff 598
% r-dynamic-payoff 21.21
avg-r-dynamic-payoff 3.04
tft-dynamic-payoff 571
% tft-dynamic-payoff 20.26
avg-tft-dynamic-payoff 2.77
wsls-dynamic-payoff 609
% wsls-dynamic-payoff 21.6
avg-wsls-dynamic-payoff 2.86
TOTAL
dynamic-payoff 2819
PAYOFF MATRIX:
Dynamic
Static C D
C (4,4) (0,5)
D (5,0) (2,2)
(C = Cooperate, D = Defect)


PLAYER A (STATIC)
static-cooperate 18.75
static-defect 18.75
static-random 18.75
static-tit-for-tat 18.75
static-win-stay-lose-shift 25
default-setup
setup-sliders
PLAYER B (DYNAMIC)
dynamic-cooperate 20
dynamic-defect 20
dynamic-random 20
dynamic-tit-for-tat 20
dynamic-win-stay-lose-shift 20
setup
go
export
STRATEGIES
COOPERATE - always cooperate
DEFECT - always defect
RANDOM - randomly cooperate or defect
TIT-FOR-TAT - IF an opponent cooperates on the interaction cooperate on the next interaction with them. IF an opponent defects on this interaction, defect on the next interaction with them. Initially cooperate.
WIN-STAY, LOSE-SHIFT - IF we have both cooperated in the last round, then I will cooperate once again. IF we have both defected, then I will cooperate with a certain probability, set through the slider WSLS-SLIDER-"BREED". IF you have cooperated and I have defected, then I will defect again. IF you have defected and I have cooperated, then I will defect.
ticks: 93776
Player A (static) - Strategies Avg Payoffs
cooperate
defect
random
tit-for-tat
win-stay-lose-shift
Avg Payoffs
Iterations
Player's Total Avg Payoffs
dynamic
static
Payoff
Player B (dynamic) - Strategies Avg Payo...
c-static-payoff 328
% c-static-payoff 12.69
avg-c-static-payoff 1.73
d-static-payoff 615
% d-static-payoff 23.79
avg-d-static-payoff 3.31
r-static-payoff 445
% r-static-payoff 17.21
avg-r-static-payoff 2.33
tft-static-payoff 416
% tft-static-payoff 16.09
avg-tft-static-payoff 2.62
wsls-static-payoff 781
% wsls-static-payoff 30.21
avg-wsls-static-payoff 2.85
TOTAL
static-payoff 2585
c-dynamic-payoff 324
% c-dynamic-payoff 12.3
avg-c-dynamic-payoff 1.92
d-dynamic-payoff 747
% d-dynamic-payoff 28.35
avg-d-dynamic-payoff 3.23
r-dynamic-payoff 474
% r-dynamic-payoff 17.99
avg-r-dynamic-payoff 2.38
tft-dynamic-payoff 581
% tft-dynamic-payoff 22.05
avg-tft-dynamic-payoff 2.71
wsls-dynamic-payoff 509
% wsls-dynamic-payoff 19.32
avg-wsls-dynamic-payoff 2.72
TOTAL
dynamic-payoff 2635
GAME #988 PLAYER A: Random DECISION: D PAYOFF:2 PLAYER B: WSLS DECISION: D PAYOFF:2
GAME #989 PLAYER A: Defect DECISION: D PAYOFF:5 PLAYER B: Cooperate DECISION: C PAYOFF:0
GAME #990 PLAYER A: WSLS DECISION: D PAYOFF:2 PLAYER B: Defect DECISION: D PAYOFF:2
GAME #991 PLAYER A: WSLS DECISION: C PAYOFF:4 PLAYER B: Random DECISION: C PAYOFF:4
GAME #992 PLAYER A: Cooperate DECISION: C PAYOFF:4 PLAYER B: WSLS DECISION: C PAYOFF:4
GAME #993 PLAYER A: Cooperate DECISION: C PAYOFF:4 PLAYER B: Tit-for-tat DECISION: C PAYOFF:4
GAME #994 PLAYER A: Random DECISION: C PAYOFF:4 PLAYER B: Tit-for-tat DECISION: C PAYOFF:4
GAME #995 PLAYER A: Random DECISION: C PAYOFF:4 PLAYER B: Tit-for-tat DECISION: C PAYOFF:4
GAME #996 PLAYER A: Random DECISION: D PAYOFF:2 PLAYER B: Defect DECISION: D PAYOFF:2
GAME #997 PLAYER A: Defect DECISION: D PAYOFF:2 PLAYER B: Tit-for-tat DECISION: D PAYOFF:2
