Dissertation

Cesar Esparza

August 24, 2015

Introduction

In nature every living creature has specific characteristics that determine their role in their environments. Each living creature from microscopic entities to a blue whale or the tallest tree, posses patterns that allow them to live, some longer than other. We perceive many similarities among living, born at certain point, grow, reproduce and die, that is the path for all. But as similarities exist also differences, living entities in cold environments have different obvious characteristics to others in a hotter environment. But yet despite of those different physical differences that exist I dare to say the most interesting is behaviour. It defines role every existing element has, hunter and prey is the first idea that comes to my mind when talking about roles. But we can go deeper in the nature of these to an we can see that the behaviour is different, but a prey can also be a hunter. That is the essence of behaviour how particular it is to each type of organism although the same goal is always the intention, survival. As humans we might be one of the most complex creatures in our environment. For some basic needs like eating to survive is not something to be concerned about. And in my mind this is one of the reasons our behaviours have changed, one can say they have evolved, I like to call it adapting. The basic rules of survival have changed for humans, a big percentage of the humans do not hunt to eat anymore, some do farming and act as providers for the others. And the key word for me is rules, rules dictate who we are and how we behave around others. It is bold to say that we all play a game, life, and the rules are not only dictated by our natural environment but also by the social environment. Something I find interesting is asking the question, why do we play for?

In this work the intention is not to follow the existential line of previous paragraph, but to give a general framework of a particular branch of game theory and building a program in python to solve simple problems presented in the normal form. The program built in Python will use a simplified genetic algorithm. And the approach of the model is called Agent based modelling, which will be introduced in another chapter. Basic relevant knowledge about evolutionary game theory will be presented. And the Python program will be used to solve some well known normal form games (games is strategic form) like prisoners' dilemma, rock-paper-scissors, matching pennies, and some others. We will also use the program to simulate a an experiment made by the professor in political science Robert Axelrod in 1980, the objective was to see how cooperation evolved in an environment where selfish behaviours exist when interacting through time. The experiment consisted in a tournament where several well-known game theorists participated by providing a strategy that was set to interact with strategies from others, and then determine which strategy was more successful.

Abstract

The literature review can be found in the following section. Some history of Game theory will be mentioned, Nash equilibrium is briefly mentioned but still important since the concept of equilibrium is important to discuss in game theory, simple concept about normal form games, history of evolutionary game theory and an explanation of what evolutionary game theory is and the introduction of the concept of equilibrium in evolutionary game theory which is known as evolutionary stable strategies(ESS), and a brief explanation of the similarities with Nash equilibrium.

1 Game Theory

Game theory or theory of games is a wildly known theory, which studies the interaction of decisions. This interaction is given to the interdependence between the participants in an environment. Resulting from this there can be mainly two types of interaction cooperative or competitive (Watson 2013)[1]. The whole concept of game theory has a very wide application in many different sciences like economics, companies in a competing market, setting import and exporting tariffs, agreements in wages between employers and employees, auctions, to name some(Gibbons 1992)[2], political sciences Nolan McCarthy mentions examples such as raising more money than other candidates when running for a political position and what to do when the opposition is considered weak or strong, accepting new policies if the outcomes are not certain, an example when a jury has to sentence a defendant among others(McCarty et al 2007)[3], in psychology when studying the behavior of individuals with respect of others, although application of strict assumptions of game theory such as rationality been assumed in the same level for all participants still is a controversial topic for the applying game theory in many areas of social psychology (Rapoport 1999)[4] (Colman 2003)[5] , biology which will be mentioned further in relation with evolutionary game theory, and other disciplines. Studies that started exploring the potential solution of some games can be traced back to 1713 when James Waldegrave in a letter communicated a mixed solution for a two person game to his colleague Pierre Remond de Montmort (Hykov 2004)[6] . Since Waldegrave many others did studies that now we relate to game theory (Watson 2013)[1]. We will start from the french mathematician Emile Borel who in a note in his work "La théorie du jeu et les équations, intégrales à noyau symétrique gauche" (Borel 1921)[7] mentioned:

"The problems of probability and analysis suggest themselves concerning the art of war, or economic or financial speculations, are not without analogy with problems concerning games, though they generally have a higher degree of complication".

In 1924 Borel on his note "On games that involve chance and the skill of the Players" (Borel 1953)[8] he mentions that "the study of games that involve at once chance and the skill of the players appears to me similarly able to furnish an opportunity for mathematical research, the applications of which might far surpass the limits of the restricted domain to which this first study is limited. /such research might be extended to very many questions in which psychological unknowns figure along with algebraic unknowns." Then Borel continues saying that the only author who had studied problems with this focus was Joseph Bertrand in his "Calcul des Probalites" in 1889, distinguishing between mathematical and psychological aspects in an example given of a game of baccarat, but goes on stating why Bertrand study was incomplete (Borel 1953)[8].

