The Evolutionary Hawk-Dove Model on Graphs

Tara Pesman
3971937

Animal conflict

July 2, 2015

Abstract

In evolutionary game theory, the Prisoner's Dilemma is a well-studied model for a diverse range of real-life situations. The Hawk-Dove game is closely related to the Prisoner's Dilemma, but has received much less attention. In this thesis the Hawk-Dove game on graphs is explored, as based on a study by Hauert & Doebeli (2004). The experimental part of this thesis consists of a program written in `c++` for modeling the Hawk-Dove game on graphs, reproducing Hauert & Doebeli's results, as well as extending the scope of research to different graphs, update rules, and payoffs. This part is preceded by the necessary theoretical background in game theory, evolutionary game theory, graph theory and the Prisoner's Dilemma and Hawk-Dove game.

Supervised by dr. M. Ruijgrok, Mathematical Institute, Utrecht University

Source cover photo: `http://pel.hu/photo/gp.php?pg=wildlife`

Contents

1 Introduction

The aim of this thesis is two-fold. First, to give an exploration of the theoretical background necessary to be able to formally understand evolutionary game-theoretical models. Second, to actually provide a working model of an evolutionary game, in this case the Hawk-Dove game on graphs. The research question for this experimental part is: what factors influence the proportion of Doves in the population?

There are two parts to this thesis, corresponding to the two aims: The first part consists of a theoretical treatment of game theory, evolutionary game theory, and graph theory, as well as an in-depth discussion of the Hawk-Dove game. The second part consists of a step-by-step discussion of the program written to model the Hawk-Dove game on graphs, as well as a discussion of the obtained results. The code for the program is provided in the appendix.

Part I

History and Theory

In the first part of this thesis, the history[1] and theory of game theory, evolutionary game theory, graph theory and the Hawk-Dove game are discussed. Though this part is self-contained and gives a good overview of the discussed topics, the scope is to some degree specialized so as to provide the necessary theoretical basis for the second part: experimenting with the Hawk-Dove game on graphs.

2 Game Theory

This section is based on Leyton-Brown & Shoham (2008).

Game theory studies the interaction of self-interested agents. Self-interested does not necessarily mean that agents want others to be worse of. It simply means that each agent has her own idea of what states of the world she would like to be in. In her interaction with other agents she will try to bring about these states of the world.

The way these preferences of an agent are formalized is by using *utility theory*, which quantifies an agent's degree of preference across her available alternatives, as well as describes how the agent will change her preferences in the face of uncertainty regarding which alternative she will actually receive. Utility theory thus provides a way in which states of the world can be mapped to the real numbers: this mapping is called a *utility function*. The unit name of these numbers is *util* and they are interpretated as representing the level of happiness of an agent. The added value of defining an agent's preferences using a utility function can be seen clearly in the case of uncertain circumstances. If an agent does not know in which state of the world she is in, the utility function still ascribes a utility value to the different actions she can perform, namely, the expected value of the utility function with respect to the relevant probability distribution over the alternative states of the world.

Hence, given sufficient knowledge of the different possible outcomes of actions, as well as the probabilities of the different states of the world, describing the optimal actions for one

[1] Any history discussed is presented in the form of milestones, emphasizing key authors and works in the field; this is by no means exhaustive.

player is straightforward. However, when describing a world containing *two or more* agents, analysis is made more complicated by the influence the agents have on each other's utilities. These kind of interactions can be modelled using *non-cooperative* game theory[2]. Just like the term 'self-interested', 'non-cooperative' is defined somewhat differently here than in its colloquial use: agents do not need to have conflicting preferences, they only need to make their decisions independently from each other.

2.1 Milestones in the History of Game Theory

This section is based on Hykšová (2004).

The history of the field of game theory goes back about three hunderd sixty years[3]. In the year 1654, the correspondence between the two brilliant French mathematicians Pierre de Fermat (1607-1665) and Blaise Pascal (1623-1662) led to the advent of probability calculus. The thereby provided concept of *probability* is one of the necessary pillars of the field of game theory, as can be seen from the introduction above.

In the over two hunderd fifty years to follow, the field of game theory was quite distached and constituted many isolated examples. Only in the beginning of the twentieth century, in the period 1921-1928, was the field of game theory properly defined mathematically. The French mathematician Émile Borel (1871-1956) defined the notion of *method of play*, which is our modern day *pure strategy* (see section 2.2.2), as well as looked for solutions to *mixed strategy games* (see section 2.2.2), using what is now called the minimax priniciple (see section 2.2.3)[4]. Borel's definitions were on the right track, however, they did not yet provide a full-blown mathematical framework for the theory of games. This was provided by the Hungarian polymath John von Neumann (1903-1957), who published the paper "Zur Theorie der Gesellschaftsspiele" (1928) a few years after Borel. In contrast, his paper did provide a comprehensive and exact formalization of the important notions in game theory, as well as a formal proof of the minimax principle. In part because of this paper and the influence it has had on further developments in the field of game theory, Von Neumann is often attributed the title of 'founder of game theory'.

Sixteen years later, Von Neumann collaborated with German economist Oskar Morgenstern (1902-1976) to write a book that would prove to be another milestone in the

[2]The other branch of game theory is *coalitional* game theory and uses the group of agents as its unit of analysis, rather than the individual agents making up that group as is done in non-cooperative game theory.

[3]Before this time game-theoretic insights have been documented since ancient times. For an overview of historical examples, see Ross (2014).

[4]He did this in a series of notes, see Hykšová (2004) sources [4]-[6].

history of game theory. In *The Theory of Games and Economic Behavior* (1944) they showed the broad applications of game theory in economics as well as other disciplines. Furthermore, they laid out a full-fledged axiomatic utility theory, which up till then had not been clearly defined. The field of game theory became, with the publishing of this book, a proper mathematical discipline.

The next milestone came only five years after Von Neumann and Morgenstern's paper in the form of the work of American mathematician John Forbes Nash Jr. (1928-2015), who died only recently in a car accident (23rd of May). Nash contributed to the field of game theory in a number of ways, his most well-known contributions being published in his Ph.D. thesis "Non-Cooperative Games" (1949)[5]: he defined *normal-form games* (see section 2.2.2), developed the notion of the *Nash equilibrium* (see section 2.2.3.3) and proved that each game[6] contains at least one Nash equilibrium.

Following the papers by Von Neumann and Morgenstern and Nash a lot of subsequent research was conducted, really sparking the extension of the scope of research into areas beyond game theory's original field of application, economics. One of these disciplines is Political Science, of which game theory is an inseparable part. The first paper published in this discipline appeared in 1954 and used game theoretical notions to determine the power of the members of the United Nations Security Council (Shapley & Shubik). Today game theory is used in modeling "various situations related to elections, legislature, politics of interest groups, lobbies, bargaining, etc." (Hykšová (2004)).

The application of game theory is not limited to human affairs, but is used extensively to study conflict and cooperation in the biological world as well. Even though some papers concerning this area of interest were published in the 1960s, it were two later papers which proved to be the milestones in applying game theory to this field. In 1973 the British evolutionary biologist John Maynard Smith (1920-2004) and American geneticist George R. Price (1922-1975) published their paper "The Logic of Animal Conflict". The work inspired a host of new research and applications, which Maynard Smith summarized nearly a decade later in his book *Evolution and the Theory of Games* (1982). More on this interesting subfield of game theory will be discussed later (see section 3).

Close to the publication of *Evolution and the Theory of Games*, a book was published by the American political scientist Robert Axelrod (1943-) as well. His book *The Evolution of*

[5]He wrote his thesis in 1949, after which the most important results were published in a brief paper (1950) and a more detailed paper (1951).

[6]That is, each game with a finite number of players and action profiles (see sections 2.2.2 and 2.2.3.3).

Cooperation (1984) does what its title announces and discusses how game theory may be used to model the emergence of cooperation. On the one hand this was a powerful finding in support of the theory of biological evolution, which up till then had not been able to explain cooperation convincingly. The reason cooperation seemed inexplicable for so long is its apparent clash with natural selection, which seems to favor those who only act with their own self-interest at heart. Axelrod used a game theoretical computer model, for which he asked collegues to submit strategies in a tournament, to show that cooperation *can* in fact emerge, even with players who are only interested in their own good. On the other hand, Axelrod's book, by shedding light on an important and before inexplicable phenomenon, showed the might and relevance of game theoritical analysis.

Finally, in 1994, John Nash was awarded the Nobel Prize in economics for his contributions to non-cooperative game theory, together with Hungarian-American economist John Harsanyi and German economist Reinhard Selten, who extended his work. Nowadays game theory is regularly used in a number of different disciplines, ranging from economics to biology (see section 3.5).

2.2 Games in Normal-Form

This section is based on Leyton-Brown & Shoham (2008).

In game theory there are two different ways of representing a game: in *extensive form* and in *normal-form*. The former presents games in the form of *game trees* and has particular merrits, such as a good representation of incomplete information, however, each extensive form game can be made into a normal-form game. The latter is therefore seen as the standard form and will be the focus of this thesis.

2.2.1 Prisoner's Dilemma

Game theory is concerned with modeling interactions, called *games*, between agents, often called *players*. In order to get some intuition regarding the type of interactions game theory is used for, let us start with giving a mostly qualitative account of the well-known Prisoner's Dilemma (PD). The situation is as follows. Imagine you and your accomplish have robbed a bank together, however, the police have caught onto you and taken both of you into custody. They are questioning you and your partner separately, hoping to squeeze out a confession. You are presented with the following alternative scenarios: you confess and your partner

keeps quiet, in which case you walk free (*temptation payoff*); you confess but you partner does as well, in which case you get the punitive sentence of ten years in prison (*punishment payoff*); you do not confess but your partner does, in which case you get the increased sentence of twenty-five years in prison (*sucker payoff*); you do not confess and neither does your partner, in which case you get the reduced sentence of five years in prison (*reward payoff*). The game is symmetrical, i.e. your partner is offered the exact same combinations of actions and consequences. The payoffs for each of the four scenarios can be put in order of preference:

$$T > R > P > S$$

where T is the temptation payoff of 0 years in prison, R the reward of 5 years, P the punishment of 10 yeas, and S the sucker of 25 years. The value that each of these prison sentences represents to you can be expressed in utils. Let us say that a prison sentence of twenty-five years is worth minus three utils to you, as it takes away from your happiness; ten years is worth minus two utils; five years is worth minus one util; and no sentence is worth zero utils. Then the game may be represented by the following two-dimensional vector:

	keep quiet	**confess**
keep quiet	-1	-3
confess	0	-2

Table 1: *PD: payoff matrix for the row player*

The rows in Table 1 represent the alternative actions, called *strategies*, available to you; likewise the columns represent the alternative actions available to your partner. The entry $\big[$*keep quiet, confess*$\big]$ in the vector represents the utility, or *payoff*, you receive if you use the strategy *keep quiet* and your partner uses the strategy *confess*. In symmetrical games, one of these matrices is enough to provide the information necessary for all players, however, there are many asymmetrical games as well. In order to accommodate this better, Table 1 may be expanded to include the payoffs for the opposing player as well:

	C	**D**
C	$-1, -1$	$-3, 0$
D	$0, -3$	$-2, -2$

Table 2: *PD: payoff matrix for both the row and the column player*

In Table 2 each entry contains two utilities, the first representing the payoff of the row player (you in this case), the second representing the payoff of the column player (your partner in crime). Here C and D are short for *cooperate* and *defect* respectively, which are the

conventional names for the two strategies in the PD and refer to the interaction between the players: to cooperate is to not rat each other out and keep quiet, whereas to defect is to sell out the other and confess.

Using payoff matrices, game theory can be used to determine what a player's optimal strategy is, thereby predicting which strategy she will actually adopt, given she is perfectly rational[7]. There are different ways in which game theory may determine which strategy may be called 'the best', which will be discussed next (see section 2.2.3). However, first a formal definition of a normal-form game needs to be given.

2.2.2 Definition of Games and Strategies

Now that we have a better grasp of what a 'game' in game theory constitutes, the time has come to give a formal definition of a normal-form game. As mentioned, John Nash, in his 1949 work, was the first to formally define normal-form games. In order to formalize a normal-form game it is necessary to define the *players*, *strategies*, and *utilities* of the game:

Normal-form game

A finite, n-person normal-form *game is a tuple* (N, A, u)*, where*

- N *is a finite set of* n *players indexed by* i*;*
- $A = A_1 \times ... \times A_n$*, where* A_i *is the finite set of actions available to player* i*. Each vector* $a = (a_1, ..., a_n) \in A$ *is called an* action profile*;*
- $u = (u_i, ..., u_n)$*, where* $u_i : A \mapsto \mathbb{R}$ *is a real-valued* utility function *for player* i*.*

Let us continue by further formalizing the players' strategies. A pure strategy is one in which the player selects one action and always chooses that action as her strategy. In contrast, a mixed strategy is one in which a player has several possible actions and chooses which one to play each time:

Mixed strategy *Let* $A_i = \{a_1, ..., a_m\}$ *be the set of actions available to player* i*. Then a* mixed strategy *for player* i *is a vector* $(p_1, ..., p_m)$ *such that* $p_i \geqslant 0$ *and* $p_1 + ... + p_m = 1$*. Player* i *will use action* a_j *with probability* p_j*.*

The expected payoff for playing a mixed strategy simply follows from the linearity of the utility function. Furthermore, pure strategies are of the form $(0, 0, ..., 0, 1, 0, ..., 0)$, so that player i will use action a_j with probability 1.

[7] This assumption may not be as straightforward as it seems (see section 3.1).

2.2.3 Analyzing Games

Having defined games in normal-form and the strategies players in these games have available to them, it is time to see how game theory reasons about these games. For games involving one player, game theory uses the concept of an *optimal* strategy to predict which course of action is deemed 'best'. The optimal strategy is that strategy that maximizes the player's payoffs in the given environment, which is a source of uncertainty, due both to probabilistic events and partial information. In games involving mulitple players, however, analysis is not quite so unambiguous, since each player's optimal strategy depends on the strategies the other players are using, thus making the solution to the game a dynamic concept, rather than a stable one. To deal with multi-agent games, game theorists has developed several ways of reasoning about the best course of action, called *solution concepts*, each interesting in its own way. Here four of the most fundamental ones will be discussed: minimax and maximin, pareto optimal, dominance, and Nash equilibrium.

2.2.3.1 Zero-sum Games, Minimax and Maximin

Historically, the first type of games to be analyzed were so-called *zero-sum* games. The concept of a zero-sum game applies to two-player games in which the payoff of one player is equal to that of the second player, but differing in sign. A more appropriate name is *constant-sum* games, since the two payoffs do not need to cancel to zero, but *do* always add to some constant value[8]:

Zero-sum game *A two-player normal-form game is* zero-sum *if there exists some c such that for all $a \in A_1 \times A_2$ it is the case that $u_1(a) + u_2(a) = c$.*

In other words, for all action profiles (a: set of chosen actions by each player, in this case two) within the range of pairs of possible actions of the players ($A_1 \times A_2$), there exists some constant c such that the players' utilities add up to this constant.

To determine what action is best for a player in a constant-sum game, two related solution concepts were defined. The concept of *minimax* tells a player to minimize her opponent's maximal payoffs, whereas the concept of *maximin* tells a player to maximize her own minimal payoffs. In zero-sum games the minimax and maximin solutions are identical, which is a theorem by John von Neumann. However, for non-zero-sum games, they are generally not equivalent.

[8]This is the case because utility functions are linear, allowing the application of linear transformations, such as mulitiplication or addition, to the payoff vector without changing the game.

Taking the example of the PD (see Table 2), a non-zero-sum game, the minimax strategy would be to defect, since your opponent would maximally receive -2, rather than 0 in case you would have cooperated. In the case of the PD the maximin strategy is the same as the minimax strategy, namely to defect, since your minimal payoff would be -2, rather than -3 in case you would have cooperated.

2.2.3.2 Dominance

The strongest and most straightforward solution concept is that of dominance. Let us start by defining a *strategy profile*:

Strategy profile *The set of strategies used by the players, called the* strategy profile*, is given by* $s = (s_1, ..., s_n)$.