GAME #998 PLAYER A: Cooperate DECISION: C PAYOFF:0 PLAYER B: Defect DECISION: D PAYOFF:5
GAME #999 PLAYER A: Tit-for-tat DECISION: D PAYOFF:2 PLAYER B: WSLS DECISION: D PAYOFF:2
GAME #1000 PLAYER A: WSLS DECISION: C PAYOFF:0 PLAYER B: Tit-for-tat DECISION: D PAYOFF:5
PAYOFF MATRIX:
Dynamic
Static C D
C (4,4) (0,5)
D (5,0) (2,2)
(C = Cooperate, D = Defect)


PLAYER A (STATIC)
static-cooperate
static-defect
static-random
static-tit-for-tat
static-win-stay-lose-shift
default-setup
setup-sliders
PLAYER B (DYNAMIC)
dynamic-cooperate
dynamic-defect
dynamic-random
dynamic-tit-for-tat
dynamic-win-stay-lose-shift
setup
go
export
STRATEGIES
COOPERATE - always cooperate
DEFECT - always defect
RANDOM - randomly cooperate or defect
TIT-FOR-TAT - If an opponent cooperates on the interaction cooperate on the next interaction with them. If an opponent defects on this interaction, defect on the next interaction with them. Initially cooperate.
WIN-STAY, LOSE-SHIFT - If we have both cooperated in the last round, then I will cooperate once again. If we have both defected, then I will cooperate with a certain probability, set through the slider WSLS-SLIDER-"BREED". If you have cooperated and I have defected, then I will defect again. If you have defected and I have cooperated, then I will defect.
Player A (static) - Strategies Avg Payoffs
Player B (dynamic) - Strategies Avg Payo...
Player's Total Avg Payoffs
Avg Payoffs
Iterations
Payoff
cooperate
defect
random
tit-for-tat
win-stay-lose-shift
dynamic
static
c-static-payoff
% c-static-payoff
avg-c-static-payoff
d-static-payoff
% d-static-payoff
avg-d-static-payoff
r-static-payoff
% r-static-payoff
avg-r-static-payoff
tft-static-payoff
% tft-static-payoff
avg-tft-static-payoff
wsls-static-payoff
% wsls-static-payoff
avg-wsls-static-payoff
TOTAL
static-payoff
c-dynamic-payoff
% c-dynamic-payoff
avg-c-dynamic-payoff
d-dynamic-payoff
% d-dynamic-payoff
avg-d-dynamic-payoff
r-dynamic-payoff
% r-dynamic-payoff
avg-r-dynamic-payoff
tft-dynamic-payoff
% tft-dynamic-payoff
avg-tft-dynamic-payoff
wsls-dynamic-payoff
% wsls-dynamic-payoff
avg-wsls-dynamic-payoff
TOTAL
dynamic-payoff
PAYOFF MATRIX:
Dynamic
Static C D
C (4,4) (0,5)
D (5,0) (2,2)
(C = Cooperate, D = Defect)


PLAYER A (STATIC)
static-cooperate
static-defect
static-random
static-tit-for-tat
static-win-stay-lose-shift
default-setup
setup-sliders
PLAYER B (DYNAMIC)
dynamic-cooperate
dynamic-defect
dynamic-random
dynamic-tit-for-tat
dynamic-win-stay-lose-shift
setup
go
export
STRATEGIES
COOPERATE - always cooperate
DEFECT - always defect
RANDOM - randomly cooperate or defect
TIT-FOR-TAT - If an opponent cooperates on the interaction cooperate on the next interaction with them. If an opponent defects on this interaction, defect on the next interaction with them. Initially cooperate.
WIN-STAY, LOSE-SHIFT - If we have both cooperated in the last round, then I will cooperate once again. If we have both defected, then I will cooperate with a certain probability, set through the slider WSLS-SLIDER-"BREED". If you have cooperated and I have defected, then I will defect again. If you have defected and I have cooperated, then I will defect.
Player A (static) - Strategies Avg Payoffs
Player B (dynamic) - Strategies Avg Payo...