Game Theory was further researched and formally presented by the Hungarian mathematician John von Neumann in 1928 with his work Theory of Parlor Games, stating in the introduction that " any event given external conditions and the participants situation (provided the latter are acting of their own free will) may be regarded as a game of strategy if one looks at the effect it has on the participants. (von Neumann 1928)[9]. And in 1944 John von Neumann and Oskar Morgestern published Theory of Games and Economic Behavior in which they established theory of games of strategy as an instrument to study problems in the economic behavior (Neumann, Morgestern 1944)[10]. Despite of the great contribution of von Neumann and Morgestern, game theory became widely used after the Doctoral dissertation from John F. Nash was published in 1950. The some of the main arguments from Nashs work are that he gives the possibility of analyzing games with more n-players, he introduces the concept of non-cooperative games having at least one equilibrium point and gives the idea of a dynamical approach to study cooperative games (Nash 1950)[11]. The last point I mentioned refers to how a cooperative game can be reduced to non-cooperative form, because the idea of a cooperative game established by von Neumann and Morgenstern was that cooperation was given under the assumption described in Nashs words players can communicate and form coalitions which will be enforced by and umpire.(Nash 1950)[11] , but he proves this condition is not the only one that would define a cooperative game. The work from Nash opened the possibility of a broader application of the essential concept of game theory, many people have extended the fundamental concepts adopted from Nashs work and a lot of studies

for the development of game theory have been made.

1.1 Normal form games

(Watson 2013)[1] mentions that there are several mathematical ways to describe games. In game theory there are two most common ways to represent a game called extensive form and normal form. The extensive form which represent in a form of a game tree the different actions that can be taken by each player. Starts with an initial node from which it branches out representing the possible choices and then after each branch another node is placed where a second player has other choices that branch out, until reaching the end of the tree where the payoff for each sequence(following the branches) for the path are represented. But for the purpose of this work, the normal form representation of a game will be used and the basic information about it is the following. The normal form games, also known as strategic form of games, usually have the following elements:

- **Players** A finite set of N players.
- **Strategies:** A set S_i for each player $i \in N$. Which as we have discussed represents the action a player chooses.
- **Payoffs:** Payoff functions represented for each player represented by u_i*:* $S_1\ x\ S_2\ x\ ...\quad x\ S_N \rightarrow \mathbb{R}$ .Which represent the payoff each player obtain after interacting with each other.

For the purpose of this work, the number of players will be $N = 2$ taking the following assumptions from (Knight ????) we will represent the game with a **bi-matrix**. And we will assume that $\mathbf{S_1 = \{s_i \mid 1 \geq i \geq n\}}$ and $\mathbf{S_2 = \{s_j \mid 1 \geq j \geq n\}}$ this will be the **bi-matrix** for this game :

Player 2

Player 1		s_1	s_2	...	s_j
	r_1	$u_1(r_1,s_1)$, $u_2(r_1,s_1)$	$u_1(r_1,s_2)$, $u_2(r_1,s_2)$	...	$u_1(r_1,s_j)$, $u_2(r_1,s_j$
	r_2	$u_1(r_2,s_1)$, $u_2(r_2,s_1)$	$u_1(r_2,s_2)$, $u_2(r_2,s_2)$	...	$u_1(r_2,s_j)$, $u_2(r_2,s_j$
	.	.	.	...	.
	.	.	.	...	.
	.	.	.	...	.
	r_i	$u_1(r_i,s_j)$, $u_2(r_i,s_j)$	$u_1(r_i,s_j)$, $u_2(r_i,s_j)$	...	$u_1(r_i,s_j)$, $u_2(r_i,s_j)$

Table 1.1: Normal form game bi-matrix

We can see that each intersection of row and column have two utility functions. The first utility function represents the payoff for the row player and the second utility function represents the payoff for the column player. Utilities can be of type ordinal, which is used only to rank the alternatives from better to worse, or cardinal which indicates that the value assigned is meaningful and represents the satisfaction of the player (McCarty et al 2007)[3].

1.2 Nash Equilibrium

John Nash writes in his dissertation "... an equilibrium point is an n-tuple s so that each player's mixed strategy maximizes his payoff if the strategies of the others are held fixed. Thus each player's strategy is optimal against those of the others." (Nash 1950)[11]. After this he describes under what circumstances a mixed strategy behaves as a pure strategy. Nowadays the definition of Nash's equilibrium point is known as a Nash equilibrium and we will commonly find the phrase "no regrets" when describing the Nash equilibrium and this is because paraphrasing Nash, no player in the game could have had a better payoff given the

action chosen by her opponent. For an N player normal form game Nash equilibrium Knight 20?? gives the following definition:

$$u_i(\hat{s}) \geq u_i(\bar{s}_i, \wedge s_{-i}) \forall i \tag{1}$$

In a two player game, this means each player i has a set of strategies S, a pair of strategies $(\hat{r}, \hat{s})$ we can define the pure Nash equilibrium as:

$$u_1(\hat{r}, \hat{s}) \geq u_1(r, \hat{s}) \forall r \in S_1 \text{ and } u_2(\hat{r}, \hat{s}) \geq u_2(\hat{r}, s) \forall r \in S_1 \tag{2}$$

An "easy" way to find Nash equilibria in games with pure Nash equilibria is by looking at the table in the normal form that holds the strategies and payoffs for each player, and underlining the best response for each player when the other player's strategies are held fixed. As an example we will use the prisoner's dilemma normal form, and first we hold fixed player's 2 strategies and we underline the best response for player 1 for each of player's 2 strategies.