Additionally, let us refer to all players except for player i, by $-i$ and define their strategy profile to be $s_{-i} = (s_1, ..., s_{i-1}, s_{i+1}, ..., s_n)$. Then the strategy profile for all players can be written as $s = (s_i, s_{-i})$. Now, a strategy is *dominant* when it yields the highest payoff regardless of what strategies the other players' are using, or more formally:

Dominant strategy *A strategy* $s_i \in S_i$ *of player* i *is* dominant *if* $u_i(s_i, s_{-i}) \geqslant u_i(s'_i, s_{-i})$ *for all alternative strategies* $s'_i \in S_i$ *for player* i*, and for all alternative strategies* $s_{-i} \in S_{-i}$.

An even stronger version is provided by *strictly dominant* strategies, for which the inequality above holds strictly for all $s_i \neq s'_i$ and for all $s_{-i} \in S_{-i}$. The major advantage of the dominance solution concept is that a player using it does not need to make any predictions about the actions of her opponents. The major disadvantage is that more often than not there is no (strictly) dominant strategy.

In the PD it is easy to see that to defect is the (strictly) dominant strategy (see Table 2): if your opponent cooperates, you either get a payoff of -1 (in case you cooperate) or of 0 (in case you defect); if your opponent defects, you either get a payoff of -3 (in case you cooperate) or of -2 (in case you defect). Since $-1 < 0$ and $-3 < -2$ it is always the better to defect, regardless of the strategy your opponent uses. Because the PD is a symmetrical game, as noted before, there is a *pair* of strategies that is dominant, hence, it is expected every game always ends with the players having chosen (D, D). Interestingly, this is not so;

in certain situations the outcome of the game deviates from this prediction[9].

2.2.3.3 Nash Equilibrium

The most influential solution concept in game theory is the *Nash equilibrium*. As mentioned, this concept was developed by John Nash in his Ph.D. "Non-Cooperative Games" (1949). He proved that each game with a finite number of players and action profiles, has at least one Nash equilibrium. In order to formally define a Nash equilibrium, it is necessary to first introduce the concept of a *best response*.

Start by observing that if a player i were to know what strategy her opponents are going to play, picking which strategy is best for her is a simple case of single-agent utility-maximizing. Let us refer to the other players, i.e. all except for player i, by $-i$ and write the strategy profile for all players as $s = (s_i, s_{-i})$. If the other players were to use s_{-i}, player i would be in a simple single-agent utility-maximizing situation. Now, a player's *best response* is that strategy that will maximize the agent's payoff in the light of the knowledge or guess of the strategies the other players will use.

Best response *Player i's* best response *to the set of strategies played by the other players is a mixed strategy* s_i^* *such that* $u_i(s_i^*, s_{-i}) \geqslant u_i(s_i, s_{-i})$ *for all* $s_i \in S_i$.

The best response can be unique, in the exceptional case that the best response is a pure strategy. Generally, however, the best response is a mixed strategy, making that there are always an infinite number of best responses definable[10].

A Nash equilibrium can now be defined using the concept of best response. Informally, what it means for a set of strategies (one strategy for each player), to constitute a Nash equilibrium is that no player has an incentive to change her strategy: for any given player there is no strategy available that, if she were to switch unilaterally, she would maintain the same or receive a better payoff. This can be formalized as follows:

Nash equilibrium *The set containing the strategies of all players,* $s = (s_1, ..., s_n)$*, called the* strategy profile*, is a* Nash equilibrium *if* s_i *is the best response to* s_{-i} *for all players* i.

[9] For an overview of the five mechanisms that promote cooperation in the PD, see **?**

[10] This is because, if a player were to have two (or more) actions available to her with non-zero probability, then she must be indifferent between them. If not, she could reduce the probability at least one of them to zero. *Any* mixture, not just the one in s_i^*, of those two (or more) actions therefore is a best response.

The Nash equilibrium can be envisioned as a 'stable' state: no player would want to change her strategy if she knew what strategies the other players were using. Two versions can be distinguished, depending on the inequality sign used in the best response definition: *weak Nash*, using $\geqslant$ as in the definition above, and *strict Nash*, using $>$ instead. Mixed strategy Nash equilibria are necessarily weak, whereas pure strategy Nash equilibria may be weak or strict. Weak Nash equilibria are less stable than strict Nash equilibria, since in the former there is at least one player who has a best response to the other players' strategies that is not her equilibrium strategy.

The following is based on Peters (2008).

Finding all Nash equilibria of a game, including both mixed and pure strategies, can be a demanding task. Here it will be shown how to find the pure strategy Nash equilibria only, to give an idea of the process. As an example, consider again the PD in Table 2. For the column player, mark with an asterix his best responses to the strategies the row player can choose, yielding:

	C	**D**
C	$-1, -1$	$-3, 0^*$
D	$0, -3$	$-2, -2^*$

Table 3: *PD: highlighted best responses for the column player*

Next, mark with an asterix the best responses of the row player to the strategies the column player can choose, yielding:

	C	**D**
C	$-1, -1$	$-3, 0$
D	$0^*, -3$	$-2^*, -2$

Table 4: *PD: highlighted best responses for the row player*

Putting these together yields Table 5.

	C	**D**
C	$-1, -1$	$-3, 0^*$
D	$0^*, -3$	$-2^*, -2^*$

Table 5: *PD: highlighted best responses for both the row and column player*

The pure Nash equilibria of the game are given by those pairs of strategies that are mutual best responses, in this case $\big[D, D\big] = -2, -2$. Hence, once the players settle on these

strategies, neither of them will want to switch change strategies unilaterally.

3 Evolutionary Game Theory

The application of game theory is not limited to human affairs, but is used extensively to study conflict and cooperation is the biological world as well. In the zoological domain game theory is used for "the analysis, modeling and understanding the fight, cooperation and communication of animals, coexistence of alternative traits, mating systems, conflict between the sexes, offspring sex ratio, distribution of individuals in their habitats, etc." The list is equally extensive in the botanical domain, where game theory is used to address "questions of seed dispersal, seed germination, root competition, nectar production, flower size, sex allocation, etc." (Hykšová (2004))

3.1 Historical Context of Evolutionary Game Theory

This section is based on Sugden (2001).

Classical game theory is, as discussed above, concerned with *games*, which are defined in terms of *players*, *strategies*, and *utilities*. The main goal is to find a *solution* to a game, which is a combination of strategies, one for each player. The "Holy Grail" therefore is to find a *solution concept* that, in every game, picks out precisely one set of strategies for the players that is the solution. Until the 1980s classical game theory stayed the standard, however, some difficulties started to appear, the main three of which will be discussed here (Alexander (2009)).

Equilibrium Selection

To start there is the *equilibrium selection* problem, which consists of two complications regarding Nash equilibria. Every non-cooperative game in which players are allowed to use mixed strategies has at least one Nash equilibrium (by John Nash's famous theorem, as discussed in section 2.1), however, this is not the case for games in which only pure strategies are allowed. For some games, therefore, the solution(s) game theory seeks cannot be found. A more critical complication is presented by the multiplicity of Nash equilibria in some games. Since all Nash equilibria are equally rational to choose, how does a rational agent select one of the available alternatives? In response to this problem, game theorists

have suggested refinements of the concept of a Nash equilibrium. These have grown so numerous that nearly every choice of a Nash equilibrium can be justified by one or more of the proposed refinements, making that the object of the question has shifted from which equilibrium to pick to which refinement to use.

Hyperrationality

A second difficulty encountered in classical game theory regards the assumption of perfect rationality, which has its origin in the utility theory that forms the basis of game theory. In order to assign cardinal utility functions to agents, it is necessary that those agents have well-defined and consistent preferences for the possible outcomes of the game. Because there are infinite possible outcomes of the game, each player needs to have a well-defined, consistent set of infinitely many preferences regarding these outcomes. This is provided by the assumption of perfect rationality. The problem here is that an abundance of studies in experimental economics, as well as other fields, have shown that the behavior of real human being does not agree with the assumption of hyperrationality. Preferring A to B, B to C, and C to A, is an example of how people can have preferences that are not well-defined and consistent, as is assumed in classical game theory.

Lack of Dynamical Theory

The third difficulty discussed here concerns the dynamical nature of games. The most important aspect of a game is the interaction that takes place between the players, resulting in some outcome. An agent observes her opponent's behavior, learns from this, and responds taking into account what she has learned. Classical game theory, however, does not address this process of rational deliberation, but instead presumes, as discussed earlier, static preferences.

Development of Evolutionary Game Theory

This section is based on Alexander (2009).

The first to develop evolutionary game theory was the English evolutionary biologist R.A. Fischer (1890-1962). In his book *The Genetical Theory of Natural Selection* (1930) he investigated the approximate equality in sex ratio in mammals. How can it be that the sex ratio is approximately equal in species where the majority of males never mate? Those males

seem to be nothing more than a burden to the rest of the population. The insight Fischer had was that individual fitness, when measured as the expected number of grandchildren, depends on the distribution of males and females in the population. If the population has more females, then males have a higher individual fitness, and vice versa. Hence, evolutionary dynamics leads to equal numbers of males and females.

Even though Fischer's theory can be put in game theoretical terms, since individual fitness depends on the sex ratio in the population, thereby adding a strategic element in evolutions, he did not phrase it as such. The first to explicitly apply game theory to evolutionary biology was the American evolutionary biologist R.C. Lewontin (1929-), who published a paper called "Evolution and the Theory of Games" (not to be confused with Maynard Smith's book by the same name) in 1961, which already discussed species 'playing against Nature'. The first work of impact was "Game Theory and the Evolution of Fighting" (1972)[11], which was published a good ten years later by Maynard Smith and introduced the important notion of an *evolutionarily stable strategy* (see section 3.4). The work that actually sparked a revolution in game theory was published one year later, also by Maynard Smith, this time accompanied by Price, called "The Logic of Animal Conflict" (1973). Two other works that fuelled the expansion of evolutionary game theory into a full-fledged discipline were *Evolution and the Theory of Games* (1982) by Maynard Smith, giving an overview of the developments of the ten years previous, and *The Evolution of Cooperation* (1984) by Axelrod, which showed the application of evolutionary game theory to the social sciences (as discussed in section 2.1).

Evolutionary game theory provides progress in solving the three problems plaguing classical game theory (see section 3.1). The first difficulty presented was equilibrium selection: here, some (see Samuelson (1998)) hope that future developments in evolutionary game theory can assist with this issue. There is evidence that the second difficulty of assumed hyperrationality is (close to) not present in evolutionary game theory. Evolutionary game theory is able to successfully predict and explain the behaviors seen in plants and animals, which do not satisfy the assumption of perfect rationality. This suggests that evolutionary game theory is less dependent on rationality and may therefore be better suited to predict and explain human behavior than classical game theory. Finally, regarding the third difficulty of the lack of dynamical theory, since no assumptions of perfect rationality are made, evolutionary game theory does not presume static preferences. Moreover, evolutionary game theory naturally integrates the dynamical aspect to game playing.

[11] See section 1 of Alexander (2009).

3.2 Some Definitions

This section is based on Hauert (2008).

Before looking at evolutionary game theory in more detail, it is necessary to give descriptions of some concepts that are central to the theory of evolution. The three elements that constitute the process of evolution are *selection*, *variation*, and *random drift*.

Selection

Individuals in a population that have an above-average fitness have a greater chance of reproducing and thus of passing on their genetic or cultural traits to future generations, thereby increasing the frequency of these traits in the population. Likewise, individuals with traits that produce a below-average fitness have a smaller chance of reproducing, thus over time reducing the abundance of those traits in the population. This process is called *selection*.

Variation

Due to differences between individuals there is differentiated fitness within a population, and thus differentiated reproductive success. This *variation* between individuals is caused by a number of processes, including mutations, genetic recombination, spontaneous alterations and flawed imitations of behavioral patterns. Selection acts on these differences and amplifies them in time.

Random drift

The process of passing on traits through reproduction is stochastic. This means that even the fittest member of a population may, albeit with a small chance, not get to reproduce, and, likewise, the least fit member of the population may, with a small chance, do get to reproduce. This process of *random drift* counteracts the process of evolution. Its power increases along with decreasing population size or decreasing differences in fitness between individuals in a population.

Evolution, then, is the process by which different traits (variations) within a population are probabilistically (because of random drift) selected. It is important to note from the description of *selection* that, whereas Darwinian selection acts on *genes*, *cells* and *individuals*, this is not so in evolutionary dynamics, which instead focuses on *populations* as the subject of selection and evolution.

3.3 Differences from Classical Game Theory

Within the field of game theory, two approaches may be distinguished: classical game theory (CGT), which in this thesis is often referred to as just *game theory*, and evolutionary game theory (EGT). Maynard Smith discussed the two main differences between these two in his 1986 paper entitled "Evolutionary Game Theory".

The first difference is the replacement of *utility* by *fitness*. The concept of utility takes qualitatively distinct outcomes, quantifies them and orders them on a linear one-dimensional scale; a rather artificial process. In contrast, fitness is naturally quantified by number of offspring and the ordering is thus unambiguous. The second difference is the replacement of *rationality* by *natural selection*. In CGT, a 'solution' requires all players to act rational, which presents two problems: first, there is no unambiguous definition of 'being rational', and second, people do not behave rationally. In contrast, a 'solution' in the evolutionary game-theoretical sense refers to a stable fixed point of dynamics.

Kooistra has identifies some further necessary assumptions in EGT that differ from CGT:

- Players do not rationally choose which strategy to use, but instead they only use one strategy over and over, which may be interpreted as the expression of a gene or instinct.
- Strategies are passed on, unchanged, from parent to offspring asexually in a process called *replication* (more on this in section 3.4.3).
- Replication is not always perfect, resulting in mutants who play different strategies than their parents.
- Birth and death rate are constant and generally equal.

3.4 Two Approaches

This section is based on Alexander (2009) and Peters (2008).

Two approaches can be identified in the field of evolutionary game theory, based on the two central concepts in the theory of evolution: *mutation* and *selection* respectively. The first approach takes the concept of *evolutionary stable strategies* as the main tool of analysis. Since this approach does not actually consider the underlying process by which behaviors change in a population, it can be seen as providing a static explanation of what evolutionary stability is. The second approach constructs an explicit model of the change in frequency of

strategies present in the population and studies properties of evolutionary dynamics within that model. In contrast with the first approach, this approach, once it has created a model, does not need any of the stability concepts standardly used in analyzing games, such as discussed in section 2.2.3.

3.4.1 Hawk-Dove Game

This section is based on Peters (2008).

The example that will be used to illustrate the theory in this section is the famous evolutionary game of *Hawk-Dove* (HD). The relation of this game to the PD, as well as its relevance to this thesis will be discussed in section 5. The HD game models the following situation. Individuals in a large population meet each other at random and interact in pairs. They can either behave aggressively (Hawk) or passively (Dove) in the fight over, for example, nest sites or food. Since this behavior is genetically determined, the individuals do not really have a choice regarding which strategy to use. The payoffs in the game reflect (Darwinian) fitness, i.e. the number of offspring, and are given in Table 6.

	D	**H**
D	$2, 2$	$1, 3$
H	$3, 1$	$0, 0$

Table 6: *Hawk-Dove payoff matrix*

3.4.2 Evolutionary Stable Strategies

This section is based on Kooistra (2012) and Leyton-Brown & Shoham (2008).

In classical game theory the most important solution concept is the Nash equilibrium; in evolutionary game theory this is the evolutionary stable strategy (ESS). An ESS is a strategy that will be maintained in a homogeneous population, meaning one in which all players play one strategy s, and cannot be invaded by mutants playing a different strategy t. The concept of an ESS is formalized as follows:

Evolutionary stable stategy *Given a symmetric, two-player game* $G = (\big[1, 2\big], A, u)$*, a mixed strategy* s *is an ESS if for all* $s' \neq s$*,*

$$u\Big(s, (1-\epsilon)s + \epsilon s'\Big) > u\Big(s', (1-\epsilon)s + \epsilon s'\Big)$$

Explained in other words, this means the following. Imagine a population of natives who all play strategy s. Say a part of this population, some small proportion $0 < \epsilon < 1$, starts playing a mutant strategy s'. Then strategy s cannot be invaded, i.e. is evolutionary stable, if, for any s', it does better against the population at large (consisting of natives and a few mutants: $(1-\epsilon)s + \epsilon s'$) than a mutant strategy s' does. Since this need only be the case for small ϵ, this is equivalent to requiring:

Evolutionary stable stategy *A strategy is an ESS if for all $s' \neq s$,*

- Either $u(s,s) > u(s',s)$
- Or else $u(s,s) = u(s',s)$ and $u(s,s') \geqslant u(s',s')$

The concept of an ESS is closely related to that of the Nash equilibrium, as can be seen from the following two theorems:

Theorem 1 *Given a symmetric, two-player game $G = (\{1,2\}, A, u)$ and a mixed strategy s, if (s,s) is a strict (symmetric) Nash equilibrium of G, then s is a ESS.*

This too is easy to show, since it follows from the definition of a Nash equilibrium that s satisfies $u(s,s) > u(s',s)$, which satisfies the first condition for an ESS.