Player's Total Avg Payoffs
cooperate
defect
random
tit-for-tat
win-stay-lose-shift
dynamic
static
Iterations
Payoff
c-static-payoff
% c-static-payoff
avg-c-static-payoff
d-static-payoff
% d-static-payoff
avg-d-static-payoff
r-static-payoff
% r-static-payoff
avg-r-static-payoff
tft-static-payoff
% tft-static-payoff
avg-tft-static-payoff
wsls-static-payoff
% wsls-static-payoff
avg-wsls-static-payoff
TOTAL
static-payoff
c-dynamic-payoff
% c-dynamic-payoff
avg-c-dynamic-payoff
d-dynamic-payoff
% d-dynamic-payoff
avg-d-dynamic-payoff
r-dynamic-payoff
% r-dynamic-payoff
avg-r-dynamic-payoff
tft-dynamic-payoff
% tft-dynamic-payoff
avg-tft-dynamic-payoff
wsls-dynamic-payoff
% wsls-dynamic-payoff
avg-wsls-dynamic-payoff
TOTAL
dynamic-payoff
PAYOFF MATRIX:
Dynamic
Static C D
C (4,4) (0,5)
D (5,0) (2,2)
(C = Cooperate, D = Defect)


PLAYER A (STATIC)
static-cooperate
static-defect
static-random
static-tit-for-tat
static-win-stay-lose-shift
default-setup
setup-sliders
PLAYER B (DYNAMIC)
dynamic-cooperate
dynamic-defect
dynamic-random
dynamic-tit-for-tat
dynamic-win-stay-lose-shift
setup
go
export
STRATEGIES
COOPERATE - always cooperate
DEFECT - always defect
RANDOM - randomly cooperate or defect
TIT-FOR-TAT - If an opponent cooperates on the interaction cooperate on the next interaction with them. If an opponent defects on this interaction, defect on the next interaction with them. Initially cooperate.
WIN-STAY, LOSE-SHIFT - If we have both cooperated in the last round, then I will cooperate once again. If we have both defected, then I will cooperate with a certain probability, set through the slider WSLS-SLIDER-"BREED". If you have cooperated and I have defected, then I will defect again. If you have defected and I have cooperated, then I will defect.
Player A (static) - Strategies Avg Payoffs
Player B (dynamic) - Strategies Avg Payo...
Player's Total Avg Payoffs
cooperate
defect
random
tit-for-tat
win-stay-lose-shift
dynamic
static
Iterations
Payoff
c-static-payoff
% c-static-payoff
avg-c-static-payoff
d-static-payoff
% d-static-payoff
avg-d-static-payoff
r-static-payoff
% r-static-payoff
avg-r-static-payoff
tft-static-payoff
% tft-static-payoff
avg-tft-static-payoff
wsls-static-payoff
% wsls-static-payoff
avg-wsls-static-payoff
TOTAL
static-payoff
c-dynamic-payoff
% c-dynamic-payoff
avg-c-dynamic-payoff
d-dynamic-payoff
% d-dynamic-payoff
avg-d-dynamic-payoff
r-dynamic-payoff
% r-dynamic-payoff
avg-r-dynamic-payoff
tft-dynamic-payoff
% tft-dynamic-payoff
avg-tft-dynamic-payoff
wsls-dynamic-payoff
% wsls-dynamic-payoff
avg-wsls-dynamic-payoff
TOTAL
dynamic-payoff
PAYOFF MATRIX:
(C = Cooperate, D = Defect)

Appendix C – Lego NXT Code

Master Robot (Also known as Player A or Static)

```
#include "NXCDefs.h"
#include "C:\Program Files (x86)\BricxCC\BTlib\BTlib.nxc"
#define DISTANCE_THRESHOLD 10
#define NUM_STRATEGY 100
#define PROB_COOPERATE 50
#define BT_CONN 1
#define MAILBOX 0
#define FILE_NAME "sim.txt"
#define FILE_SIZE 6144

//here initialize all the "sliders"
#define STATIC_COOPERATE 6.25
#define STATIC_DEFECT 6.25
#define STATIC_RANDOM 75
#define STATIC_TIT_FOR_TAT 6.25
#define STATIC_WIN_STAY_LOSE_SHIFT 6.25


//OUT_C right motor
//OUT_B left motor

task WatchAhead();
task Move();
task WatchBumper();

/* Functions Prototypes */
void play_a_round();
void go_back();
sub masterBTCheck(int conn);
int GetColorValue(const byte port);
int defect();
int cooperate();
int random_strategy();
int tit_for_tat();
int win_stay_lose_shift();
void set_payoffs( int my_s, int slave_s);
string m1 = "Master pay = ";
string s1 = "Slave pay = ";
string newfile = "-----------------------------------------------------";

int screen_x, motor_pwr = 75, move_rule, MASTER_payoff = 0, SLAVE_payoff =
0, previous_move[2], long_way = 0;
bool bt_received;
mutex motorMutex;

task main()
{
SetSensorColorFull(S3);
SetSensorLowspeed(S4);
SetSensorTouch(IN_1);
SetSensorTouch(IN_2);
previous_move[0] = 1;
previous_move[1] = 1;
Precedes(WatchAhead, WatchBumper, Move);
}

/*-------------------------------------------------------------*/

task WatchAhead()