Player 2

		Cooperate	Defect
Player 1	Cooperate	3, 3	0, 5
	Defect	$\underline{5}$, 0	$\underline{1}$, 1

Figure 1.1: Best responses for player 1

In this first table we have that defect is the best response for player 1 in the case where player 2 chooses either of the strategies. Now we explore the options holding player 1's strategies fixed and we have the following.

Player 2

		Cooperate	Defect
Player 1	Cooperate	3, 3	0, $\underline{5}$
	Defect	5, 0	1, $\underline{1}$

Figure 1.2: Best responses for player 2

And now we only take the strategies that are best responses for both players. Remembering that is the strategy where neither of the players have any reason to change from given that whatever the other player choice is it will always give them the best payoff available for the interaction.

Player 2

		Cooperate	Defect
Player 1	Cooperate	3, 3	0, 5
	Defect	5, 0	$\underline{1}$, $\underline{1}$

Figure 1.3: Nash equilibrium with best responses.

1.3 Evolutionary game theory

Evolutionary game theory has its roots in evolutionary biology. Even if this overview is not intended to be focused in biology some remarks from it are worth mentioning. From the 6th edition of Origin of species Charles Darwin outlines that in nature there exist many struggles for existence. Some examples are the struggles of a species in the nature, between species and within species. Darwin stresses that the most severe struggle might be within species if they become into competition (Darwin 1872)[12]. In origin of species with the idea of all possible interactions of the 'organic beings mentions that there may be infinite varied diversities of structure for each being under the changing conditions of life, and continues saying that through the course of generations, there can occur variations that can give perhaps a slightly advantage

to some beings giving them a higher chance of survival and procreation. The preservation of favourable characteristics and destruction of what he called injurious, he called it natural selection or survival of the fittest (Darwin 1872)[12]. I go this far because later on we can think of the concept of fitness, as the utility a player has in evolutionary game theory which in general terms can be called the fittest.

Fisher might have been the first to apply game theory to evolution in his study on sex ratios in 1930 (Pallen 2009)[13] although at this time the formal definition of game theory had not been presented yet. Later in 1961 Lewontin discussed species playing against nature, and said that species should adopt the "maximin" strategy if nature presented worse-case scenarios (Maynard 1974, Sigmund 2004)[14]. In 1967 in an the article Extraordinary sex ratios wrote in Science William D. Hamilton uses game theory to model local competition and frequency-dependent fitness values (Sigmund 2004)[14]. Perhaps the most important contribution in evolutionary game theory was the one made by Professor John Maynard and Dr. George Price. John Maynards attention was first caught by the article "Antlers, Intraspecific Combat, and Altruism',' written by George Price in 1968 for 'Nature about ritualized behavior in animal contests (Maynard 1976)[15] unpublished for being too long. Then in 1973 John Maynard and George Price published a joint paper On the logic of animal conflict in which the mathematical concept of an evolutionary stable strategy is established, and it applies concepts of game theory to the study of conflicts between animals (Sigmund 2004)[14]. The idea Maynard and Price presented was that concepts from game theory could be used to characterize eventually stable endpoints in the evolutionary process, with the concept of evolutionary stable strategy (McNamara 2010)[16]. It can be said that the concept of evolutionary game theory was born with the ideas of John Maynard and George Price.

Nowadays evolution by natural selection can be thought of as a game, where some behavioural patterns (often referred to as phenotypes the equivalent to strategies when related to traditional game theory) from animals are more successful than others (Carmichael 2005)[17]. Now I can relate evolutionary game theory with traditional game theory in the some essential concepts. Animals in the biological concept are equivalent to the players (agents) that participate in a game; the environment in which animals interact is comparable to the set of rules that regulate interaction in the traditional form; as mentioned before the heritable phenotypes of animals can be thought of as the strategies that players use in the traditional form; a tricky concept to relate to evolutionary game theory is the one of payoffs (utility) in traditional game theory for this I will refer to how Maynard and Price define it in The logic of animal conflict as the contribution the contest has made to the reproductive success of the agent (Maynard & Price 1973) which could be the expressed in terms of fitness (Darwin 1872)[12] the fitness in an agent directly influences the frequency of the strategy in the population (Vincent 2005)[18], Maynard and Price take in account three factors to be taken in account: the advantages of winning compared to losing, disadvantage of being seriously injured and disadvantage of wasting time and energy in the contest (Maynard & Price 1973)[19] this are not usually considered in games but I consider important mentioning. Another very important concept is equilibrium, in some evolutionary games the existence of evolutionary stable strategies (ESS) , I will not yet talk about the mathematical implications of ESS. Roughly we can consider an ESS as a strategy that predominates in frequency in evolutionary games and that in the case of the emergence of a mutated strategy is not invaded (threatened to be reduce in number) this concept of ESS has similarities with the concept of Nash equilibria as seen by in the sense that both can be "no-regret" strategies when in a population a Nash or an ESS is played "no individual can benefit from unilaterally changing their strategy" (Vincent 2005)[18].