Theorem 2 *Given a symmetric, two-player game $G = (\{1,2\}, A, u)$ and a mixed strategy s, if s is an ESS, then (s,s) is a Nash equilibrium of G.*

From the definition of an ESS it follows that $u(s,s) \geqslant u(s',s)$, i.e. it is a best response to itself and thus a Nash equilibrium. Not every Nash equilibrium is an ESS however; this is only necessarily the case for strict Nash equilibria.

Hence, a strategy is *evolutionary stable* if it satisfies the two above-mentioned conditions. First, it needs to be a Nash equilibrium, i.e. the utility of playing strategy s against s is better or equal to the utility of playing s' against strategy s $\Big(u(s,s) \geqslant u(s',s)\Big)$. This ensures that no one player would have an incentive to switch strategies unilaterally when given the chance. Second, if the utility of playing s against s is equal to the utility of playing s' against strategy s $\Big(u(s,s) = u(s',s)\Big)$, then it must be the case that the utility of playing s against s' is better than the utility of playing s' against s' $\Big(u(s,s') \geqslant u(s',s')\Big)$. This condition ensures that if a native (playing s) encounters a mutant (playing s'), the native will prove superior, thereby preventing the mutant from infiltrating the population.

Finally, note that the definition for evolutionary stability has, similarly to the dominance concept, two versions: *weak*, which is described above, using $\geqslant$, and *strong*, replacing this by $>$. In the weaker version the players using the mutant strategy will not increase in abundance, however, their number will also not decrease. In contrast, in the stronger version the mutant *will* decrease and eventually die out.

3.4.3 Replicator Dynamics

This section is based on Peters (2008).

The concept of an ESS is based on *mutation*, however, if the concept of *selection* is to be incorporated, a more dynamical focus is necessary. The second approach to evolutionary game theory does just that and describes the process of change within a population due to selection using so-called *replicator dynamics*. A detailed treatment of the mathematics behind this is not the focus of this work, however, the general idea will be illustrated using the HD game.

Consider an infinitely large population in which two kinds of strategies are used, Hawk and Dove. Define a vector of population shares:

$$\mathbf{x} = (x, 1 - x)$$

where $x \in \big[0, 1\big]$ is the proportion of, say, Hawk, and $(1 - x)$ is the proportion of Dove in the population. Using Table 6, it can be seen that an individual playing H against the rest of the population $\mathbf{x}$ yields an expected payoff of:

$$0 \cdot x + 3 \cdot (1 - x) = 3(1 - x)$$

since she will receive a payoff of 0 for any fellow Hawk she meets (proportion x) and a payoff of 3 for any Dove she meets (proportion $(1 - x)$). Similarly, an individual playing Dove will receive an expected payoff of:

$$1 \cdot x + 2 \cdot (1 - x) = 2 - x$$

Hence, it can be concluded that the average fitness of the population is:

$$x \cdot 3(1 - x) + (1 - x) \cdot (2 - x) = 2 - 2x^2$$

Now, assume the population shares change over time, thus that $\mathbf{x}(t)$, and that this change

is proportional to the difference in average fitness between the two strategies. Then this change is given by

$$\dot{x}(t) = \frac{dx(t)}{dt} = x(t)\Big[3(1 - x(t)) - (2 - 2x(t)^2)\Big] \tag{1}$$

which is the *replicator dynamics* for the HD game. Equation 1 says that the proportion of Hawks in the population changes continuously ($dx(t)/dt$) and that this change is proportional to the difference in fitness between Hawk at time t (which is given by $3(1 - x(t))$) and the average fitness of the population (which is given by $2 - 2x(t)^2$). Simplifying equation 1 and writing x in the place of $x(t)$ yields:

$$\dot{x} = x(x - 1)(2x - 1)$$

This may be used to construct a diagram of dx/dt as a function of x, which is called a *phase diagram*: see Figure 2. It can be seen that the replicator dynamics for this version of the Hawk Dove games has three *rest points*, or points where the population shares do not change anymore ($dx/dt = 0$), namely $x = 0$, $x = \frac{1}{2}$, and $x = 1$. In the rest point $x = 0$, all members of the population play Dove, hence, each member's fitness is equal to the average fitness and the system is at rest. However, as soon as a Hawk comes along, perhaps through mutation, the members playing Dove are at a distadvantage (dx/dt, describing the share of Hawks, increases) and the number of members playing Hawk will grow. A similar story applies to the other side of the balance, this time Doves infiltrating and decreasing the Hawks' fitness below the average. The point in the center of the balance is the only stable rest point, where a small disturbance will shift back towards the center, where the ratio Hawks/Doves is 50/50.

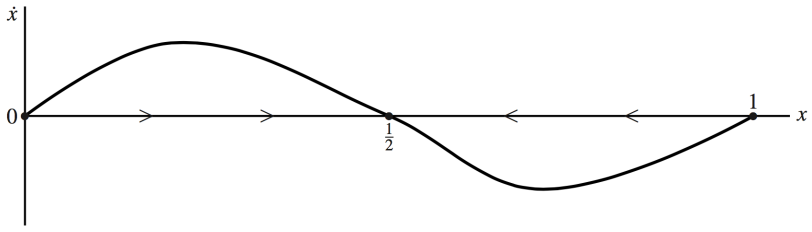


Figure 2: *Phase diagram of dx/dt as a function of x* (Peters (2008))

3.5 Uses in Disciplines

This section is based on Alexander (2009).

Evolutionary game theory started as a mathematical theory applied in biological contexts, but since then it has been in various other fields as well, such as psychology sociology, economics, philosophy, and linguistics. Some topics that have been analysed from a game-theoretical perspective thus far include[12]: altruism, behavior in the public goods game, empathy, human culture, moral behavior, private property, signaling systems and other proto-linguistic behavior, social learning, and social norms.

3.6 Prisoner's Dilemma Revisited

Let us expand on the before-mentioned example of the PD. In this section the history of this game, a more in-depth treatment of the formalization of the game, as well as research findings will be discussed.

3.6.1 Historical Context

This section is based on Poundstone (2011).

In 1950 two researchers at RAND cooperation, a back then military-subsidized think tank based in the United States, made one of the most important 'discoveries' in the field of game theory. American mathematicians Merrill Flood (1908-1991) and Melvin Dreshner (1911-1992) developed the theoretical basis for a simple, but highly informative game. Later, a consultant at RAND, Canadian mathematician Albert W. Tucker (1905-1995), constructed the famous narrative around this game, giving it the name 'Prisoner's Dilemma'. At first the game traveled through the scientific community by word of mouth, only to be published several years after its conception. Before long the game was recognized as an important analytical tool for understanding conflict situations; some even deemed it to be "one of the greatest ideas of the twentieth century" because of its simplicity and fundamental importance (Poundstone (2011)).

Interestingly, in the 1950s when the PD was 'discovered', the Western world was in a conflict-ridden time, having to deal with nuclear proliferation and arms races; the Cold War (1947-1991) is a classical illustration of a PD. The Western bloc and the Eastern bloc were in a nuclear standoff, neither willing to lay off the threat nor to actually undertake an attack. In PD terms, this can be ordered as follows. If you lay off your weapons, then either the other bloc does so as well, in which case you are both better off without the threats (R), or the other bloc procedes with the threat and attacks, since there is no deterrence from doing so

[12] See Alexander (2009) for references regarding these topics.

anymore (S). If you continue your threat, then either the other bloc does so as well and you will reach stalemate (P), or the other bloc does lay down its weapons, in which case you are free to procede and attack them (T).

The nuclear era is often ascribed to "technical progress outstripping ethical progress." Currently, technology is progessing faster than ever, again presenting a possible threat to moral considerations, making the PD arguably "one of the premier philosophical and scientific issues of our time." (Poundstone (2011))

Finally, there are quite some real-life situations may be modeled using the PD, that is, have a payoff matrix that can be written in the form of Table 7. The following two examples give an idea of how the PD can be relevant to (current) societal issues: global climate change (Rehmeyer (2012)), where each country would be better off with reduced CO_2-emissions, however, no country would like to start the restrain its own emissions; and addiction (Ainslie (2001)), where the psychological game between the current and future self may be seen as a PD, with presently relapsing being parallel to defecting and currently maintaining self-control being parallel to cooperating with the future self.

3.6.2 Formalization

This section is based on Doebeli & Hauert (2005).

The game as described in section 2.2.1 and presented in Table 2 is not the only PD: any game which payoffs can be described by the following payoff matrix and accompanying ordering of payoffs falls under this category:

	C	**D**
C	$b-c$	$-c$
D	b	0

Table 7: *PD: general payoff matrix for the row player*

The payoffs are sums of the benefits and costs of the strategy: *cooperating* results in a benefit b to the opponent, but incurs a cost c to the cooperator (where $b > c > 0$); *defection* does not result in either benefit or cost. The payoffs in Table 7 show the characteristic PD's ordering:

$$b > b - c > 0 > -c$$

which may also be written as $P_{DC} > P_{CC} > P_{DD} > P_{CD}$ where $P_{S_1S_2}$ is a player's payoff for using strategy S_1 when her opponent uses strategy S_2.

The evolutionary PD has D as its only evolutionary stable strategy (Smith (1986)), which is confirmed by the result of replicator dynamics: only pure D (i.e. $x = 1$, see section 3.4.1) is a stable equilibrium (Taylor & Jonker (1978), and Hofbauer & Sigmund (1998)).

3.6.3 Extensions and Findings

This section is based on Doebeli & Hauert (2005).

The PD has been extensively studied, focusing in particular on extensions of the game, since the dynamics of the one-shot two-player game are easily understood. In general, the research in this field aims to determine what (kinds of) strategies are most successful, but there is special attention to the question of how cooperation may be facilitated. The extensions discussed here concern extra features of interest that may be added onto the simple PD game, often doing so for two or more of these at the same time. Four categories of these extensions will be briefly discussed, after which the extension that is the focus of this thesis will be discussed in some more depth.

Iterated interactions

The iterated PD (IPD) consists of several rounds of the simple PD, which gives players the chance to react to past experience. The idea is that players may choose to cooperate (more often) in the light of the threat of reduced future payoffs due to retaliation against current defection. Whereas in the regular PD players can only choose between the strategies pure C and pure D, in this line of research the focus is on the influence of strategies that may take into account past experience. As an example, the winning strategy in Axelrod's computer tournament (see section 2.1), *Tit For Tat* (TFT), takes its opponent's previous move into account: it starts by cooperating and after that it mimics the moves its opponent makes (Axelrod (1984)). To date, the strategy that has proved most consistently successful is *Pavlov*, which takes its previous payoffs into account instead and follows the rule *win-stay, lose-shift*[13] (Kraines & Kraines (2000)).

Spatial structure

In the simple PD the population of players is *well-mixed*, meaning that the chance of a player i meeting another player j is the same for all $j \in G$, where $j \neq i$. However, this does not agree with many real-life situations[14], where there is a varying distance between players, which influences the possibility to interact together. Spatial

[13] In other words, if the previous payoff was high (T or R, see section 2.2.1) do the same move again, if the previous payoff was low (P or S) switch moves.

[14] Unless the population is sufficiently small.

games model this, placing players on a spatial network, where the players are the nodes and the connections between the nodes the possible interactions between the players. The spatial PD has been studied extensively and the results are clear: adding spatial structure to the game promotes cooperation. The mechanism behind this is the formation of cooperative clusters. Cooperators at the boundaries of these clusters are exploited by defectors, however, within the cluster they can get the high payoffs receivable for mutual cooperation, keeping them safe from extinction.

Continuous PD

In the regular PD players can only choose between the strategies pure C and pure D, however, in many situations that can be modeled by the PD it is not hard to imagine there might be a continuum of strategies between pure C and pure D available to the players. This is formalized by taking a player's investment x to lie on an the interval $[0, x_{max}]$, where 0 is the minimal possible investment (pure D) and x_{max} is the maximal possible investment (pure C). The focus in this line of research, then, is on the evolutionary dynamics of the 'trait' x. The general results echo those of the IPD.

N-player interactions

In all versions of the PD considered so far, the interactions were between two players at one time, even though they were members of a larger population. It is possible, however, to consider interactions involving more than two players at one time. These N-player games are called *Public Goods* games and have an extensive history in the field of economics (Kagel & Roth (1995)). As an example, consider a group of people, each having 10 dollars, who can all put some money into a public pot. Their investment will be tripled and divided equally over the group, resulting in 30 dollars for each player if they all invest. However, rationally no player will invest, since each invested dollar only yields a return of 50 cents to the investor. In this line of research the dynamics of Public Goods game are studied to see what may facilitate high investment (cooperation). Results indicate that smaller groups, as well as voluntary participation in the group facilitate cooperation. (Doebeli & Hauert (2005))

Reputation and Tags

In the regular PD players do not have a means of distinguishing between opponents, however, in the real world players often do have the chance to condition their strategy on their which opponent they face. One line of research in this field focuses on *reputation*. Reputation is based on the idea of *indirect reciprocity*, or 'I help you, someone else helps me'[15] (Alexander (1987)). A player's past defection affects her

[15]In contrast with 'I help you, you help me', which is the guideline in *direct reciprocity*.

image, or reputation, and influences whether current opponents are willing to cooperate with her. Another, related line of research focuses on *tag-based* games, where players cooperate only with players who are similar to them with respect to some neutral characteristic, such as color (Riolo et al. (2001), Hochberg et al. (2003), Axelrod et al. (2004)). The results of this extension cannot be briefly summarized here, due to the amount of explanation necessary.

The focus of this thesis in on spatial extension, or spatial games. These are formalized by placing players on graphs: the nodes in the graph are the players, and the connections between the nodes represent (possible) interactions. The next section introduces the concepts necessary to consider games on graphs in more detail.

4 Graphs

This section is based on Newman (2010), supplemented by Wang & Chen (2003).

Before games on graphs can be considered in more detail, it is necessary to give a brief introduction to the relevant graph-theoretical background.

4.1 Introduction to Graph Theory

Graph theory is the branch of mathematics that deals with graphs, also called *networks*. A *graph* is a collection of *vertices*, or *nodes* in computer science jargon, that are joined by *edges*, or *links*, see Figure 3. The number of vertices is called the *order* and is commenly denoted by n; the number of edges is called the *size* of the graph and is denoted by m. In this thesis the vertices represent players and the edges represent the connections, i.e. potential interactions, between them. The graphs of concern are so-called *simple* graphs: networks that do not have any *multiedges* (more than one edge between any two vertices) or *self-loops* (edges connecting vertices to themselves). Another property that will be assumed for the graphs in this thesis is *undirectedness*: an edge connects two vertices in both directions, meaning that if vertex a is connected to vertex b it is necessarily the case that vertex b is connected to vertex a as well. Finally, the graphs considered here are *unweighted*: edges are either present (1) or not present (0), there are no intermediate strengths of connection (a number between 0 and 1).