     {
      while (true)
        {
         if ( SensorUS( S4 ) < DISTANCE_THRESHOLD )
           {
```

```
            motor_pwr = 35;
           }
        }
     }

/*--------------------------------------------------------------*/

task Move()
   {
    while (true)
       {
        if (long_way == 0)
           {
            switch(GetColorValue(S3))
              {
               case INPUT_YELLOWCOLOR:  Acquire(motorMutex);
                                        Off(OUT_B);
                                        OnFwd(OUT_C, motor_pwr);
                                        Release(motorMutex);
                                        break;

               case INPUT_WHITECOLOR:   Acquire(motorMutex);
                                        Off(OUT_B);
                                        OnFwd(OUT_C, motor_pwr);
                                        Release(motorMutex);
                                        break;

               case INPUT_BLACKCOLOR:   Acquire(motorMutex);
                                        Off(OUT_C);
                                        OnFwd(OUT_B, motor_pwr);
                                        Release(motorMutex);
                                        break;

              case INPUT_GREENCOLOR:   motor_pwr = 75;
                                       PlayTone(440, 1000);
                                       Acquire(motorMutex);
                                       OnFwd(OUT_BC, motor_pwr);
                                       Release(motorMutex);
                                       break;

              /* not really needed, but we tell this one to react as a white
color to preserve the right movements */
              case INPUT_REDCOLOR:     Acquire(motorMutex);
                                       Off(OUT_C);
                                       OnFwd(OUT_B, motor_pwr);
                                       Release(motorMutex);
                                       break;
              }
           }
        else
            {
              switch(GetColorValue(S3))
              {
               case INPUT_BLACKCOLOR:     Acquire(motorMutex);
                                          Off(OUT_C);
                                          OnFwd(OUT_B, motor_pwr);
                                          Release(motorMutex);
                                          break;

               case INPUT_REDCOLOR:       Acquire(motorMutex);
                                          Off(OUT_B);
                                          OnFwd(OUT_C, motor_pwr);
                                          Release(motorMutex);
                                          break;

               case INPUT_GREENCOLOR:     long_way = 0;
                                          break;
               }
             }
       }  /*end while */
 } //end task

/*--------------------------------------------------------------*/

// move rule
// 0- master goes back, slave forward
```

```
// 1- master goes forward, slave back
// 2- both go back

task WatchBumper()
     {
      int finished = 0;
      while (true)
         {
          if (SENSOR_1 == 1 || SENSOR_2 == 1)
            {
             play_a_round();
             Acquire(motorMutex);
             switch(move_rule)
                {
                 case 0:
                 case 2:      go_back();
                              break;

                 default:     Off(OUT_BC);
                              Wait(7000);
                              motor_pwr = 75;
                              break;
                }
             Release(motorMutex);
            }
          }
      }


/*-------------------------------------------------------------*/

void go_back()
   {
    int finished = 0;
    OnRev(OUT_BC, 40);
    finished = 0;
    while (finished == 0)
        {
         Wait(1);
         if ( GetColorValue(S3) == INPUT_GREENCOLOR )
            {
             finished = 1;
             long_way = 1;
             motor_pwr = 75;
            }
         }
         OnFwd(OUT_BC, motor_pwr);
         Wait(200);