Thinking of how the idea of evolutionary game theory was conceived from behavior of animals, comparing it to the traditional game theory, there are some characteristics that distinguish them from each other. First and perhaps one of the most relevant assumptions in traditional game theory is that every player is rational which means they make rational decisions to maximize their profits, and also they are aware of the possible payoffs of the other players and that other players are rational, also the rational players are aware of the game rules, evolutionary game theory does not make such assumption of rational players, instead the strategies are hard wired to them. Traditional game theory it is about choosing from different strategies looking to optimize the payoffs, whilst evolutionary game theory is to determine strategies that will endure through time. Traditional game theory as said before has a set of strategies from which it can choose, whilst

in evolutionary game theory the strategies are already defined given that they are inherited although there can be present some occasional mutations. Also evolutionary game theory there will be groups of players that possess the same set of strategies and the same related payoff from these strategies, in traditional game theory each player has their own set of strategies and their own associated payoffs per strategy (Vincent 2005)[18]. The application of evolutionary game theory in different areas of study has grown, and with this some assumptions change. For example in the biological application players do not choose their strategies and never change them, unlike in the economic application the players are people, who can choose and change their strategies (Samuelson 1997)[20].

What is evolutionary game theory? It can be defined as the combination of some game theoretical concepts, with the concepts of natural selection in evolution. It is a change in the focus from traditional game theory, because the main goal of evolutionary game theory is to observe the stable equilibria and how it changes through time with the interactions between the organisms (players) different behaviours (strategies), instead of only focusing in optimizing outcomes for a single game. Something important to note about the interaction is that in evolutionary biology and evolutionary game theory, the concerning interaction is between individuals of the same species. In this sense we can identify two main approaches to evolutionary game theory (McKenzie 2009)[21]. The first approach is the static approach which is directly derived from the work of Maynard and Price, the main tool for analyzing is the ESS. The second approach through the study of the population dynamics (change in density of existing strategies) and of how the strategies evolve in the model built (McKenzie 2009)[21].

1.4 Evolutionary Stable Strategy (ESS)

To explain better the concept a symmetric game will be analysed. It is important to mention that the games that we will focus on are two player games. When describing ESS the most frequent example used is Hawk-Dove, first used by Maynard and Price in their paper "The logic of animal conflict" in 1973 which has been mentioned before. It is worth mentioning that this game has a structure of the well-known prisoners dilemma game. 'Prisoners dilemma formally presented by Albert Tucker to psychology students in Stanford University in 1950. As we may know prisoners dilemma is a game in which two individuals who committed a crime are interrogated in separate rooms, and they are not able to exchange information with each other. So they are presented with 2 options (strategies), they can confess or not. And the interaction between the choices each one have go as follow. One confesses and the other does not, both of them confess or neither confess. Each interaction has a pay-off. From this point on I will refer to the players of the games we describe as agents. First a general definition of the ESS. To illustrate this it will be used a similar approach to the one by Easley & Kleinberg in "Networks, Crowds and Markets: Reasoning about a Highly connected World" We know that a strategy is evolutionarily stable when a population of agents can resist the invasion of emergent mutated agents (new strategy). As mentioned before in evolutionary biology as well as in evolutionary game theory, fitness can be defined as reproductive success (Kleinber 2010)[22]. Therefore agents with higher fitness value are majority in a population or in time will become majority, whereas agents will low fitness value will be minority and in time with a very high probability will disappear. The fitness value is obtained using the pay-off values from each interaction. With the following table it will be presented a clearer explanation of this. For a symmetric strategic two-player game we have the following bimatrix.

		Agent 2	
		X	Y
Agent 1	X	a, a	b, c
	Y	c, b	d, d

Figure 1.4: General Symmetric Game

We assume that there exists a large population of agents that always take action X. Now we suppose that within the population appears a small group that take action Y and the fraction representing the number

of agents in this group is ϵ. Since ϵ represents the number of agents that choose Y, we can say that 1 - ϵ is the fraction of the agents that choose X. We will assume then that the probability of an agent using X encountering another agent at uses X is 1 - ϵ, and with a mutated that uses Y the probability is ϵ. With these and the values from the figure 1.2 we build the payoff equation for the agent that use strategy X as follows:

$$(1-\epsilon)a + \epsilon b \tag{3}$$

And with the same values we build the pay-off equation for the agents using the strategy Y as follows:

$$(1-\epsilon)c + \epsilon d \tag{4}$$

For X to be an evolutionary stable strategy, we need:

$$(1-\epsilon)a + \epsilon b > (1-\epsilon)c + \epsilon d \tag{5}$$

It should be easy to see that for X to be an ESS a c for small values of ϵ, on the other hand when the values of ϵ are closer to 1, for X to be ESS a = c and b d. Since the pay-offs for each agents are the result from the interactions, we should understand the following.