There are several ways in which graphs can be represented mathematically, the relevant one

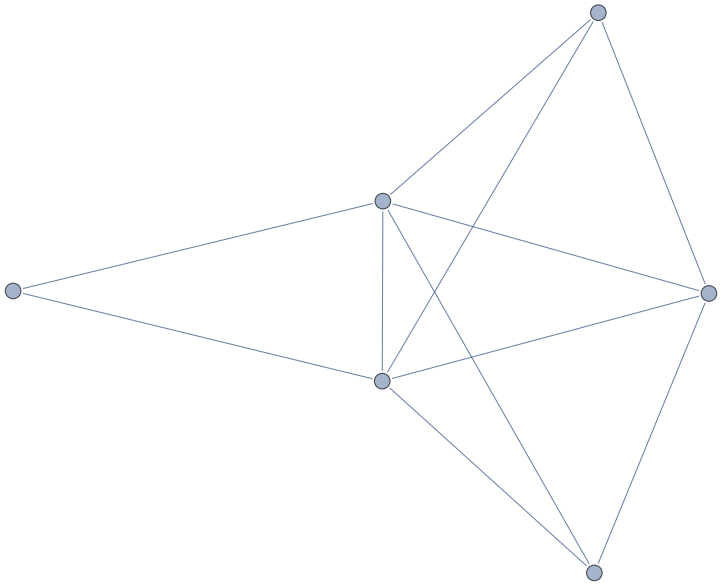


Figure 3: *Example of a graph with nodes and edges*

here being the *adjacency matrix*. Let us consider a graph with n vertices, labeled $1, ..., n$ [16] and let an edge between any two vertices i and j be labeled $\big[i, j\big]$. Then an adjacency matrix A may be constructed such that each entry for row i and column j, A_{ij} or $\big[i, j\big]$, contains either a 1, indicating a connection between players i and j, or a 0, indicating no connection. In the light of the properties assumed for the networks in this thesis, three things may be noted about the adjacency matrices: they will all be symmetric[17] due to the undirectedness of the edges; they will all have zeros in their main diagonal due to the lack of self-loops; and they will only contain zeros and ones, due to unweighted edges. An example is given by in Table 8, which is the adjacency matrix for Figure 3.

	j=0	j=1	j=2	j=3	j=4	j=5
i=0	0	0	0	1	1	1
i=1	0	0	0	1	1	1
i=2	0	0	0	1	0	1
i=3	1	1	1	0	1	1
i=4	1	1	0	1	0	1
i=5	1	1	1	1	1	0

Table 8: *Adjacency matrix for the graph in Figure 3*

[16] The labeling can be arbitrary, as long as it is unique, so each label refers to one vertex only.
[17] Unspecified symmetry in a matrix is with respect to the main diagonal.

4.2 Some Measures

Various types of information about a graph can be given, some concern the graph as a whole, others concern only a part, or even individual elements, of the graph. In this section two concepts will be discussed concerning individual vertices, which averages also provides information regarding the graph as a whole.

4.2.1 Degree Distribution

One of the most-studied concepts in the measurement of graphs is *centrality*, which addresses the question which vertices are the most important, or central, in a network. Many different measures of importance have been put forward, perhaps the simplest of which is *degree centrality*. The *degree*, as degree centrality is often called, is perhaps the most important characteristic of a single vertex and given by the number of edges connected to it. Consider an undirected graph with n vertices, then the degree k_i of vertex i is given by the sum of player i's connections (A_{ij}):

$$k_i = \sum_{j=1}^{n} A_{ij}$$

Knowing the degree for every vertex makes it possible to calculate the mean value of the degree for the entire graph by taking the sum of degrees of all vertices, divided by the number of vertices considered:

$$c = \frac{1}{n} \sum_{i=1}^{n} k_i$$

Knowing the average degree of a graph will give information on the connectedness of the vertices: the higher the degree, the more connections per vertex, thus the more connected the vertices in the graph are. A nice way of representing this information is in a *degree distribution* diagram: a histogram showing the number of vertices as a function of the number of connections. An example can be seen in Figure 4.

4.2.2 Clustering Coefficient

The notion of a *clustering coefficient* is very relevant to the analysis done in this thesis's experimental part. Before it can be defined however, the notions of *path length* and *closed triads* need to be introduced. A *path* in a graph is any route between two vertices (which may intersect itself) that runs along the edges of the network. The *path length* is the number of edges traversed along the path. If vertex a is connected to vertex b, and vertex b is connected

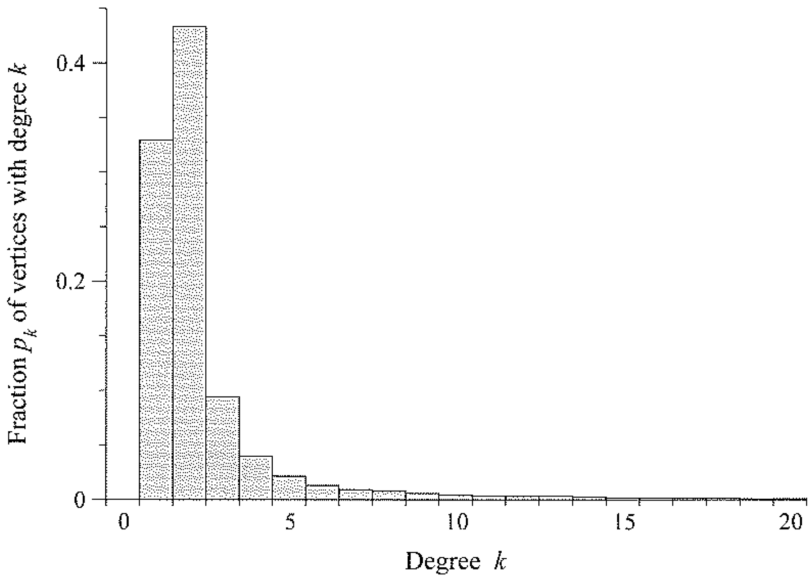


Figure 4: *The degree distribution of the internet* (Newman (2010))

to vertex c, the three vertices and their edges form a path of length two. If vertex a is also connected to vertex c, the then created path is said to be *closed*: it forms a loop of path length three, or a triangle. The vertices a, b, and c form what is called a *closed triad*.

Having defined what it means for a path to be closed, the *clustering coefficient* can be calculated as follows. In a graph, take all paths of length two, count them and check which ones are part of closed triads. Take the latter number and divide it by the former to get the clustering coefficient:

$$C = \frac{\text{(number of closed paths of length two)}}{\text{(number of paths of length two)}}$$

The range for C is between 0, meaning there are no closed triads, an example of which is a square lattice, and 1, meaning all vertices are part of closed triads, an example of which is a triangular lattice.

The way the clustering coefficient is described above focuses on it as a measure of the graph as a whole, of the general clustering of the vertices. This is not the only way in which it can be used however. Imagine there are two types of edges, say black and white. The coefficient can also be used to measure the clustering of just the black or just the white vertices. Similarly, it may be used to investigate any other property exhibited by the vertices.

4.3 Types of Graphs

There are roughly two categories of graph that may be distinguished: regular graphs and random graphs.

4.3.1 Regular Graphs

Graphs in which each vertex has the same number of edges are called *regular* graphs. There are *full* graphs, in which every vertex is connected with every other vertex; there are graphs in which the vertices are connected in some sort of pattern which matches the requirement of an equal number of edges for each vertex; and there are *nearest-neighbor* connected graphs, which are more generally known as *lattices*. In general, regular graphs are clustered, but do not exhibit the small-world effect (see section 4.3.3).

4.3.2 Random Graphs

The opposite of a regular graph is a random graph, which is created by connecting vertices with a certain probability, the *connection probability* p. The goal in the study of random graphs is to find out what influence one property of a graph has on the wider behavior of the network. In order to study this, graphs are created that exhibit one certain property, but are otherwise completely random, which are then compaired to full random graphs. An example is the study of connection probability: there is a threshold p for which connectedness (all vertices are, at least indirectly, connected to each other) in a random graph is as good as guaranteed. In general, random graphs do not exhibit clustering, but do show the small-world effect (see section 4.3.3).

4.3.3 Small-World Effect

A regular graph can be made into a random graph by rewiring some of its connections 'at random'; the more rewiring the more random. In the beginning of this process, when a small part of the connections has been rewired, the graph gains a new feature called the *smallworld effect*. This phenomenon is one of the most remarkable network phenomena and refers to the finding that many real-world networks have a small average path length between vertices, i.e. that getting from one vertex to another is remarkably easy. Thus, an advantage of the small-world property is the effective distribution of information over the vertices, increasing the likelyhood of achieving synchronization within the network, e.g. passing a piece of

information to all members in a social network. Interestingly, even though a smallworld network is made by increasing the randomness of a regular graph, by approximation it itself may be seen as a regular network as well, since, on average, each vertex has the same number of edges.

The small-world effect has been demonstrated in many empirical studies, one of which is the famous letter-passing study[18] conducted by the American social psychologist Stanley Milgram (1933-1984) in the 1960s. People were asked to help get a letter from an initial receiver to a distant target person. Out of the letter that made it to their target person, many did so in a remarkably small number of steps, with an average of six passes.

4.4 Games on Graphs

Graphs *per se* are interesting to study for the information they provide regarding the structural properties of networks, however, they are also widely used as a substrate for modeling dynamical processes *on* graphs, such as games. Comprehensive treatments and literature reviews for spatial approaches to evolutionary game dynamics, as well as evolutionary graph theory are given in Nowak & Sigmund (2000), Nowak et al. (2010), and Allen & Nowak (2014), the last of which provided the list of update rules described in this section.

4.4.1 Brief Introduction

This section is based on Doebeli & Hauert (2005).

The first to mention spatial structure as potentially interesting for the study of games was Robert Axelrod (1984), however, research only took off after the influencial paper by Austrian biomathematician Martin A. Nowak (1965-) and Australian scientist Robert M. May (1992). The idea of spatial games is to place the members of a population on a grid, or more generally, on some network (formalized by a graph). Each round all members interact (play) with all their neighbors[19], after which some update rule will be applied. The update rule used describes how the current state of the population (which members use which strategies) will update, or evolve, to the next round of the game. The creation of a 'new' generation of members of, or rather, played strategies in the population may be interpreted in one of two ways. Either it may be seen as *reproduction*, one player with a successful

[18] See Schnettler (2009).
[19] The scope of neighbors varies depending on the definition of 'neighbor'.

strategy reproducing and her offspring taking over some other player's node in the network; or as *imitation* or learning, a player adapting her strategy to that of a more successful neighbor. There are quite a number of different ways in which this strategy update may be implemented, some of which will be discussed in the next section.

4.4.2 Update Rules

One of the most important things to consider in evolutionary game theory is the type of *update rule* that will be used. An update rule refers to the way evolution is executed in an evolutionary game theoretical model. Many different versions can and have been thought of and the choice of which one to use generally influences the obtained results. A list of important update rules in presented below. Here G is the set of players in the population, i.e. the collection of vertices in the $g$raph; F_i is a function that depends on the received payoffs of player i[20], which affects the selection strength[21]; e_{ij} is the strength of the connection between the players, i.e. the edge weight in a weighted graph.

Birth-Death (BD) A player $i \in G$ is chosen at random with probability proportional to F_i to reproduce. This player's offspring will replace a neighboring player $j \in G$, who will be chosen with probability proportional to e_{ij}.

Death-Birth (DB) A player $j \in G$ is chosen at random with uniform probability to be replaced. This player will be replaced by the offspring of a neighboring player $i \in G$, who will be chosen to reproduce with probability proportional to $F_i e_{ij}$.

Imitation (IM) A player $j \in G$ is chosen at random with uniform probability to be replaced. This player will be replaced by the offspring of a neighboring player $i \in G$, *which may be player j itself*, who will be chosen to reproduce with probability proportional to F_i. Note that this update rule is only meaningful for an unweighted graph.

Pairwise Comparison (PC) A player $j \in G$ is chosen at random with uniform probability to *potentially* be replaced. This player will potentially be replaced by the offspring of a neighboring player $i \in G$, who will be chosen with probability proportional to e_{ij}. Player i reproduces with probability proportional to $\theta(F_i - F_j)$, i.e. the difference in fitness between players i and j weighted with some function θ [22], which determines

[20] This function must be positive, increasing, and differentiable, with $F(0) = F'(0) = 1$.
[21] Or how much the selection of a player depends on her received payoffs.
[22] This function must be bounded between 0 and 1 and $\theta(x) - \theta(-x)$ must be differentiable at $x = 0$.

the influence of this difference in fitness. If player i does not reproduce, player j is not replaced.

Birth-death with payoffs affecting death (BD-D) A player $i \in G$ is chosen with uniform probability proportional to F_i to reproduce. This player's offspring will replace a neighboring player $j \in G$, who will be chosen with probability proportional to $\frac{e_{ij}}{F_j}$.

Death-birth with payoffs affecting death (DB-D) A player $j \in G$ is chosen at random with probability inversely proportional to F_j to be replaced. This player will be replaced by the offspring of a neighboring player $i \in G$, who will be chosen with probability proportional to e_{ij}.

The first three of these rules will be used in the experimental part of this thesis (see section 6.6).

5 Hawk-Dove Revisited

Let us expand on the before-mentioned example of the HD game, which is the focus of the experimental part of this thesis. This game is closely related to the PD, differing only slightly in ordering of payoffs[23]. However, even though the games are very similar, the PD has been studied more extensively than the HD game, the former being used to model situations in a whole range of disciplines, while the latter has thus far been used mainly in the context of competition and escalation in animal conflict (Doebeli & Hauert (2005)). This is due to the belief that many real-life situations can be modeled as PDs, however, using empirical studies to confirm this is difficult, as determining the payoffs and thus payoff ordering in a real-life situation is challenging. The HD game is another model that may be applied and is a "viable and biologically interesting alternative" to the PD (Hauert & Doebeli (2004)).

There are two other games which are identical to the HD game[24], namely *Chicken* and the *Snowdrift* game[25]. The narrative for the Chicken game is as follows: two drivers are on collision course and in order to avoid the hit at least one of them has to diverge from the course he's on. Here moving aside is to similar to playing Dove (D), while to stay on course is similar to playing Hawk (H). The narrative for the Snowdrift game uses the PD

[23]Thus is discussed in depth in the first subsection of this section (5.1).

[24]The HD game is characterized by its payoff ordering, as will be discussed inthe first subsection of this section (5.1), not by the story accompanying it.

[25]This is due to parallel development in different research areas (Osborne & Rubinstein (1994)).

nomenclature of *cooperate* and *defect*. Again, consider two drivers, this time they have been caught in a snow storm and are stuck in a snowdrift. Digging the cars out is to *cooperate* (C), whereas abstaining from digging is to *defect* (D). The payoffs for both these games have the characteristic Hawk-Dove ordering, respectively:

$$P_{HD} > P_{DD} > P_{DH} > P_{HH} = P_{DC} > P_{CC} > P_{CD} > P_{DD}$$

In the rest of this section a more in-depth treatment of the formalization of the game is provided, as well as relevant research findings.

5.1 Formalization

This section is based on Newman (2010), supplemented by Doebeli & Hauert (2005).

In order to compare the HD game's payoffs and characteristic payoff ordering to that of the PD, it is convenient to use the same nomenclature for the strategies. This means that, in this section, the nomenclature of the *Snowdrift* game will be used: playing Dove is equal to *cooperating* and playing Hawk is equal to *defecting*. Similarly to the PD, the payoffs for the HD game in Table 6 can be described more generally as:

	C or Dove	**D** or Hawk
C or Dove	$b - \frac{c}{2}$	$b - c$
D or Hawk	b	0

Table 9: *HD: payoff matrix for the row player*

Again, the payoffs are sums of the benefits and costs of the strategy, however with a small difference: *cooperating* results in a benefit b to *both* the cooperator and the opponent, but incurs a cost c to the cooperator; *defecting* does not result in either benefit or cost (where $b > c > 0$). The ordering of the payoffs is therefore slightly different, creating the characteristic HD-ordering

$$b > b - \frac{c}{2} > b - c > 0$$

which may also be written as

$$P_{DC} > P_{CC} > P_{CD} > P_{DD}$$

In other words, cooperating against a defector is more beneficial than defecting against a defector. This means that cooperators have an advantage when rare, leading the replicator

dynamics of the HD game to converge to a mixed stable equilibrium, containing both cooperators and defectors[26].

Note that the HD game converts to the PD when the costs are high: $2b > c > b$. Hence, the HD game may be seen as a relaxed version of the PD, in which the procured benefits are shared between both interacting players, and the incured costs are shared between cooperators.

5.2 Analysis

This section is based on Hauert (2008).

In games with two players the stable Nash equilibrium for the HD game can be determined using the method of evolutionary stable strategies. Given the set of actions available to either player i is $A_i = \{H, D\}$ (see section 2.2.2), the mixed strategy vector $(p_1, p_2) = (.5, .5)$ is the only ESS in the game.

For games involving more than two players, replicator dynamics provide a calculation of what strategy state the population as a whole may be expected to evolve into. Using the procedure that is laid out in section 3.4.3 for the HD game in Table 9 yields the replicator dynamics

$$\dot{x} = x(1-x)\big[b - c(1 - \frac{x}{2}) - xb\big]$$

In contrast to the PD replicator dynamics, which have $x = 1$ as the stable rest point (see section 3.6.2), these replicator dynamics have as a stable rest point

$$x = \frac{c}{2b - c} \tag{2}$$

Hence, the stable proportion of Hawks, and consequently Doves, depends on the relation between the values for the costs c and benefits b.

Finally, note that these results pertain to well-mixed populations; if this assumption does not hold the results will be different, as will be seen in part II.