         OnFwd(OUT_C, motor_pwr);
         OnRev(OUT_B, motor_pwr);
         Wait(700);
   }

/*-------------------------------------------------------------*/

void play_a_round()
   {
    int result_static = Random(NUM_STRATEGY);
    int SLAVE_strategy, slave_strategy_id;
    string str_strategy, str_slave_strategy,  my_decision,
str_strategy_of_master;
    int  strategy,cont;
    unsigned long length;
    byte handle;
    unsigned int result;
    const int file_size = FILE_SIZE;
    string move_rule_result, in, in1;

    result = CreateFile(FILE_NAME, file_size, handle);
    if (result == LDR_FILEEXISTS)
       {
       result = OpenFileAppend(FILE_NAME, file_size, handle);
       ClearScreen();
       TextOut (0,LCD_LINE3 , "File gia esistente");
       WriteLnString(handle, newfile, length);
       }
```

```
    if (result == LDR_SUCCESS)
       {
       ClearScreen();
       TextOut(0, LCD_LINE4 ,"File aperto ok");
       }


    if (result_static < STATIC_COOPERATE)
      result_static = 1;
    else
      if (result_static < (STATIC_COOPERATE + STATIC_DEFECT))
        result_static = 0;
      else
        if (result_static < (STATIC_COOPERATE + STATIC_DEFECT +
STATIC_RANDOM))
          result_static = 3;
        else
          if (result_static < (STATIC_COOPERATE + STATIC_DEFECT +
STATIC_RANDOM + STATIC_TIT_FOR_TAT))
            result_static = 2;
          else
             result_static = 4;

   if (result_static == 0)
     {
     strategy = defect();
     strcpy(str_strategy, "defect");
     }

   if (result_static == 1)
     {
     strategy = cooperate();
     strcpy(str_strategy, "cooperate");
     }

   if (result_static == 2)
     {
     strategy = tit_for_tat();
     strcpy(str_strategy, "tit-for-tat");
     }

   if (result_static == 3)
     {
     strategy = random_strategy();
     strcpy(str_strategy, "random");
     }

   if (result_static == 4)
     {
     strategy = win_stay_lose_shift();
     strcpy(str_strategy, "win-stay-lose-shift");
     }


      previous_move[1] = strategy;

      /* here the slave has to send its strategy result to the MASTER who
will calculate the move-rule */

      masterBTCheck(BT_CONN);  /* checking if connected through Bluetooth */
      //sending MASTER strategy

       str_strategy_of_master = NumToStr(strategy);
       BTSendMessage(BT_CONN, MAILBOX, str_strategy_of_master);

      //receiving the slave strategy for output file

       bt_received = FALSE;
      while ( bt_received == FALSE )
         {
          in1 = BTReceiveMessage(BT_CONN, MAILBOX, TRUE);
          if (strlen(in1) > 0)
              bt_received = TRUE;
          }
```

```
slave_strategy_id = StrToNum(in1);


if (slave_strategy_id == 0)
   strcpy(str_slave_strategy, "defect");

if (slave_strategy_id == 1)
   strcpy(str_slave_strategy, "cooperate");

if (slave_strategy_id == 2)
   strcpy(str_slave_strategy, "tit-for-tat");

if (slave_strategy_id == 3)
   strcpy(str_slave_strategy, "random");

if (slave_strategy_id == 4)
   strcpy(str_slave_strategy, "WSLS");

/* I've to receive the message by repetetly polling the mailbox */
bt_received = FALSE;
while ( bt_received == FALSE )
   {
    in = BTReceiveMessage(BT_CONN, MAILBOX, TRUE);
    if (strlen(in) > 0)
        bt_received = TRUE;
   }
bt_received = FALSE;
SLAVE_strategy = StrToNum(in);
previous_move[0] = SLAVE_strategy;
// here i got my result and the SLAVE result
if ( (strategy == 0) && (SLAVE_strategy == 0) )
   move_rule = 2;
if ( (strategy == 1) && (SLAVE_strategy == 0) )
   move_rule = 0;
if ( (strategy == 0) && (SLAVE_strategy == 1) )
   move_rule = 1;
if ( (strategy == 1) && (SLAVE_strategy == 1) )
   move_rule = Random(2);
// here the MASTER know everithing
// must send the move-rule also to the SLAVE
move_rule_result = NumToStr(move_rule);
// sending move-rule-result to the SLAVE
BTSendMessage(BT_CONN, MAILBOX, move_rule_result);


set_payoffs(strategy, SLAVE_strategy);

WriteString(handle, "<--->      Master strategy: ", length);
WriteString(handle, str_strategy, length);
WriteString(handle, "   Decision: ", length);