- For X to be an ESS, the pay-off for interacting with X should be greater or at least equal to the pay-off that an Y gets when interacting with X.
- Or for X to be an ESS when X and Y get the same pay-off while interacting with X, X needs to get a better pay-off when interacting with Y than Y interacting with Y.

We see that if the previous does not hold, then $(1-\epsilon)a + \epsilon b < (1-\epsilon)c + \epsilon d$ and therefore X is not an ESS.

1.4.1 Evolutionary stable strategy and Nash equilibrium

To gain a better understanding of the concepts linking ESS and Nash equilibrium, let us remember what Nash equilibrium is. An action (strategy) profile s^*, where no player i can do better by choosing a different action (strategy) from s_i^*, holding players j action (strategy) s_j^* fixed (Osborne 2004)[23]. Including the notions of rationality we already know. Now a similar explanation to the one given in Networks, Crowds, and Markets: Reasoning about a Highly Connected World by Easley and Kleinber[22]. Looking back at table ???, we can take (X, X) as a Nash equilibrium, therefore we assume X is a best response to both players. And this would mean the following according to the arbitrary values we assumed:

$$a \geq c$$

The conditions for a strategy to be evolutionarily stable are the following:

$$a > c, \text{ or } a = c \text{ and } b > d$$

If $a > c$ then X would be a strict Nash equilibrium, on the other hand if $a = c$ *and* $b > d$ the condition is not a strict but still a Nash equilibrium. Also we can draw the following conclusions:

- *If X (any strategy s) is not a Nash equilibrium, then X is not evolutionarily stable.*
- *If X (any strategy s) is evolutionarily stable, then X is a Nash equilibrium.*
- *If X (any strategy s) is a Nash equilibrium, but $a = c$ and $b < d$ then X is not evolutionarily stable.*

Hoping to provide a more understandable explanation of the previous, an example with values will be used and since evolutionary game theory is originated from evolutionary biology, as many literature that defines ESS, we will use an example where the players are animals of a certain type.

Let us assume that there is a population of wolves all of the same size, but at some point in time a

mutation occurs and some bigger wolves result from this. This mutation now represents the fraction ϵ of the population, and the rest of the population (small size wolves) are represented by 1 - ϵ. These species of wolves as many others hunt in packs and given this cooperative nature the small size wolves share the prey in exactly half. Nevertheless the big sized wolves cannot afford to be so equally sharing, they need a bigger share of the prey since their body needs more nutrients, also given their size they have to make a greater effort when catching the prey. These two issues have made the big wolves aggressive to the other wolves whenever they compete for a share of the prey. The small size wolf avoids conflicts with other wolves, so if a big wolf attacks him after catching the prey they leave, but even with this situation most of the times the small size wolf still has a small share of the prey. A different scenario is given when two big wolves hunt a prey, they will fight each other until they are left seriously injured. This situation is costly for both. The expected pay-offs of these interactions are represented in the following table:

Wolf size

	Small	Big
Small	5, 5	1, 7
Big	7, 1	2, 2

Figure 1.5: Wolves Hunting

Now we will determine if the small size wolf is an evolutionarily stable strategy, for this we will use the pay-off formulas previously defined, and we have that in this population of wolves, the small size have a pay-off of:

$$(1-\epsilon)5 + 1\epsilon = 5 - 4\epsilon$$

While the population of bigger wolves have the following expected pay-off:

$$(1-\epsilon)7 + 2\epsilon = 7 - 5\epsilon$$

For small values of ϵ we see that the expected pay-off of the big wolves is higher than the small wolves pay-off. This means that small wolves are not evolutionarily stable. We can also determine if the big wolves are evolutionarily stable. We now assume that in a population of big wolves, some mutation of small wolves appear in the population in a fraction equal to ϵ. Therefore the population of big wolves is now 1 - ϵ. Now we determine the expected pay-off for big size wolves in this kind of population:

$$(1-\epsilon)2 + 7\epsilon = 2 + 5\epsilon$$

And in this population, the mutated small sized wolves will have an expected pay-off of:

$$(1-\epsilon)1 + 5\epsilon = 1 + 4\epsilon$$

In this population, we can see that the expected pay-off of the big sized wolves is greater than the one for small size wolves. This means that the big size wolf population is evolutionarily stable and a strict Nash Equilibrium.