5.3 Extensions and Findings

This section is based on Doebeli & Hauert (2005).

[26] The precise proportion of C and D, however, depends on the payoffs, which will become clear in section 6.3.

Similarly to the PD, extensions[27] to the Hawk-Dove model have been studied as well, though significantly less so.

Additional strategies

Several new strategies, additional to pure D and pure H have been put forward. An example is *Retaliator*, which bases its action on the action of the other player: it starts by playing Dove, but if the other player plays Hawk, it switches to Hawk as well. The dynamics in games with additional strategies lead to different rest points and may tell us about real-world animal conflicts, since mixed and/or conditional strategies model reality better.

Iterated interactions

Iteration in the HD game promotes cooperation, as is the case in the PD (Dubois & Giraldeau (2003) and McElreath (2003)).

Spatial structure

In the PD adding spatial structure promotes cooperation, however, interestingly this is not the case for the HD game. The study that the experimental part of this thesis is based on, Hauert & Doebeli (2004), found that spatial structure inhibits the propagation of cooperators, or Doves. More on this in section 7.1.

N-player interactions

Public Goods games for the HD game are different from those in the PD. Whereas in the latter, each additional cooperator increases the benefits to the group with the same amount as the first, in the former, the benefits each additional Dove provides to the group is less than the previous Dove (Hauert et al. (2006)). Not much research has been done in this field yet.

Continuous PD

Whereas the dynamics of the trait x are quite sensible in the PD, they are much less so in the HD game and they can display some suprising features. There are three types of evolutionary dynamics that can occur: trait x may monotonically increase or decrease, leading to a pure D or pure H state; trait x may evolve to some intermediate x^* leading to an ESS; or trait x may evolve to some intermediate x^* leading to an 'evolutionary branching point', which makes the population develop into two diverging groups, each with its own distinct strategy.

In this thesis the focus lies on extending the HD game spatially, therefore extra attention is paid to the discussion of the spatial extension here. Giving a game a spatial structure

[27] See section 3.6.3 for explanations of the different extensions.

means that players do not randomly interact, but only with those sufficiently close to them in space. Adding this feature to the game makes the model more realistic, since interactions in the real world are bound by space as well.

Part II

The Experiment

Part two of this thesis is devoted to describing the experimental program written for the HD game on graphs. First the program written will be explained step-by-step, walking through the line of thought of the code. After this, the input from the user and the output created by the program are discussed. Finally, the results of the program are analyzed and some concluding words are provided.

The research question that will be addressed is this: in the analysis in section 5.2, the results described are for *well-mixed* populations, however, how will the results be different if the population is *spatially ordered* instead? The aim is to reproduce the results obtained by Hauert & Doebeli (2004) and study some additional factors which may influence the stable rest point for populations on graphs.

6 The Program Step-by-Step

The idea of the program is to create a model containing players on a graph. Each player has a strategy and interacts with other players in the vicinity (section 6.2), obtaining payoffs (section 6.3) depending on her own strategy and the strategy her neighbor has (section 6.4). One evolution (section 6.5) consists of all players playing their neighbors and getting payoffs, after which the graph is updated according to an update rule (section 6.6). In order to run the program some input is asked from the user, after which the program generates its output (section 6.7).

Note that the description of the program as provided in this section is general, leaving out intricasies such as precise selection procedures and special cases that need explicit mentioning. These, however, can be found in the code itself (see appendix A), complete with comments in human language to explain the process.

6.1 Process of Writing the Program

The program written for this thesis is provided in appendix A and contains the c++ code, as well as comments for most steps in the code, explaining them in human language. At the

start of this thesis I did not have any programming experience, other than a small 2.5 ECTS course in c-language. I taught myself on the way, using online resources. This learning process is visible in the code, which is, at times, longer and less logical than possible.

Finally, note there was no special reason for chosing to program in c++ for this particular project, other than that I had the wish to learn a versatile and powerful low-level programming language.

6.2 Types of Graphs

The first array to be created is the `connection array`, which contains the adjacency matrix (see section 4.1) for the graph on which the players will be vertices and their connections will be edges. In the program three different graphs can be chosen by the user: a regular graph, a random graph, or a smallworld graph (see section 4.3). Regardless of the choice between graphs, the program will start by generating an `n`×`n`-array and initialize its entries to zero.

Regular

The code for the regular graph, a square lattice, was written later on in the process of building the code. By that time I had started using functions more, rather than defining processes in a more mechanistic manner.

The idea is to loop through each player (`i` is `0` till `n-1`) and for each of them fill four entries in the `connection array` with a `1`: for each of the four neighbors of player `i`. In order to know which players neighbor player `i` at the right, left, bottom, and top, four functions are defined, which output the numbers of the players connected to player `i`.

Random

A `for-loop` loops through each row `i`, and within this loop a second `for-loop` loops through each column `j` within the considered row `i`, effectively considering every entry in the array. For each entry a `0` or `1` is generated at with probability `conprob`, which is calculated using `n` such that every player has, on average, four connections.

Smallworld

To start, a regular ring graph[28] is created by filling the entries for row `i` with a `1` if the column player is a neighbor or next-to-neighbor, i.e. `j=i-1` and `j=i+1` or `j=i-2` and

[28] Intuitively this may be pictured as a circle of people holding hands, each connected to her two neighbors only.

`j=i+2`, and a `0` if this is not so. Special cases are the first, and second row players, `i=0` and `i=1`, which are connected to the last column player (`j=n-1`), and to the second-to-last and last (`j=n-1` and `j=n-2`) column players, respectively. A similar case applies to the last and second-to-last row players, `i=n-1` and `i=n-2`.

Next, the connections, i.e. `1`'s in the `connection array`, in the regular ring graph are rewired with some inputted probability `beta`. 'Rewiring' means that connection of player `i` with some player `j` will be broken and replaced with a connection with some player `k` [29].

Each of these procedures is only done for the entries to the right of the `0`-diagonal, after which the entries to the left are filled by copying their mirror images on the right.

6.3 Payoffs

The `2×2 - payoff array` is filled with payoffs consistent with the HD game. There are two payoff matrices that the user may choose: `hd123` as given in Table 10, or `hd91011` as given in Table 11.

	0 = Dove	**1** = Hawk
0 = Dove	2	0
1 = Hawk	3	1

Table 10: *Hawk-Dove payoff matrix* `hd123`

	0 = Dove	**1** = Hawk
0 = Dove	10	0
1 = Hawk	11	9

Table 11: *Hawk-Dove payoff matrix* `hd91011`

Using replicator dynamics (equation 2), we expect to find the following for a well-mixed population: the proportion of Doves for `hd123`:

$$1 - x = 1 - \frac{c}{2b - c} = 1 - \frac{2}{2 \cdot 3 - 2} = 0.5$$

and for `hd91011`:

$$1 - x = 1 - \frac{c}{2b - c} = 1 - \frac{2}{2 \cdot 9 - 2} = 0.9$$

[29] There are two requirements for `k`: player `k` is equal to player `i`, and `k` and `i` were not yet connected to each other.

Thus, since Hauert & Doebeli found that adding spatial structure to the HD game decreases the proportion of Doves, the expectation is that the average proportion of Dove for `hd123` will be less than 0.5, and for `hd91011` less than 0.9.

6.4 Strategies

The `strategy array` defined may be seen as a `n`-dimensional vector where each entry `i` contains the strategy of player `i`. It is first defined and initialized with zeros, and then filled with either a `0`, meaning Dove, or a `1`, meaning Hawk, at random.

6.5 Evolutions

One round, evolution, or `iteration`, of the game consists of all players on the graph playing their neighbors, after which their strategies will be updated. At the end of each iteration some players will mutate according to some probability inputted by the user.

Score array

Define the `score array`, a `n`-dimensional vector where each entry `i` contains the accumulative score of player `i` during the current iteration. Fill it by looping through the `connection array` and, using the `payoff array`, assigning scores for each connection to the two players involved.

Normalized reproductive rate

The scores in the `score array` are used to calculate the `reproductive rate`, which is a function depending on a player's score. The amount in which this function depends on the player's score may be varied using the *selection strength* parameter `delta` (a greater `delta` means a bigger dependence on the received payoff). Normalizing this rate gives the `normalized reproductive rate`, which is the chance a player within a population has to reproduce, e.g. a normalized reproductive rate of `0.06` means a 6% chance of reproducing.

Update-loop

This is where the update-loop starts, which depends on the update-rule inputted by the user. The next section (6.6) is dedicated to discussing this in more detail.

Mutation

After the players' strategy array has been updated, some players will mutate. The

user inputs two probabilities: the rate of overall mutation as a percentage of `n`, which gives the number of players in the population that will possibly mutate; and the rate of individual mutation as a proportion of the selected players, which gives the probability a selected player will actually mutate. These are then used to select players and have them 'switch' strategies, i.e. go from `0` to `1`, or vice versa.

Count cooperators

The number of cooperators in the `strategy array` are counted[30] . This number is then converted to a proportion of `n` and printed, together with the number of the iteration, to the `trackc array`.

This whole process is put in a `loop`, which continues for the number of iterations inputted by the user. Note, that since the `strategy array` is defined outside this loop it persists after each iteration. This is a record of history of the population, thus making continuous evolution possible.

6.6 Update rules

Three of the six update rules described in section 4.4.2 have been used in the code written for this thesis[31]: Birth-Death (BD), Death-Birth (DB), and Imitation (IM).

Birth-Death

The `birthdeath` code consists of the following steps:

- *Selection player `i`*
  First, some player `i` is selected from all players `n` to reproduce, however, players have a probability to be chosen proportional to their `normalized reproductive rate` (`Fi`). The selection is done by creating and filling a `cumulative distribution array`; a `n`-dimensional vector. In a loop `i=0` to `i=n-1`, three processes take places for each player `i`. First, entry `i` in the `cumulative distribution array` is filled with the sum of the `normalized reproductive rates` of players `0` till `i`. Second, a random number between 0 and 1 is generated. Finally, this number is checked against the entry in the `cumulative distribution array`: if the random number is smaller than the number stored in the array, the loop stops and the considered player `i` is selected reproduce.

[30] This does not happen for all iterations, just for a subset of them: more on this in section 6.7 in the description of `datapoints`.

[31] Note that the edge weight, e_{ij}, is 1 in this thesis, since the graphs are unweighted.

- *Selection player* `j`

  The selection of player `j` is partly similar to that of player `i`. First, player `i`'s neighbors are determined by counting her connections in the `connection array`. Then, a loop similar to the one used in the selection of player `i` is started, which goes through the same process, except for one difference: the chance of being selected is uniformly distributed over the neighbors of i.

- *Replacement of player* `j`

  Finally, after both players have been selected, player `j`'s strategy is replaced by that of player `i` in the `strategy array`.

As may be seen from the descriptions given in section 4.4.2, the update rules BD, DB and IM are very similar. The code used for BD, therefore, can be rearranged and slightly tweaked in order to produce these latter two rules. For **Death-Birth**, it suffices to switch the two selection procedures around[32]. For **Imitation**, it suffices to take the code for DB and tweak it such that player `j` can also be chosen in the selection for player `i`, i.e. the player chosen to be replaced may 'replace' herself.

6.7 The Input and Output

As may be taken from the above descriptions, there are a number of parameters that are inputted by the user. The input will later-on be used in labeling the graphs that display the results of the program (see section 7.3.2). In order to read these easily and correctly, below is an overview of the different parameters and the input they take:

parameter	Command(s) that may be put in
n	$\in \mathbb{N}$ is the number of members in the population
iterations	$\in \mathbb{N}$ is the number of rounds all members of the population play their neighbors
game	The payoff matrix used[33]: • `hd123`: uses $P_{HD} > P_{DD} > P_{DH} > P_{HH} = 3 > 2 > 1 > 0$ • `hd91011`: uses $P_{HD} > P_{DD} > P_{DH} > P_{HH} = 11 > 10 > 9 > 0$

[32] There are some adjustments that need to be made, because of the following difference between BD and DB. In BD choosing `i` is from all `n` and proportional to `Fi`, and choosing `j` is from only `i`'s neighbors and uniform. In DB, choosing `j` is from all `n` and uniform, and choosing `i` is from only `j`'s neighbors and proportional to `Fi`.

[33] More on the influence of this in section 6.3.

graph The type of graph used:

- `regular`: square lattice, every member has exactly four neighbors; one to the right, left, top and bottom
- `random`: connections between members are created with probability `conprob` $\in [0, 1]$, which is calculated by the program such that each member has, on average, four connections
- `smallworld`: connections in a regular ring graph are rewired with probability **beta** $\in [0, 1]$; each member starts by being connected to its two neighbors and the neighbors of those neighbors

updaterule The kind of update rule used:

- `birthdeath`: update rule BD
- `deathbirth`: update rule DB
- `imitation`: update rule IM

birthrate $\in [0, 100]$ is the percentage of `n` that will reproduce

delta $\in [0, 1]$ describes the amount of dependence of the `reproductive rate` function `Fi` on the relative received payoff of a member within the population: a greater `delta` means a member's score has a greater influence on her chance to reproduce

mutation Whether or not mutation occurs after each round:

- `yes`:

  overallmutrate $\in [0, 100]$ is the percentage of `n` that will possibly mutate

  indmutrate $\in [0, 1]$ is the chance each of the possibly mutating members have to actually mutate
- `no`

datapoints $\in \mathbb{N}$ is the number of times the number of cooperators in the population is counted, e.g. `iterations` $= 10000$, if `datapoints` $= 20$, then after every $\frac{10000}{20} = 5000$ iterations the number of cooperators in the current strategy array will be counted

The program will ask the user the following:

```
Number of players; Which game; Type of graph; Connection
probability; Rewiring probability; Number of iterations; Update
rule; Rate of birth; Selection strength delta; Mutation; Rate of
overall mutation; Rate of individual mutation; Number of
datapoints =
```

In response, the user may input something of the form:

```
9 hd123 regular 0.1 10000 imitation 20 0.5 no 20 0.5 100
```

which runs the program with `n` = 9, `game` = `"hd123"`, etc.

An example of what comes out of the program is:

```
Connection array for regular graph:
011100100
101010010
110001001
100011100
010101010
001110001
100100011
010010101
001001110

Payoff array:
21
30

Generated strategy array:
1
1
1
1
0
1
0
0
0
```

```
Final strategy array (including updates and mutations):
1
1
1
1
1
1
1
1
1

Average proportion of cooperators = 0.00440044
```

7 Analysis

Research on the PD and the HD game (in Snowdrift nomenclature) focuses on the proportion of cooperators in the population and how this is influenced by different factors, such as iteration, spatial structure, payoffs, etc. (see section 3.6.3). In this section the aims of the analysis, the procedure necessary to work towards those aims, as well as the obtained results are discussed.

7.1 Aims

The two aims of analysis for the program are the following. First, to reproduce the findings from Hauert & Doebeli (2004), and second, to study some additional factors that may influence the proportion of cooperators in the population.

First aim

A study conducted by Christoph Hauert and Michael Doebeli from the University of British Columbia analyzed the Snowdrift game[34] on graphs (2004). As seen in section 3.6.3, spatial structure in the PD *promotes* cooperation, however, Hauert and Doebeli found that spatial structure in the HD (using Snowdrift nomenclature) *inhibits* cooperation. At the time of publication, Hauert and Doebeli's finding was

[34]Which is equivalent to the HD game, see section 5.1.

quite groundbreaking, since spatial structure was seen as "necessarily beneficial for cooperative behavior", which turns out is not the case.

Hauert and Doebeli used a 100x100 square lattice, where each player had four connections[35]. The update rule used is similar to Pairwise Comparison (see section 4.4.2), except there is no selection of players j that will possibly be replaced. Instead, every site undergoes potential replacement.

Second aim

After running the program with a square lattice and the imitation[36] update rule in order to reproduce Hauert & Doebeli's results, the second aim is to expand the scope of investigation and look at other factors: different types of graphs, different update rules, and different payoffs.

7.2 Procedure for Obtaining Data

In order to reach the two just-described aims, two methods of data collecting were used: 1) tracking the proportion of cooperators and calculating the meta-average, as well as plotting the obtained data , and 3) creating a visual representation of the spatial distribution of Hawks and Doves for the cases involving a `regular` graph, i.e. square lattice, in order to investigate clustering of strategies.