my_decision = NumToStr(strategy);
WriteString(handle, my_decision , length);
WriteString(handle, "   Payoff: ", length);
 my_decision = NumToStr(MASTER_payoff);
WriteString(handle, my_decision, length);


WriteString(handle, "   Slave strategy: ", length);
WriteString(handle, str_slave_strategy, length);
WriteString(handle, "   Decision: ", length);
my_decision = NumToStr(SLAVE_strategy);
WriteString(handle, my_decision, length);
WriteString(handle, "   Payoff: ", length);
my_decision = NumToStr(SLAVE_payoff);
WriteLnString(handle, my_decision, length);

CloseFile(handle);


ClearScreen();
TextOut (0,LCD_LINE1 , "My decision");
NumOut (0, LCD_LINE2, strategy);
TextOut (0,LCD_LINE3 , "His strategy");
NumOut (0, LCD_LINE4 ,SLAVE_strategy);
TextOut (0,LCD_LINE5 , "Move Rule");
```

```
      NumOut (0, LCD_LINE6, move_rule);
      TextOut (0,LCD_LINE7 , NumToStr(MASTER_payoff));
      TextOut (0,LCD_LINE8 , NumToStr(SLAVE_payoff));
      //screen_x += 13;

   }

/*-------------------------------------------------------------*/

int defect()
  {
   return 0;
  }

int cooperate()
   {
    return 1;
   }

int random_strategy()
   {
    return Random(2);
   }

int tit_for_tat()
   {
    return previous_move[0];
   }

int win_stay_lose_shift()
   {
    int val;
    if ( (previous_move[0] == previous_move[1]) && (previous_move[0] == 0) )
       {
        /* D/D case */
        if ( Random(101) <= PROB_COOPERATE )
           val = 1;
        else
           val = 0;
       }
    else
        {
         if ( (previous_move[0] == previous_move[1]) && (previous_move[0] ==
1) )
            {
             /* C/C case */
             val = 1;
            }
         else
             {
              /* all other cases */
              val = 0;
             }
        }
    return val;
   }
/*-------------------------------------------------------------*/

void set_payoffs( int my_s, int slave_s)
   {
    if ( my_s == 1 && slave_s == 1 )
          {
           MASTER_payoff +=4;
           SLAVE_payoff +=4;
          }
    if ( my_s == 1 && slave_s == 0 )
          {
           MASTER_payoff +=0;
           SLAVE_payoff +=5;
          }
    if ( my_s == 0 && slave_s == 1 )
          {
           MASTER_payoff +=5;
           SLAVE_payoff +=0;
          }
    if ( my_s == 0 && slave_s == 0 )
```

```
        {
         MASTER_payoff +=2;
         SLAVE_payoff +=2;
        }
   }


int GetColorValue(const byte port)
 {
 unsigned int r[], n[], s[];
 int cv;
 ReadSensorColorEx(port, cv, r, n, s);
 return cv;
 }

 /*-------------------------------------------------------------*/


 sub masterBTCheck(int conn){
   string cStr;
   cStr = NumToStr(conn);
   cStr = StrCat("on line",cStr);
   if (!BTCommCheck(conn)){
      TextOut(5,LCD_LINE2,"*Connect Slave");
      TextOut(5,LCD_LINE3,cStr);
      TextOut(60,LCD_LINE3,"from");
      TextOut(5,LCD_LINE4,"this NXT menu");
      Wait(3000);
      Stop(true);
   }
}
```

Slave Robot (Also known as Player B or Dynamic)

```
#include "NXCDefs.h"
#include "C:\Program Files (x86)\BricxCC\BTlib\BTlib.nxc"
#define DISTANCE_THRESHOLD 10
#define NUM_STRATEGY 100
#define PROB_COOPERATE 50
#define BT_CONN 1
#define MAILBOX 0


//here initialize all the "sliders"
#define DYNAMIC_COOPERATE 20
#define DYNAMIC_DEFECT 20
#define DYNAMIC_RANDOM 20
#define DYNAMIC_TIT_FOR_TAT 20
#define DYNAMIC_WIN_STAY_LOSE_SHIFT 20
//OUT_C right motor
//OUT_B left motor

int GetColorValue(const byte port);
task WatchAhead();
task Move();
task WatchBumper();
void play_a_round();
void go_back();
sub slaveBTCheck(int conn);
int defect();
int cooperate();
int random_strategy();
int tit_for_tat();
int win_stay_lose_shift();

int move_rule, long_way = 0, previous_move[2], motor_pwr = 75;
bool bt_received;
mutex motorMutex;

task main()
```

```
{
SetSensorColorFull(S3);
SetSensorLowspeed(S4);
SetSensorTouch(IN_1);
SetSensorTouch(IN_2);
Precedes(WatchAhead, WatchBumper, Move);
}
/*-------------------------------------------------------------*/

task WatchAhead()
     {
      while (true)
        {
         if ( SensorUS( S4 ) < DISTANCE_THRESHOLD )