1.5 Agent-Based Modelling (ABM) and object oriented programming (OOP)

Generally speaking agents are created entities that represent entities of the real world, and make them interact to study their behavior. One of the first known scientists to show interest in the concept of entities was John von Neumann (Wilensky, Rand 2015)[24]. He thought in the future it would be useful to have artificial machines that could reproduce autonomously, in order to represent objects such as celestial objects (Wilensky, Rand 2015)[24]. Given to the suggestion of his colleague Stanislaw Ulam, von Neumann developed a simple model of cellular automaton. In which each cell take one of multiple states, and then the changes in it are based on the history of previous states from the cell and neighboring cells (Janssen, Ostrom 2006)[25].

In 1970 Martin Gardner published a game by the british mathematician John Conway. Which was a simplified model of cellular automata applied into what he called Game of Life (Janssen, Ostrom 2006)[25]. In which an infinite universe is divided into cells, each cell interact with the neighboring cells (8 cells around it), according to a set of established rules creating complicated patterns according to simple states of the cells, which are alive or dead (Rendell 2001)[26]. In 1973 in a joint paper published in Nature (Nature Publishing Group) by Professor Maynard Smith and George R. Price describe the results of simulations made, which can be considered an example of an agent-based model (Maynard & Price 1973)[27].

In 1971, 1978 the economist Thomas Schelling used a chessboard in which by moving pennies and dimes he represented what he described in his work "Models of segregation" (Janssen, Ostrom 2006)[25]. Finally another important work using agents were the simulations made by the political scientist Robert Axelrod in 1984 (Janssen, Ostrom 2006)[25], in which he asked game theorists from different disciplines to submit a strategy which was used in the simulation of a tournament of one of the types of games called 'prisoners dilemma'. Each strategy interacted with the other strategies, during a number of rounds (Axelrod 1984))[28].

The previously mentioned works are some of the most relevant that involve this agent based modeling. Agent-based models (ABM) are being increasingly used to model complex systems.(Billari, et al 2006))[**?**]. Robert Axelrod in his book 'The complexity of Cooperation defines ABM as the simulation of agents and their interactions. This type of modeling is different from the traditional type, and it is a type of simulation that is viewed as 'bottom-up (Axelrod 1997, Billari 2006, Bonabeau 1994, Gilbert 1999)[29] used for understanding properties of social systems (Axelrod 1997)[29], by representing complex behaviours in the hopes of emulating some specific aspect. There are 3 ways for doing science according to Axelrod, which are induction, deduction and agent-based modeling, the latter starts like deduction with a set of explicit assumptions then it generates simulated data that can be analized inductively. And unlike induction the data produced comes from specific rules rather than real world data (Axelrod 1997)[29]. Joshua Epstein points ABM as "a new tool for empirical research" (Epstein, 2006)[30]. So it is very important to remember that our final goal with ABM is not to recreate reality, its rather for demonstrating a principal and understand how it works. There are at least three main components to take in account when building an ABM, the number of agents with characteristic variables (agents contain data together with methods which act of the data), a set of rules and the environment in which the interaction takes place (Bruun 2004)[31] this will be described in further detail in the description of the simulation. In agent based modeling what matters is not how decisions are made but how the interaction is given (Bruun 2004)[31].

A very important step when building a simulation is to choose a programming language. Axelrod mentions that a good agent based model should achieve three goals: validity this is for the program to implement the model correctly, usability which involves the fact that the person building the model and other users can run the program, interpret the output and understand how it works, and the third point is extendability which means the program should be adaptable for new uses in the future (Axelrod 1997)[29] . For the type of simulation that will be built, object oriented programming (OOP) is appropriate. In the recent years this language has gained popularity, even that it can be traced for a long way back. Before the concept of OOP was well established, all computing languages were procedural languages, which means that it looked like he program was all contained in one long procedure all data and the logic were presented in a long code.

OOP's concern is on 'objects' (Pokkunuri 1989)[32] this objects can be anything one can think of i.e. an action, a thing, a person, a place, of course they have to be well defined. Object is the basic element. Objects possess attributes of procedures and data. Storing the data in variables and responds to messages by executing procedures (in Python called methods). The program is divided in individual objects (modules) each one can be viewed as an abstract data type. Each of the objects contain their own methods and data (Pokkunuri 1989)[32]. Communication between objects requesting action can be called message. The purpose of this is that each object represent parts of whole program, but by breaking it down in modules each can be thought of as a particular action which we can relate in an easier how ideas and actions are structured in our everyday

life. As an example we can think of an apple which would be our object and this apple contains attributes such as color, size, weight, etc. but it can also contain methods such as growing, changing color, falling from a tree, etc. This way of conceptualizing ideas in such conventional way in relation to our everyday life make it a perfect candidate for using in simulations. In this project OOP would simplify structuring the ABM, in which it was mentioned before we have so very defined concepts (agents, rules and environments).