7.2.1 Proportion of Cooperators

At the end of the iteration-loop, for a subset of the iterations, the number of cooperators is counted, converted to a proportion of the population and printed to the `trackc array` (see section 6.5). From the values in the `trackc array` an average proportion of cooperators per iteration is calculated: `averagepropc`. For the analysis, an additional for-loop is put around the entire program, running it ten times in a row. After this loop, all ten values for `averagepropc`, which are printed to a new array `metatrackc`, are averaged again, leading to one meta-average: `metaaveragepropc`. These meta-averages of the proportion of cooperators are generated for different versions of the game, identified by the input, or 'fingerprint', described at the end of section 6.7, allowing their comparison.

In addition to meta-averages, the fluctuations within one run of the program, over the `10000` iterations, can be plotted using the data stored in that run's `trackc array`. At the end of

[35] They used variations as well, lattices where each player had six connections, for instance.

[36] Out of the update rules written for this program, imitation is most similar to pairwise comparison as used by Hauert & Doebeli.

the program some code is written which converts the filled `trackc array` into a form such that Mathematica is able to plot the data in it (see section 7.3.2). The plots thus created are labeled using the fingerprint described in section 6.7.

7.2.2 Clustering of Cooperators

There is an interesting difference between clustering as seen in the PD and clustering as seen in the HD game, as noticed by Doebeli & Hauert (2005). Because in the PD the cooperators at the edges get exploited, the clusters tend to have small (disadvantageous) boundaries compared to their volume. In contrast, in the HD game it is advantageous to have a strategy different from your neighbor, leading to filament-like clusters, i.e. with large (advantageous) boundaries compared compared to their volume. In order to investigate this in the program for this thesis as well, for a couple of versions of the game on a regular graph a visual representation of the lattice will be created, showing the strategies of the occupants of the lattice.

7.3 Results and Discussion

In this section the results obtained with the just-described procedure will be presented, as well as discussed.

7.3.1 Variations

A number of factors have been varied in order to assess its influence on the average proportion of Doves. An overview of what will and what will not be analyzed in this section is provided here.

Negligible to no influence
A shallow analysis of the following factor have shown it to have little to no influence on the obtained results:

- *The number of iterations*
  For versions of the game that settle in a state with no Doves, there is a threshold of iterations, a minimum number that needs to have been run, in order for this end state to certainly be reached. Other than the, the results do not change either qualitatively or quantitatively when changing the number of iterations.

Possible influence

A shallow analysis of the following factors indicates a possible influence on the obtained results. As investigations of these factors lie outside the scope of this thesis, further research is necessary.

- *The number of players*
  Some results have indicated that the higher the number of players, the lower the average proportion of Doves. This is not readily explicable from the theory of the game, hence, further analysis must show whether this is actually the case, and if so, what is the mechanism behind it.

- *The degree of the graph*
  The degree for all three graphs is set to exactly or on average four. Changing this influences the connectedness of the graph and is thus logically expected to influence the dynamics on the graph.

- *Mutation*
  The program contains an extensive part dedicated to applying mutation at the end of each iteration, however, analyzing this unfortunately turned out to be beyond the scope of time available. What may be noted, is that mutation is a kind of disturbance of the patterns obtained without mutation, hence, effects such as a change in the amount of fluctation of the proportion Doves, or Doves coming back into the population after extinction are expected and seem indeed to be present.

- *Parameters used*
  The parameters inputted by the user, `beta`, `birthrate`, `delta`, `overallmutrate`, and `indmutrate`, are expected to have significant influences on the stable rest point of the HD game, hence their presence as a parameter. Their influence has not been tested.

Studied influence

The analysis in this thesis focuses on exploratory research into the influence of the following three factors:

- *Type of update rule*
  Three versions: imitation, deathbirth, and birthdeath.

- *Type of graph*
  Three versions: regular, random, and smallworld.

- *Payoffs*
  Two versions: hd123, and hd91011.

For a more in-depth discussion of these factors, see section 6.7.

7.3.2 Data: Proportion of Cooperators

The analysis focused on investigating the influence of three factors: the type of update rule (`imitation`, `deathbirth`, `birthdeath`), which payoffs (`hd123`, `hd91011`), and the type graph (`regular`, `random`, `smallworld`).

The following data were obtained:

	hd123	**hd91011**
regular	0.307861	0.568673
random	0.320574	0.552238
smallworld	0.330356	0.533753

Table 12: *Meta-averages of proportion of cooperators for* **imitation**

	hd123	**hd91011**
regular	0.337376	0.548901
random	0.356020	0.537782
smallworld	0.331921	0.528881

Table 13: *Meta-averages of proportion of cooperators for* **deathbirth**

	hd123	**hd91011**
regular	0.00908911	0.532327
random	0.0557426	0.521554
smallworld	0.277941	0.55398

Table 14: *Meta-averages of proportion of cooperators for* **birthdeath**

Concerning the first aim:
Of the three update rules written into the program, `imitation` is most similar to Pairwise Comparison, as used by Hauert and Doebeli. Consider Table 12, which, in the middle column displays the meta-averages for `hd123` and for the second row gives these for a `regular` graph. For these payoffs replicator dynamics predict (see section 6.3) the simulation to reach a stable rest point for a proportion of Dove = 0.5, if the population were to be well-mixed.

Based on the results for Hauert and Doebeli, we expected this proportion to be less, which is indeed what we see: 0.307861. The same goes for the meta-average of `hd91011` on a `regular` graph and with the `imitation` update rule: 0.568673, which is smaller than the expected 0.9. Hence, these results succesully reproduce Hauert and Doebeli's results.

Concerning the second aim:
Based on Tables 12, 13, and 14, the following observations may be made regarding the three factors considered:

`updaterule`

The update rules `imitation` and `deathbirth` produce similar results (compare Tables 12 and 13), however, update rule `birthdeath` produces results that deviate from the other update rules, though only for `hd123` and not for `hd91011`.

`game`

The type of payoff used makes a big difference, as expected (see section 6.3), with `hd123` yielding a much lower stable proportion of Doves than `hd91011`.

`graph`

For the update rules `imitation` and `deathbirth`, the type of graph does not seem to make a difference in the obtained results (compare rows in Table 12 and in Table 13), however, for the update rule `birthdeath` and payoff `hd123` there are noticable differences between the three types of graphs (see the middle column Table 14). Interestingly, the type of graph used does not influence the results for the update rule `birthdeath` and payoff `hd91011`.

The work done here is mainly exploratory and (thus) descriptive. Finding an explanation for these differences requires additional, in-depth, mathematical analysis, which the scope of this thesis, unfortunately, does not allow space nor time for.

Visualization of the proportion of Doves:
To illustrate, the following four plots show the fluctuations of the proportion of Dove within one run of the program, i.e. over `10000` iterations.

Figure 5a shows the data for a run of the program with `hd123`, `regular`, and `imitation`, however, this plot is characteristic for `hd123`, for all three types of graphs for both update rules `imitation` and `birthdeath` (see Tables 12 and 13). Figure 5b shows the data for a run of the program with `hd91011`, `regular`, and `imitation`, however, this plot is characteristic for `hd91011`, for all three types of graphs for both update rules `imitation`

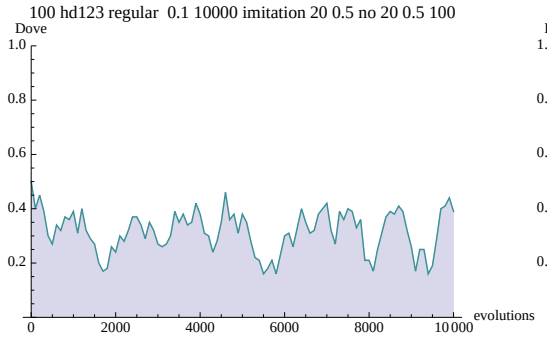
and `birthdeath` (see Tables 12 and 13). Figures 5c, 5d, and 5e show the data for a run of the program with `hd123`, `birthdate` and, respectively, `regular`, `random`, and `smallworld` (see Table 14). Figure 5f shows the data for a run of the program with `hd91011`, `regular`, and `birthdeath`, (see Table 14).

7.3.3 Data: Clustering of Cooperators

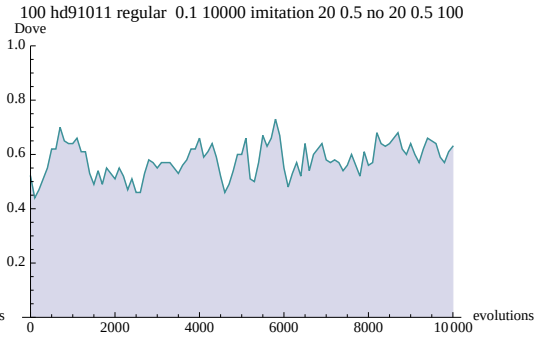
Cooperating in the PD is promoted by spatial structure through the formation of cooperative clusters (see section 3.6.3). It is therefore interesting to take a look at cluster formation in the HD game. Figure 6 shows the spatial structure for a regular graph of 10 x 10 players for three different versions of the HD game. It can be seen from Figures 6, in particular from Figure 6a, showing the regular graph for `hd123` and `imitation`, that the Doves present (the red nodes) are clustered together. The observed difference in clustering notices by Doebeli & Hauert (2005) seems to be present: the clustering in Figure 6 does have a large boundary surface compared to its volume.

8 Conclusion

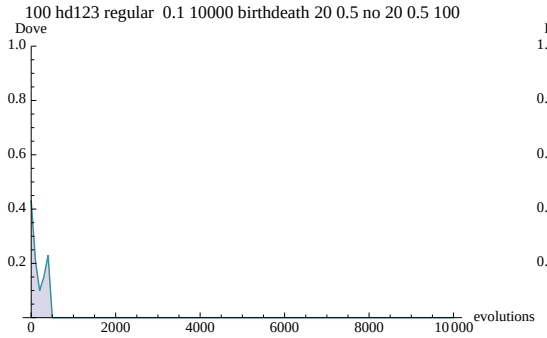
In conclusion, in order to be able to write a program and study an evolutionary game on graphs, background knowledge from a wide array of fields is required. This has been a major learning experience, providing insights into a number of different fields, ranging from mathematics and computer science, to biology and psychology, to philosophy. As a result, the analysis of the results obtained from the program written is more descriptive and exploratory, which is logical in the light of my main goal for this thesis: to learn to set up my own experiment, program in `c++`, and write a report on the findings.



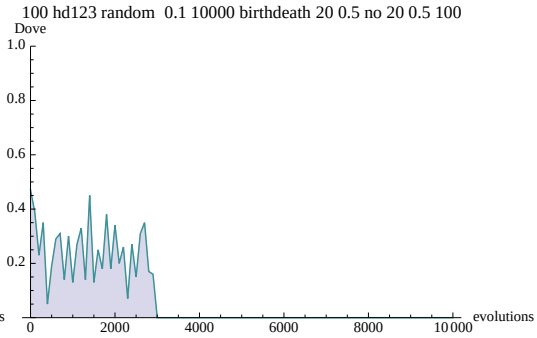
(a) Average proportion of Doves `hd123`, `regular`, `imitation`



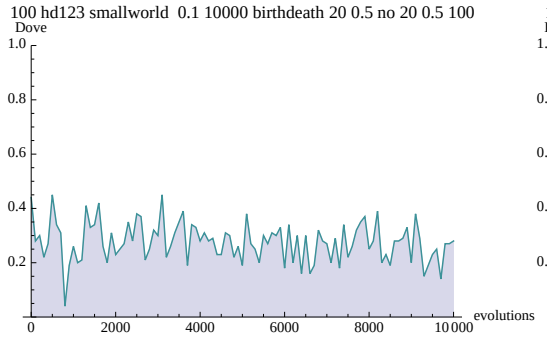
(b) Average proportion of Doves for `hd91011`, `regular`, `imitation`



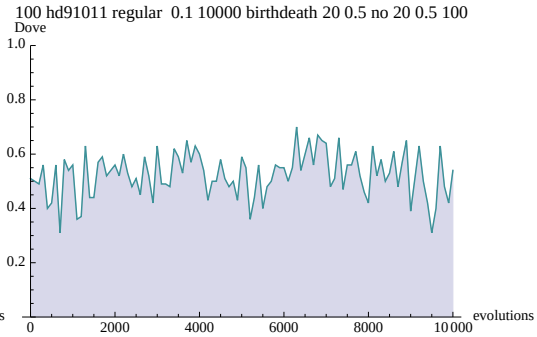
(c) Average proportion of Doves for `hd123`, `regular`, `birthdeath`



(d) Average proportion of Doves for `hd123`, `random`, `birthdeath`

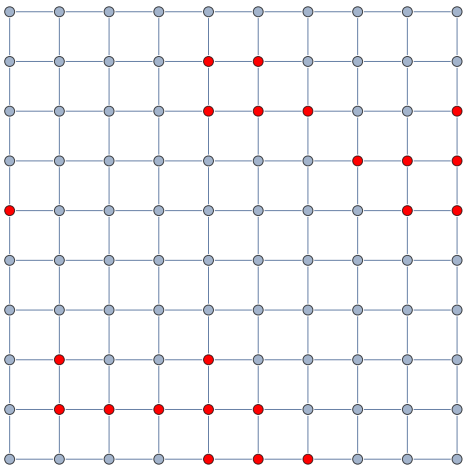


(e) Average proportion of Doves for `hd123`, `smallworld`, `birthdeath`

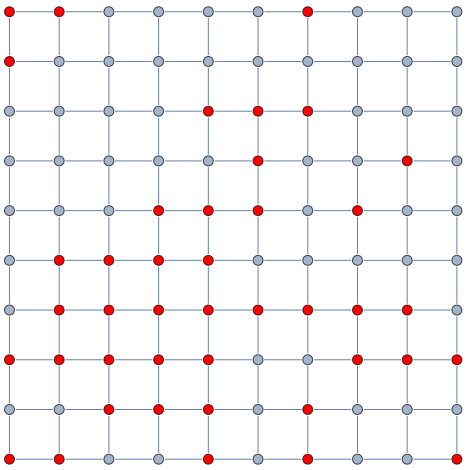


(f) Average proportion of Doves for `hd91011`, `regular`, `birthdeath`

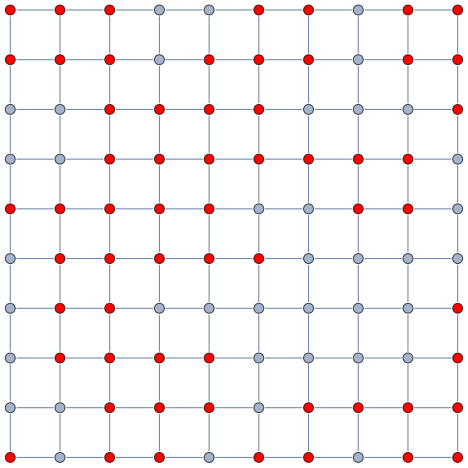
Figure 5: *Plots of the average proportion of Doves as a function of the number of iterations*



(a) Strategy distribution for `hd123` and `imitation`



(b) Strategy distribution for `hd123` and `deathbirth`



(c) Strategy distribution for `hd91011` and `imitation`

Figure 6: *Clustering of strategies: grey nodes represent Hawks, red nodes represent Doves*

References

Ainslie, G. (2001), *Breakdown of will*, Cambridge University Press.

Alexander, J. M. (2009), *Evolutionary Game Theory*, fall 2009 edn.

Alexander, R. D. (1987), *The biology of moral systems*, Transaction Publishers.

Allen, B. & Nowak, M. A. (2014), 'Games on graphs', *EMS surveys in mathematical sciences* **1**(1), 113–151.

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

Axelrod, R., Hammond, R. A. & Grafen, A. (2004), 'Altruism via kin-selection strategies that rely on arbitrary tags with which they coevolve', *Evolution* **58**(8), 1833–1838.

Doebeli, M. & Hauert, C. (2005), 'Models of cooperation based on the prisoner's dilemma and the snowdrift game', *Ecology Letters* **8**(7), 748–766.

Dubois, F. & Giraldeau, L.-A. (2003), 'The forager's dilemma: Food sharing and food defense as risk-sensitive foraging options', *The American Naturalist* **162**(6), 768–779.

Fisher, R. (1930), 'The genetical theory of natural selection.'.

Hauert, C. (2008), *Evolutionary dynamics*, Springer, pp. 11–44.

Hauert, C. & Doebeli, M. (2004), 'Spatial structure often inhibits the evolution of cooperation in the snowdrift game', *Nature* **428**(6983), 643–646.