           {
            motor_pwr = 40;
           }
        }
     }

/*-------------------------------------------------------------*/

task Move()
   {
    while (true)
       {
        if (long_way == 0)
           {
            switch(GetColorValue(S3))
              {
               case INPUT_YELLOWCOLOR:  Acquire(motorMutex);
                                        Off(OUT_C);
                                        OnFwd(OUT_B, motor_pwr);
                                        Release(motorMutex);
                                        break;

               case INPUT_WHITECOLOR:   Acquire(motorMutex);
                                        Off(OUT_C);
                                        OnFwd(OUT_B, motor_pwr);
                                        Release(motorMutex);
                                        break;


               case INPUT_BLACKCOLOR:   Acquire(motorMutex);
                                        Off(OUT_B);
                                        OnFwd(OUT_C, motor_pwr);
                                        Release(motorMutex);
                                        break;

              case INPUT_GREENCOLOR:   motor_pwr = 75;
                                       PlayTone(440, 1000);
                                       Acquire(motorMutex);
                                       OnFwd(OUT_BC, motor_pwr);
                                       Release(motorMutex);
                                       break;

              case INPUT_REDCOLOR:     Acquire(motorMutex);
                                       Off(OUT_B);
                                       OnFwd(OUT_C, motor_pwr);
                                       Release(motorMutex);
                                       break;
              }
           }
        else
            {
              switch(GetColorValue(S3))
              {
               case INPUT_BLACKCOLOR:     Acquire(motorMutex);
                                          Off(OUT_B);
                                          OnFwd(OUT_C, motor_pwr);
                                          Release(motorMutex);
                                          break;

               case INPUT_BLUECOLOR:      Acquire(motorMutex);
                                          Off(OUT_C);
                                          OnFwd(OUT_B, motor_pwr);
```

```
                                        Release(motorMutex);
                                        break;

              case INPUT_GREENCOLOR:    long_way = 0;
                                        break;
              }
             }
       }  /*end while */
 } //end task

/*--------------------------------------------------------------*/

// move rule
// 0- master goes back, slave forward
// 1- master goes forward, slave back
// 2- both go back

task WatchBumper()
     {
      while (true)
          {
           if (SENSOR_1 == 1 || SENSOR_2 == 1)
             {
              play_a_round();
              Acquire(motorMutex);
              switch(move_rule)
                 {
                  case 1:
                  case 2:       go_back();
                                break;

                  default:      Off(OUT_BC);
                                Wait(7000);
                                motor_pwr = 75;
                                break;
                 }
              Release(motorMutex);
             }
           }
      }

/*--------------------------------------------------------------*/

void go_back()
   {
    int finished = 0;
    OnRev(OUT_BC, 40);
    finished = 0;
    while (finished == 0)
        {
         Wait(1);
         if ( GetColorValue(S3) == INPUT_GREENCOLOR )
            {
             finished = 1;
             motor_pwr = 75;
             long_way = 1;
            }
         }
         OnFwd(OUT_BC, motor_pwr);
         Wait(200);
         OnFwd(OUT_C, motor_pwr);
         OnRev(OUT_B, motor_pwr);
         Wait(200);
   }


/*--------------------------------------------------------------*/


void play_a_round()
   {