To be able to work with OOP there is one idea we need to understand properly, which is the difference between class and object. A class can be thought of as a general description or blueprints of something but is not the thing itself. And what the class is defining are abstract ideas of what was mentioned before, 'attributes and 'methods, formally a class is defined as a template from which objects are created (Dyke 1989)[33]. The next concept is object, would be the "tangible" instance of the initial template which is the class(Luna 2012)[34]. To understand this correctly we can think of the class as the blueprints of something we want to build, and the object is that thing we wanted to build from the blueprints. And when creating an object the abstract ideas from the class, become specific characteristics of the object. A relevant property is inheritance, which allows a class to inherit methods and attributes from another class, this makes the class that inherits a subclass of the class it inherits from, its important to mention that this subclass can redefine inherited methods and can add methods that can differentiate it from the class it inherited from (Dyke 1989)[33]. Some other relevant properties from OOP are dynamic binding, which is not exclusive for OOP, means that the binding of operator to a particular operation takes place at the run time. Encapsulation is another and it describes the scope of unrestricted reference to the attributes of an object. An object can examine and modify its own attributes, and allows access to its attributes to other objects through accessing functions allowing it to have control over any changes requested from other variables. Data abstraction refers to how any object can be required for any information, and the fact that who requests gets what he/she asked for (Pokkunuri 1989)[32].

With the created code it will be noticed that individual agents with individual characteristics are created, and they are set to interact in a certain way under determined rules, the final goal will be to have a general perspective of how the interaction of these individual objects leads to a resulting conclusion. In this case we will be able to observe how agents represent different strategies and what is the result of the interaction which will be how one strategy dominates another or how they interact and find a balance. The language chosen for writing the code is Python which is a high level programming language, designed by Guido van Rossum. It serves general programming purposes and it is a very effective object oriented language.

2 Genetic algorithm relating to evolutionary game theory

Algorithms inspired in evolution for optimization and machine learning, have been the subject of interest for many. After the second world war von Neumann showed great interest in artificial intelligence, his main studies were focused and can be said that originated what now known as cellular automata(Boden 2006) [35]. The mathematician Nils Barricelli wrote simulations of evolutionary processes, in the interest of creating artificial life in computers using nature processes. He beccame the first to write a genetic algorithm software, and his first work about it was published in 1954 (Simon 2013)[36]. Another person who would use computer programs to simulate evolution was the biologist Alexander Fraser. Fraser was interested in evolution, but in the real world evolution happened to slowly for this reason he decided to use simulations created in digital computers. Fraser wrote in 1957 "Simulation of genetic systems by automatic digital computers" and with these he became the first person to use computer simulations with the purpose of simulation biological evolution (Simon 2013)[36]. Other scientists also used computers to simulate evolutionary processes like Han-Joachim Bremermann his first paper about it was a published in 1958 entitled " The evolution of intelligence". George Box, was another scientist who worked in the field. Interesting about Box is that his main interest was not artificial life or evolution, his interest was more in solving real life problems. The first paper he published that can be related to genetic algorithm was in 1957 and the title was "Evolutionary operation:

A method for increasing industrial productivity"(Simon 2013)[36]. Box states that living things advance in means of two mechanisms: 1. Genetic variability and 2. Natural selection (Goldberg 1989)[37], and in raw terms he associates a new discovery of a process to a mutation and using or discarding different less efficient processes as natural selection. George Friedman outlined in 1956 how a "selective feedback computer" might evolve electrical circuits to maintain internal temperature of a mobile robot (Boden 2006) [35], this work being similar to nowadays genetic algorithm (Simon 2013)[36].In 1966 Lawrence Fogel along with Alvin Owens and Michael Walsh wrote a book about genetic algorithms with the title "Artificial Intelligence through Simulated evolution" (Simon 2013)[36]. In 1960s and 1970s James Holland with students and colleagues from University of Michigan developed the genetic algorithm (Melanie 1999)[38]. Holland's goal was to study the phenomenon of adaptation of processes in natural systems and develop ways in which the mechanics of adaptation might be imported to computer systems. Holland introduced a population-based algorithm with crossover, inversion and mutation (Melanie 1999)[38].

Genetic algorithm is a type of evolutionary algorithm. In general terms genetic algorithm can be seen as an imitation of biological evolution for problem solving. A genetic algorithm combines the survival of the fittest with structured yet randomized information exchange. Genetic algorithms are no simple random walk, they exploit historical information to speculate on new search points expecting improved performance. The main research in genetic algorithms revolves around robustness, the balance between eficiency and efficacy for survival in different environments (Goldberg 1989)[37] Genetic algorithms are blind, they perform an effective search only requiring payoff values associated with individual strings (Goldberg 1989)[37]. Melanie 1999 mentions that although there no widely accepted definition for genetic algorithm in the evolutionary - computational community, most methods self called "genetic algorithm" have in common the use of population of chromosomes, selection according to fitness, crossover to produce new offsprings, and random mutation of the new offspring.