Hauert, C., Michor, F., Nowak, M. A. & Doebeli, M. (2006), 'Synergy and discounting of cooperation in social dilemmas', *Journal of theoretical biology* **239**(2), 195–202.

Hochberg, M. E., Sinervo, B. & Brown, S. P. (2003), 'Socially mediated speciation', *Evolution* **57**(1), 154–158.

Hofbauer, J. & Sigmund, K. (1998), *Evolutionary games and population dynamics*, Cambridge University Press.

Hykšová, M. (2004), 'Several milestones in the history of game theory', *Jubiläen–Chance oder Plage* pp. 49–56.

Jackson, M. O. (2011), 'A brief introduction to the basics of game theory', *Available at SSRN 1968579* .

Kagel, J. H. & Roth, A. E. (1995), *The handbook of experimental economics*, Princeton university press Princeton, NJ.

Kooistra, S. (2012), Logic in classical and evolutionary games, PhD thesis, Citeseer.

Kraines, D. P. & Kraines, V. Y. (2000), 'Natural selection of memory-one strategies for the iterated prisoner's dilemma', *Journal of Theoretical Biology* **203**(4), 335–355.

Lewontin, R. C. (1961), 'Evolution and the theory of games', *Journal of theoretical biology* **1**(3), 382–403.

Leyton-Brown, K. & Shoham, Y. (2008), 'Essentials of game theory: A concise multidisciplinary introduction', *Synthesis Lectures on Artificial Intelligence and Machine Learning* **2**(1), 1–88.

McElreath, R. (2003), 'Reputation and the evolution of conflict', *Journal of Theoretical Biology* **220**(3), 345–357.

Nash, J. (1949), Non-cooperative games, PhD thesis, Princeton University.

Nash, J. (1951), 'Non-cooperative games', *Annals of mathematics* pp. 286–295.

Nash, J. F. et al. (1950), 'Equilibrium points in n-person games', *Proc. Nat. Acad. Sci. USA* **36**(1), 48–49.

Neumann, J. v. (1928), 'Zur theorie der gesellschaftsspiele', *Mathematische Annalen* **100**(1), 295–320.

Newman, M. (2010), *Networks: an introduction*, Oxford University Press.

Nowak, M. A. (2006), 'Five rules for the evolution of cooperation', *science* **314**(5805), 1560–1563.

Nowak, M. A. & May, R. M. (1992), 'Evolutionary games and spatial chaos', *Nature* **359**(6398), 826–829.

Nowak, M. A. & Sigmund, K. (2000), 'Games on grids', *The Geometry of Ecological Interactions: Simplifying Spatial Complexity* pp. 135–150.

Nowak, M. A., Tarnita, C. E. & Antal, T. (2010), 'Evolutionary dynamics in structured populations', *Philosophical Transactions of the Royal Society B: Biological Sciences* **365**(1537), 19–30.

Osborne, M. J. & Rubinstein, A. (1994), *A course in game theory*, MIT press.

Peters, H. (2008), *Game theory: A Multi-leveled approach*, Springer Science & Business Media.

Poundstone, W. (2011), *Prisoner's dilemma*, Anchor.

Rehmeyer, J. (2012), 'Game theory suggests current climate negotiations won't avert catastrophe', *Science News Prime* **29**.

Riolo, R. L., Cohen, M. D. & Axelrod, R. (2001), 'Evolution of cooperation without reciprocity', *Nature* **414**(6862), 441–443.

Ross, D. (2014), *Game Theory*, winter 2014 edn.

Samuelson, L. (1998), *Evolutionary games and equilibrium selection*, Vol. 1, Mit Press.

Schnettler, S. (2009), 'A structured overview of 50 years of small-world research', *Social Networks* **31**(3), 165–178.

Shapley, L. S. & Shubik, M. (1954), 'A method for evaluating the distribution of power in a committee system.', *American Political Science Review* **48**(03), 787–792.

Smith, J. M. (1982), *Evolution and the Theory of Games*, Cambridge university press.

Smith, J. M. (1986), 'Evolutionary game theory', *Physica D: Nonlinear Phenomena* **22**(1), 43–49.

Smith, J. M. & Price, G. (1973), 'lhe logic of animal conflict', *Nature* **246**, 15.

Sugden, R. (2001), 'The evolutionary turn in game theory', *Journal of Economic Methodology* **8**(1), 113–130.

Taylor, P. D. & Jonker, L. B. (1978), 'Evolutionary stable strategies and game dynamics', *Mathematical biosciences* **40**(1), 145–156.

Von Neumann, J. & Morgenstern, O. (1944), *Theory of games and economic behavior*, Oxford UP.

Wang, X. F. & Chen, G. (2003), 'Complex networks: small-world, scale-free and beyond', *Circuits and Systems Magazine, IEEE* **3**(1), 6–20.