    int result_dynamic = Random(NUM_STRATEGY);
    int  strategy, cont = 0;
    string result, slave_strategy;
    string in, in2;
    bool finito;
```

```
    if (result_dynamic < DYNAMIC_COOPERATE)
      result_dynamic = 1;
    else
      if (result_dynamic < (DYNAMIC_COOPERATE + DYNAMIC_DEFECT))
        result_dynamic = 0;
      else
        if (result_dynamic < (DYNAMIC_COOPERATE + DYNAMIC_DEFECT +
DYNAMIC_RANDOM))
          result_dynamic = 3;
        else
          if (result_dynamic < (DYNAMIC_COOPERATE + DYNAMIC_DEFECT +
DYNAMIC_RANDOM + DYNAMIC_TIT_FOR_TAT))
            result_dynamic = 2;
          else
             result_dynamic = 4;
   slaveBTCheck(0);  /* checking if connected through bluethooth */
    bt_received = FALSE;
    while ( bt_received == FALSE )
         {
          in2 = BTReceiveMessage(0, MAILBOX, TRUE);
          if (strlen(in2) > 0)
              bt_received = TRUE;
         }
   previous_move[1] = StrToNum(in2);
   slave_strategy = NumToStr(result_dynamic);
   BTSendMessage(0, MAILBOX, slave_strategy);

   if (result_dynamic == 0)
     strategy = defect();

   if (result_dynamic == 1)
     strategy = cooperate();

   if (result_dynamic == 2)
     strategy = tit_for_tat();

   if (result_dynamic == 3)
     strategy = random_strategy();

   if (result_dynamic == 4)
     strategy = win_stay_lose_shift();



    /*
    switch(random_val)
      {
       case 0:        strategy = defect();
                      break;
       case 1:        strategy = cooperate();
                      break;
       case 2:        strategy = random_strategy();
                      break;
       case 3:        strategy = tit_for_tat();
                      break;
       case 4:        strategy = win_stay_lose_shift();
                      break;
      }

      */

      previous_move[0] = strategy;

      /* here the slave has to send its strategy result to the MASTER who
will calculate the move-rule */

      result = NumToStr(strategy);   /* converting num into string */
      BTSendMessage(0, MAILBOX, result);
      /* I've to receive the message by repetetly polling the mailbox */
      bt_received = FALSE;
      while ( bt_received == FALSE )
         {
          in = BTReceiveMessage(0, MAILBOX, TRUE);
          if (strlen(in) > 0)
```

```
            bt_received = TRUE;
         }
      bt_received = FALSE;
      move_rule = StrToNum(in);
      ClearScreen();
      TextOut (0,LCD_LINE1 , "My decision");
      NumOut (0, LCD_LINE2, strategy);
      TextOut (0,LCD_LINE3 , "Move Rule");
      NumOut (0, LCD_LINE4, move_rule);
      /* here I got the rule I have to follow to move around */
   }

/*-------------------------------------------------------------*/

int defect()
  {
   return 0;
  }

int cooperate()
   {
    return 1;
   }

int random_strategy()
   {
    return Random(2);
   }

int tit_for_tat()
   {
    return previous_move[1];
   }

int win_stay_lose_shift()
   {
    int val;
    if ( (previous_move[0] == previous_move[1]) && (previous_move[0] == 0) )
       {
        /* D/D case */
        if ( Random(101) <= PROB_COOPERATE )
           val = 1;
        else
           val = 0;
       }
    else
        {
         if ( (previous_move[0] == previous_move[1]) && (previous_move[0] ==
1) )
            {
             /* C/C case */
             val = 1;
            }
         else
              {
               /* all other cases */
               val = 0;
              }
        }
    return val;
   }
/*-------------------------------------------------------------*/

int GetColorValue(const byte port)
 {
 unsigned int r[], n[], s[];
 int cv;
 ReadSensorColorEx(port, cv, r, n, s);
 return cv;
 }

 /*-------------------------------------------------------------*/

 sub slaveBTCheck(int conn){
   string cStr;
   cStr = NumToStr(conn);
```

```
   cStr = StrCat("on line",cStr);
   if (!BTCommCheck(conn)){
      TextOut(5,LCD_LINE2,"*Connect NXT");
      TextOut(5,LCD_LINE3,cStr);
      TextOut(60,LCD_LINE3,"from");
      TextOut(5,LCD_LINE4,"master NXT menu");
      Wait(3000);
      Stop(true);
   }
}
```