Appendices

A Code

```
//
//  main.cpp
//  latexfilecpp
//
//  Created by Tara Pesman on 7/2/15.
//  Copyright (c) 2015 Tara Pesman. All rights reserved.
//

#include <iostream>
#include <stdlib.h>      // for rand()
#include <math.h>        // for exp()
#include <random>        // for uniform_real_distribution
using namespace std;


////////////   FUNCTIONS FOR REGULAR GRAPH   ////////////

// Define functions for COORDINATES
// takes the position p of the player (i.e. her number) and returns her x-
    coordinate in the lattice
int functionpx (int p, int s) {
    return p % s;
}
// takes the position p of player (i.e. her number) and returns her y-
    coordinate in the lattice
int functionpy (int p, int s) {
    return (p-(p%s))/(s);
}
// takes the coordinates of a player in the lattice and returns her position p
int functionback (int x, int y, int s) {
    return (x+(y*s));
}

// Define functions for NEIGHBORS
// functions return the number (position) of the neighbor next to the
    considered player (position p)

// neighbor right (i+1)
int functionnr (int p, int s) {
    int x=functionpx(p, s);
```

```
    int y=functionpy(p, s);
    return functionback((x+1) % s, y, s);
}
// neighbor left (i-1)
int functionnl (int p, int s) {
    int x=functionpx(p, s);
    int y=functionpy(p, s);
    return functionback((x+s-1) % s, y, s);
}
// neighbor bottom (i+n)
int functionnb (int p, int s) {
    int x=functionpx(p, s);
    int y=functionpy(p, s);
    return functionback(x, (y+1) % s, s);
}
// neighbor top (i-n)
int functionnt (int p, int s) {
    int x=functionpx(p, s);
    int y=functionpy(p, s);
    return functionback(x, (y+s-1) % s, s);
}

////////////   FUNCTIONS FOR UPDATE RULES   ////////////

// Define function for REPRODUCTIVE RATE (Fi)
float functionFi (int fi, float delta) {
    float Fi = 1 + delta*fi;
    return Fi;
}
// Define function for NORMALIZED REPRODUCTIVE RATE (nFi)
float functionnFi (float Fi, float sumFi) {
    float nFi = (Fi / sumFi);
    return nFi;
}

////////////   MAIN   ////////////

int main(){

    /// ASK USER FOR INFORMATION - SHORTCUT
    // quick entering of variables, example input: 100 hd123 random 0.1 10000
        imitation 20 0.5 no 20 0.5 100
    cout << "Number of players; Which game; Type of graph; Rewiring
        probability; Number of iterations; Update rule; Rate of birth;
        Selection strength delta; Mutation; Rate of overall mutation; Rate of
        individual mutation; Number of datapoints = \n";
    int n;
```

```
    string game;
    string graph;
    float beta;
    int iterations;
    string updaterule;
    int birthrate;
    float delta;
    string mutation;
    int overallmutrate;
    float indmutrate;
    int datapoints;
    cin >> n >> game >> graph >> beta >> iterations >> updaterule >> birthrate
         >> delta >> mutation >> overallmutrate >> indmutrate >> datapoints;

////////////   BEGIN   ////////////
////////////   META   ////////////
////////////   LOOP   ////////////

    // define variable necessary for calculating a meta-average of the
        averages of the proportion of cooperators
    float averagepropc;
    float metatrackc[10][1];

    // initialize metatrack array to zero
    for (int i=0; i<10; ++i) {
        metatrackc[i][0] = 0;
    }

    for (int t=0; t<10; ++t) {

////////////   GRAPHS   ////////////

    /// DEFINE connection array
        // define square n x n connection array
        int con[n][n];

        // initialize connection array to zero
        for (int i=0; i<n; ++i) {
            for (int j=0; j<n; ++j) {
                con[i][j] = 0;
                // print column i in row i to screen
                cout << con[i][j];
            }
            // print an enter after each printed row
            cout << "\n";
        }
```

```
/// GENERATE adjacency matrix for REGULAR GRAPH
    // square lattice with sqrt(n) dimensions
    // part of this code that is written using functions

    if (graph == "regular") {
        // convert the number of players into the dimension of the lattice
             they will play on by taking the square root of n
        int s=sqrt(n);

        // loop through the players and for each player i fill four
            entries (i's neighbors, given by functionnr, -nl, -nb, -nt) in
             the connection array with a 1
        for (int i=0; i<n; ++i) {
            con[i][functionnr(i, s)]=1;
            con[i][functionnl(i, s)]=1;
            con[i][functionnb(i, s)]=1;
            con[i][functionnt(i, s)]=1;
        }
    }

/// GENERATE adjacency matrix for RANDOM GRAPH with average 90 connections
     per player
    if (graph == "random") {
        float conprob = (float) 90/(float)(n-1);

        // set seed for rand() to zero (the numbers that are transformed
            in order to get pseudo-random numbers) and fix number length (
            see notes)
        srand(static_cast<unsigned int>(time(NULL)));

        //loop through the connection array: first row (i=0) left to right
             (j=0 to j=n-1), then next row (i=1 etc.)
        //cout << "\nRandom graph connection array:\n";
        for (int i=0; i<n; ++i) {
            for (int j=0; j<n; ++j) {
                //fill entries to the right of the 0-diagonal with a 1
                    with probability conprob
                if (j>i){
                    // generate random number between 0 and 1 (e.g.
                        randconprob=0.03) to let the considered couple ij
                        be connected with the preferred probability
                        conprob (say 0.05, so 5% of connections to the
                        right of the 0-diagonal will be connected)
                    std::random_device rdconprob;
                    std::mt19937 genconprob(rdconprob());
                    std::uniform_real_distribution<> disconprob(0, 1);
                    float randconprob = disconprob(genconprob);
```

```
                    // only if i has less than 4 connections give it the
                        chance to get connections
                    if (randconprob<conprob) {
                        con[i][j] = 1;
                    }
                }
                if (i==j) {
                    // fill diagonal entries with zeros
                    con[i][j]=0;
                }
                if (j<i){
                    // fill entries to the left of the 0-diagonal with the
                         value assigned to their mirror-entry to the right
                         of the 0-diagonal
                    con[i][j]=con[j][i];
                }
            }
        }
    }


/// GENERATE adjacency matrix for SMALL-WORLD GRAPH
    if (graph == "smallworld") {

    // GENERATE REGULAR RING GRAPH: 4 connections
        // fill connection array (i,j) with 1 if j==i-1 or j==i+1 (i.e. j
            is i's neighbor) and 0 otherwise
        cout << "\nSmall-world connection array before rewiring:\n";
        for (int i=0; i<n; ++i) {
            for (int j=0; j<n; ++j) {
                // connect each player i to the neighbors and next-to-
                    neighbors on either side
                if (j==i-1 || j==i-1 || j==i+1 || j==i+2){
                    //fill entry with 1 (connection) if j is i's (next-to)
                         neighbor
                    con[i][j] = 1;
                }
                // special case: if i is the first player, then i needs to
                     be connected to the last and second-to-last column
                    players
                if (i==0) {
                    con[i][n-1] = 1;
                    con[i][n-2] = 1;
                }
                // special case: if i is the second player, then i needs
                    to be connected to the last and second-to-last column
```

```
                    players
                if (i==1) {
                    con[i][n-1] = 1;
                }
                // special case: if i is the last player, then i needs to
                    be connected to the first and second column players
                if (i==n-1) {
                    con[i][0] = 1;
                    con[i][1] = 1;
                }
                // special case: if i is the second-to-last player, then i
                     needs to be connected to the first column player
                if (i==n-2) {
                    con[i][0] = 1;
                }
                // print column i in row i to screen
                cout << con[i][j];
            }
            // print an enter after each printed row
            cout << "\n";
        }

    // REWIRE CONNECTIONS in regular ring graph
        // set seed for rand() to zero (the numbers that are transformed
            in order to get pseudo-random numbers) and fix number length (
            see notes)
        srand(static_cast<unsigned int>(time(NULL)));

        // loop through the small-world connection array and
            probabilistically rewire some of the connections
        for (int i=0; i<n; ++i) {
            for (int j=0; j<n; ++j) {
                // entries to the right of the 0-diagonal get a chance to
                    be rewired (go from 1 to 0)
                if (j>i){
                    // generate random number between 0 and 1 (e.g.
                        randbeta=0.03) to let the considered connection be
                         rewired with the preferred probability beta (say
                        0.05, so 5% of connections to the right of the 0-
                        diagonal will be rewired)
                    std::random_device rdbeta;
                    std::mt19937 genbeta(rdbeta());
                    std::uniform_real_distribution<> disbeta(0, 1);
                    float randbeta = disbeta(genbeta);

                    // if the randomly generated float is smaller than
                        beta (e.g. 0.03<0.05) then the considered
```

```
                    connection will actually be rewired
                if (randbeta<=beta) {
                    //cout << "\nYes, randbeta<=beta!";
                    // randomly generate some column player k, which
                        will be connected to i instead of player j.
                again:
                    int k;
                    k = rand()%n;
                    // if k is i itself, then generate a new k
                    if (k==i) {
                        goto again;
                    }
                    // if k already has a connection with i, then
                        generate a new k
                    if (con[i][k]==1) {
                        goto again;
                    }
                    // if there are no problems with k, rewire
                    else{
                        // sever the connection between i and j
                        con[i][j] = 0;
                        // make a new connection between i and k
                        con[i][k] = 1;
                    }
                }
            }
        }
    }


    // adjust the entries to the left of the 0-diagonal to the rewired
         connection (so the graph stays undirected)
    cout << "\n\nSmall-world connection array after rewiring:\n";
    for (int i=0; i<n; ++i) {
        for (int j=0; j<n; ++j) {
            // entries to the left of the 0-diagonal will be adjusted
                according to the new value assigned to their mirror-
                entry to the right of the 0-diagonal
            if (j<i){
                con[i][j]=con[j][i];
            }
            // print column i in row i to screen
            cout << con[i][j];
        }
        // print an enter after each printed row
        cout << "\n";
    }
```

```
        }

    /// PRINT connection array as a check
        cout << "\nConnection array for " << graph << " graph\n";
        for (int i=0; i<n; ++i) {
            for (int j=0; j<n; ++j) {
                // print column j in row i to screen
                cout << con[i][j];
            }
            // print an enter after each printed row
            cout << "\n";
        }

////////////   PAYOFFS   ////////////

    /// DEFINE PAYOFF ARRAY
        // define the 2 by 2 pay-off matrix for the Hawk Dove game and fill
            entries with correct payoffs: {{R,S},{T,P}}
        int pay[2][2]; //= {{2,1},{3,0}};
        if (game == "hd123") {
            pay[0][0] = 2;
            pay[0][1] = 1;
            pay[1][0] = 3;
            pay[1][1] = 0;
        }
        if (game == "hd91011") {
            pay[0][0] = 10;
            pay[0][1] = 9;
            pay[1][0] = 11;
            pay[1][1] = 0;
        }

        // print the pay-off array to the screen as a check
        cout << "\n\nPayoff array:\n";
        for (int i=0; i<2; ++i) {
            for (int j=0; j<2; ++j) {
                cout << pay[i][j];
            }
            cout << "\n";
        }

////////////   STRATEGY ARRAY   ////////////

    /// GENERATE STRATEGY ARRAY
        // one-dimensional array to generate a random distribution of the
            available strategies: n players (rows) with one strategy (column)
            each. currently have only two strategies: cooperate=C=0 and defect
```

```
            =D=1.
        int strategy[n][1];

        // print strategy array to screen as check
        cout << "\nGenerated strategy array:\n";

        srand(static_cast<unsigned int>(time(NULL)));


        // fill array using a for-loop through the columns (players)
        for (int i=0; i<n; ++i) {
            // generate 0 or 1 at random
            strategy[i][0]= rand()%2;
            // print strategy array (as one-dimensional column vector) row by
                row (i=0, i=1,...) to screen
            cout << strategy[i][0] << "\n";
        }


////////////    ITERATION LOOP    ////////////

    /// ANALYSIS: necessary definitions for counter inside the loop
        int countcforzero = 0;
        int moduloanalysis = iterations/datapoints;
        int a = 0;
        int countcfori = 0;

        // array to keep track of the cooperators: 6 rows, for each
            measurement moment, 2 columns, for tracking i and the proportion
            of cooperators
        float trackc[datapoints+1][2];

        //initialize to zero
        for (int i=0; i<datapoints+1; ++i){
            for (int j=0; j<2; ++j) {
                trackc[i][j] = 0;
            }
        }

        // loop through strategy-array and count the cooperators (zeros)
            before the iteration-loop
        for (int i=0; i<n; ++i) {
            if (strategy[i][0] == 0) {
                countcforzero += 1;
            }
        }
        // fill the first row of the trackc-array manually
```

```
    trackc[0][0]= 0;
    trackc[0][1]= (float)countcforzero/(float)n;


/// FOR-LOOP FOR NUMBER OF PREFERRED ITERATIONS/EVOLUTIONS/GAMES

    for (int k=0; k<iterations; ++k) {

    // DEFINE AND FILL SCORE ARRAY
        // array to keep the score (1 column) for the players (n rows).
        int score[n][1]; //= {{0}};

        // initialize score array to all zeros
        for (int i = 0; i < n; ++i){
            score[i][0] = 0;
        }

        // print score array to screen as check
        // cout << "\nConnection check:\n";

        // define array to keep a count of the number of opponents that
            each player is connected to
        int countcon[n][1];
        for (int i = 0; i < n; ++i){
            countcon[i][0] = 0;
        }

        // for-loop through the rows (players) in the connection array.
        for (int i=0; i<n; ++i) {
            // for-loop through the columns (possible opponents of
                considered player) of connection array: see whether 0 (don
                't play each other) or 1 (play each other)
            for (int j=0; j<n; ++j) {
                // if entree contains a 1, then add pay-off for player i
                    to his/her entree in the score array entree
                if (con[i][j]==1){
                    // set score for player j equal to previous value +
                        the value in the pay-off strategy for the
                        considered encounter of player j with opponent j.
                    score[i][0] += pay[strategy[i][0]][strategy[j][0]];
                    // count the number of encounters for each player
                    countcon[i][0] += 1;
                }
            }
        }

    // CALCULATE NORMALIZED REPRODUCTIVE RATE
```

```
            // define reproductive-rate array and initialize to all zeros
            float reprate[n][1];
            for (int i = 0; i < n; ++i){
                reprate[i][0] = 0;
            }

            float sumFi = 0;

            // for-loop through the score array: call functionFi to calculate
                the reproductive rate (Fi) for each player (i.e. each row i)
                and print this to reproductive-rate array:
            for (int i=0; i<n; ++i) {
                int fi = score[i][0];
                reprate[i][0] = functionFi(fi, delta);
                // at the end of each loop through a player his calculated
                    reproductive rate (Fi) is added to the variable sumFi,
                    which sums all values of Fi (will be used to normalize Fi
                    --> Pi)
                sumFi += reprate[i][0];
            }

            // define normalized reproductive-rate array and initialize to all
                 zeros
            float norreprate[n][1];
            for (int i = 0; i < n; ++i){
                norreprate[i][0] = 0;
            }

            // for-loop through the reproduction-rate array: call functionnFi
                to normalize the reproductive rate (Fi) for each player (i.e.
                each row i) and print this to normalized reproductive-rate
                array:
            for (int i=0; i<n; ++i) {
                float Fi = reprate[i][0];
                norreprate[i][0] = functionnFi(Fi, sumFi);
            }


////////////// FOR-LOOP FOR UPDATING THE PREFERRED NUMBER OF TIMES //////////////
// within one iteration (still in iteration-loop)

        // BIRTHDEATH UPDATE RULE
            if (updaterule == "birthdeath") {
                // calculate the number of players that will reproduce
                int reproductors = n*((float)birthrate/(float)100);

                for (int i=0; i<reproductors; ++i) {
```

```
            // SELECT PLAYER I that will reproduce
                // generate random number (randi) for selecting player i (
                    rand() is less appropriate for [0,1], see notes)
                std::random_device rdi;
                std::mt19937 geni(rdi());
                std::uniform_real_distribution<> disi(0, 1);
                float randi = disi(geni);
                // define cumulative distribution array for i and
                    initialize to all zeros
                float cumdisi[n][1];
                for (int i = 0; i < n; ++i){
                    cumdisi[i][0] = 0;
                }

                // for-loop to simultaneously generate the cumulative
                    distribution for i (cumdisi) and check its values
                    against randi
                int si = 0;
                // cout << "\nCumulative distribution array for i:\n";
                for (int i=0; i<n; ++i) {
                    // fill cumdis array
                    if (i==0) {
                        cumdisi[i][0] += norreprate[i][0];
                    } else {
                        cumdisi[i][0] = cumdisi[i-1][0] + norreprate[i
                            ][0];
                    }

                    // check just filled entry against random number for i
                         (randi): if equal to or smaller than randi, take
                        that value of i (this is the player that will
                        reproduce) and break out of the loop
                    if (cumdisi[i][0]>=randi) {
                        si = i;
                        // break out of current code and start executing
                            from "selectionidone"
                        goto bdselectionidone;
                    }
                }
            bdselectionidone:

            // SELECT PLAYER J that will be replaced by player i's
                offspring
                // generate random number (randj) for selecting player j,
                    since rand() is less appropriate for (0,1): see notes.
                std::random_device rdj;
                std::mt19937 genj(rdj());
```

```
            std::uniform_real_distribution<> disj(0, 1);
            float randj = disj(genj);

            // define cumulative distribution array for i (vector with
                # of dimensions equal to # of connections for player
                si) and initialize to all zeros
            float cumdisj[countcon[si][0]][1];
            for (int i = 0; i < countcon[si][0]; ++i){
                cumdisj[i][0] = 0;
            }

            // for-loop to simultaneously generate the cumulative
                distribution for j (cumdisj) and check its values
                against randj
            int abssj = 0;

            for (int i=0; i<countcon[si][0]; ++i){
                // fill cumdisj array, e.g. four connections, then
                    fill with 1/4=0.25 ; 2/4=0.50 ; 3/4=0.75 ;
                    4/4=1.00 respectively.
                float relchance = (float)(i+1)/(float)countcon[si][0];
                cumdisj[i][0] += relchance;

                // check just filled entry against random number for j
                    (randj): if equal to or smaller than randj, take
                    that value of j (this is the player that will
                    reproduce) and break out of the loop
                if (cumdisj[i][0]>=randj) {
                    // NOTE: here abssj is the absolute number (e.g.
                        the second of four) of the connection, rather
                        than the number of the column of the player
                        that needs to be replaced
                    abssj = i+1;
                    // break out of current code and start executing
                        from "selectionjdone"
                    goto bdselectionabsjdone;
                }
            }
        bdselectionabsjdone:

            // for-loop horizontally through row si in the connection
                array to find out which column-player is the abssj-th
                connection and will thus be replaced
            int sj = 0;
            int counter = 0;

            for (int i=0; i<n; ++i) {
```

```
                    if (con[si][i] == 1){
                        counter += 1;
                        if (counter == abssj) {
                            sj = i;
                            goto bdselectionjdone;
                        }
                    }
                }
            bdselectionjdone:

            // REPLACE player sj's strategy by player si's strategy in the
                 strategy array
                strategy[sj][0] = strategy[si][0];
            }
        }


    // DEATHBIRTH and IMITATION UPDATE RULES
        if (updaterule == "deathbirth" || updaterule == "imitation") {

            // calculate the number of players that will reproduce
            int reproductors = n*((float)birthrate/(float)100);

            // declare variable sj here so it may be used in any part of
                this if-loop
            int sj=0;

        // SELECT PLAYER J that will be replaced
            for (int i=0; i<reproductors; ++i) {
                // generate random number (randj) for selecting player j (
                    rand() is less appropriate for [0,1])
                std::random_device rdj;
                std::mt19937 genj(rdj());
                std::uniform_real_distribution<> disj(0, 1);
                float randj = disj(genj);
                //cout << "\n\nRandom number generated to select j is " <<
                     randj;

                // define cumulative distribution array for the selection
                    of j out of n players and initialize to all zeros
                float cumdisj[n][1];
                for (int i = 0; i < n; i++){
                    cumdisj[i][0] = 0;
                }

                // for-loop through all players to simultaneously generate
                     the cumulative distribution for j (cumdisj) and check
```

```
                    its values against randj
                for (int j=0; j<n; ++j) {
                    // fill cumdisj array, e.g. four players, then fill
                        with 1/4=0.25 ; 2/4=0.50 ; 3/4=0.75 ; 4/4=1.00
                        respectively.
                    float relchance = (float)(j+1)/(float)(n);
                    // fill cumdis array
                    cumdisj[j][0] += relchance;

                    // check just filled entry against random number for j
                         (randj): if equal to or smaller than randj, take
                        that value of j (this is the player that will
                        reproduce) and break out of the loop
                    if (cumdisj[j][0]>=randj) {
                        sj = j;
                        // break out of current code and start executing
                            from "dbselectionjdone"
                        goto dbselectionjdone;
                    }
                }
            }
        dbselectionjdone:

        // SELECT PLAYER I that will reproduce and replace player j
            // generate random number (randi) for selecting player i,
                since rand() is less appropriate for (0,1): see notes.
            std::random_device rdi;
            std::mt19937 geni(rdi());
            std::uniform_real_distribution<> disi(0, 1);
            float randi = disi(geni);

            // define cumulative distribution variable for the selection
                of i
            float cumdisi=0;

            // define "sumneighbornorreprate": the sum of the norreprates
                of the neighbors of sj
            float sumneighbornorreprate=0;
            // loop through all players in the connection array, but...
            for (int i=0; i<n; ++i) {
                // ...consider only sj's neighbors (1 in the connection
                    array) and sj itself (i==sj)
                if (updaterule == "deathbirth") {
                    if(con[sj][i]==1){
                        sumneighbornorreprate += norreprate[i][0];
                    }
                }
```

```
            if (updaterule == "imitation") {
                if(con[sj][i]==1 || i==sj){
                    sumneighbornorreprate += norreprate[i][0];
                }
            }
        }

        // for-loop to simultaneously fill the cumulative distribution
             for i (cumdisi) and check its values against randi
        int si = 0;

        for (int i=0; i<n; ++i){
            // consider only sj's neighbors (1 in the connection array
                ) and sj itself (i==sj)
            if (updaterule == "deathbirth") {
                if(con[sj][i]==1){
                    // define neighbornorreprate: the relative chance
                        (e.g. 0.25=25%) of each neighbor of j to
                        reproduce and replace j
                    float neighbornorreprate = (float)norreprate[i
                        ][0]/(float)sumneighbornorreprate;

                    // fill cumdisi
                    cumdisi += neighbornorreprate;

                    // check just filled entry against random number
                        for i (randi): if equal to or smaller than
                        randi, take that value of i (this is the
                        player that will reproduce) and break out of
                        the loop
                    if (cumdisi>=randi) {
                        // NOTE: here abssj is the absolute number (e.
                            g. the second of four) of the connection,
                            rather than the number of the column of
                            the player that needs to be replaced
                        si = i;
                        // break out of current code and start
                            executing from "selectionabsidone"
                        goto dbselectionsidone;
                    }
                }
            }

            if (updaterule == "imitation") {
                if(con[sj][i]==1 || i==sj){
                    // define neighbornorreprate: the relative chance
                        (e.g. 0.25=25%) of each neighbor of j to
```

```
                        reproduce and replace j
                    float neighbornorreprate = (float)norreprate[i
                        ][0]/(float)sumneighbornorreprate;

                    // fill cumdisi
                    cumdisi += neighbornorreprate;

                    // check just filled entry against random number
                        for i (randi): if equal to or smaller than
                        randi, take that value of i (this is the
                        player that will reproduce) and break out of
                        the loop
                    if (cumdisi>=randi) {
                        // NOTE: here abssj is the absolute number (e.
                            g. the second of four) of the connection,
                            rather than the number of the column of
                            the player that needs to be replaced
                        si = i;
                        // break out of current code and start
                            executing from "selectionabsidone"
                        goto dbselectionsidone;
                    }
                }
            }
        }
    dbselectionsidone:

    // REPLACE player sj's strategy by player si's strategy in the
        strategy array
        strategy[sj][0] = strategy[si][0];
    }

/// MUTATIONS (still in the iterations for-loop)
    // after the preferred number of updates per iteration, each
        iteration ends with a probabilistic number of mutations

    // set seed for rand() to zero (the numbers that are transformed
        in order to get pseudo-random numbers) and fix number length (
        see notes)
    srand(static_cast<unsigned int>(time(NULL)));

    // define a variable which gives the number of players that will
        possibly mutate (= player m's), say 100*(20/100)=20.
    int numpossmut = (int) n * (float)overallmutrate/(float)100;

    // for-loop that iterates as many times as there are player m's (
        say 20 times), generating random numbers between 0 and n (e.g.
```

```
            23): the number selected is a player m (player 23).
        for (int i=0; i<numpossmut; ++i) {
            // generate random number for overall mutation between 0 and n
                (e.g. random=23) to select player m (possmut=23)
            int random = rand()%n;
            int possmut = random;

            // generate random number for individual mutation between 0
                and 1 (e.g. randim=0.03) to let the selected player mutate
                with the preferred indmutrate (say 0.05, so 5% of player
                m's actually mutates)
            std::random_device rdim;
            std::mt19937 genim(rdim());
            std::uniform_real_distribution<> disim(0, 1);
            float randim = disim(genim);
            // if the randomly generated float is smaller than indmuterate
                (e.g. 0.03<0.05) then the given player m will actually
                mutate (that is, flip strategies)
            if (randim<indmutrate) {
                if (strategy[possmut][0] == 1){
                    strategy[possmut][0]=0;
                }
                if (strategy[possmut][0] == 0) {
                    strategy[possmut][0]=1;
                }
            }
        }

    // COUNT cooperators in the strategy array for every i that is one
        fifth of the total number of iterations (e.g. iterations=100, then
         for k = 19 [which is the 20th iteration], 39 etc.)
        if ((k+1) % moduloanalysis == 0) {
            // counter a
            a += 1;
            // for-loop through the strategy array.
            countcfori =0;
            for (int i=0; i<n; ++i) {
                if (strategy[i][0]==0){
                    countcfori += 1;
                }
            }
            // fill the trackc-array
            trackc[a][0]= k+1;
            trackc[a][1]= (float)countcfori/(float)n;
        }

        // end iteration-loop
```

```
        }

    // PRINT final STRATEGY array to screen as check
        cout << "\n\nFinal strategy array (including updates and mutations):\n
            ";
        for (int i=0; i<n; ++i) {
            // print strategy array (as one-dimensional column vector) row by
                row (i=0, i=1,...) to screen
            cout << strategy[i][0] << "\n";
        }

    // PRINT final TRACKC array to screen as check
        cout << "\n\nTrackc array:\n";
        for (int i=0; i<datapoints+1; ++i) {
            for (int j=0; j<2; ++j) {
                cout << trackc[i][j] << "  ";
            }
            cout << "\n";
        }

    // CALCULATE average proportion of cooperators from the data in the trackc
         array
        float sumpropc=0;
        averagepropc=0;

        for (int i=0; i<datapoints+1; ++i) {
            sumpropc += trackc[i][1];
        }
        averagepropc = (float)sumpropc / (float)(datapoints+1);
        // print average proportion of cooperators for this t-th run of the
            program to the metatrackc array
        metatrackc[t][0]=averagepropc;

    }
////////////    END    ////////////
////////////    META    ////////////
////////////    LOOP    ////////////


// CALCULATE meta-average and PRINT metatrackc array to screen
    float metasumpropc=0;
    float metaaveragepropc=0;
    cout << "\n\nMetatrackc array:\n";
    for (int i=0; i<10; ++i) {
        metasumpropc += metatrackc[i][0];
        cout << metatrackc[i][0] << "  ";
        cout << "\n";
```

```
    }
    metaaveragepropc = (float)metasumpropc / (float)(10);
    // print meta-average proportion of cooperators for all 10 runs of the
        program to the screen
    cout << "\n\nMeta-average = " << metaaveragepropc << "\n\n";


    return 0;
}
```