In our code we do not use all the characteristics described for a genetic algorithm. The characteristics for the algorithm resemble more those of evolutionary game theory. The agents are created at the beginning of the simulation and are assigned with a specific strategy which they will continue to use until they are eliminated or until the simulation ends. For this the agents created do not posses a chain of phenotypes from which they will occasionally choose and play a different strategy. In our simulation the strategy is assigned randomly and this assures that at least every possible strategy have representatives during the simulation. Additionally we use the concept of asexual reproduction, which means there is no two parents for each "newborn" agent, but instead it is copy of one of the individuals that had the higher utility(fitness) during the generation. For convenience a constant population number is maintained. The concept of mutation allows us to not always select the individual with the highest payoff to reproduce, but introduces the possibility of another agent with a less efficient strategy to appear. In addition to this concept of mutation. The use a variable called exploitation rate is introduced mainly because of the article by James March "Exploration and Exploitation in Organizational Learning" in which he names two concepts that can or should be taken in account when making a choice in the organizational aspect. But I considered these concepts valid to the context of our experiment, because the simulations made by the code are random, and though randomness should help to cope with any potential bias, the random interactions may favour one strategy over another potentially good potential strategy. March 1991 mentions that exploration includes things captured by terms such as search, variation, risk taking, experimentation, play, flexibility, discovery and innovation: whilst exploitation includes refinement, choice, production, efficiency, selection, implementation and execution (March 1991)[39]. In raw terms exploration implies looking for another feasible alternative, in the case of our code we can do this in any percentage of the population, or choosing the alternative that appears to generate the highest utility in every run. In the code it is implemented as an additional decision parameter to give more possibility to other less efficient strategies during a generation to reproduce. An exploitation rate of 1 (100%) means that only the agent with highest payoff has a possibility of reproducing itself and a 0 (0%) means that the whole population is considered a candidate for reproducing itself for the next generation, and this decision is let to be handled by a random choice function. For the purpose of this work, we will be focusing on the interaction between the same species. The model we will use is simplified.

3 Software development practices

Abstract

In this section a description on how the code was built will be found. The focus used to build the code was a discipline in computer programming called test driven development, a description of what it is will be given. Version control system will be explained, what is a version control system and the elemental use of the version control system Git will be explained, which is used for interaction between different people that contribute to a single project. An overview of the library built, explaining how the different modules contained.

3.1 Test Driven Development

The idea of OOP goes well with a programming discipline which is being used more commonly, and that is test-driven development(TDD). Is very similar to the idea of test first development (TDF) described by Kent Beck in "Extreme programming", and the main idea is to build a test to assert that the result from a small piece of the application code will be reached, this is very likely to make the programmer think of what is the result she wishes to obtain and very likely as consequence build a very efficient code. Usually a fundamental step of TDF is described as "write a test that will fail", then under that idea build the functional part of the code that will make that test pass. Test driven development (TDD) follows this same idea, but adds an additional ingredient which is refactoring. In the book "Test-driven development: an example" Kent talks about a TDD "mantra" which is described in 3 simple steps, the first two are the same principles as the ones of TDF. The first step he calls it "Red" and says to write a small test that doesn't work, the second step "Green" is to make the test work (by writing the functional code), and the last step "Refactor" which is getting rid of all possible redundancies in the code without affecting the functional code (Kent 2003)[40]. The testing is usually automated unit testing, the concept of unit testing is useful when using OOP, since the intention of OOP is to keep a clear and functional code with a certain degree of abstraction, unit testing is testing each small component of a program. It is important to say that TDD is intended to test the code in small steps, and not the final application of the code. So in TDD testing and designing the code go together in small steps to produce a final simple and testable code.

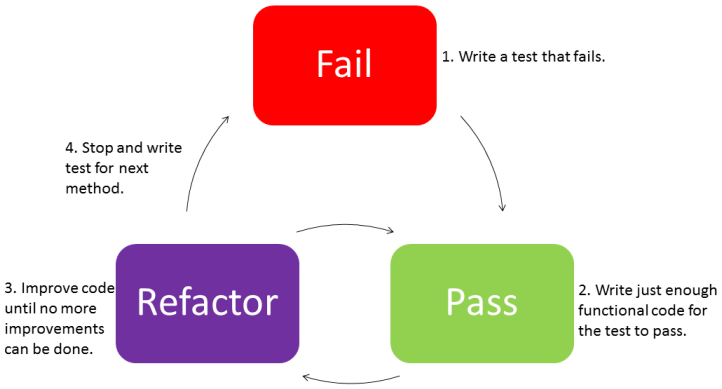

Figure 3.1. Basic TDD diagram.

For testing in this code **unittest** test module contained in the standard Python library was used. **unittest** from Python supports some important concepts for testing for example test fixture, test case, test suite and test runner (Van Rossum 2003)[41]. For the code given that each function of each module was tested we used

the smallest unit for testing which is the class **TestCase**. Although there exist many methods for testing in **TestCase**, for this code the main methods used when testing were:

- **assertEqual:** Asserts an element a is equal to an element b (a == b) if the values are not equal the test fails.
- **assertNotEqual:** Asserts an element a is not equal to an element b (a !=b) if the values are equal the test fails.
- **assertTrue:** Asserts an argument passed is True (bool(x) is True) test fails if argument is not True.
- **assertFalse:** Asserts an argument passed is False (bool(x) is False) test fails if argument is not False.
- **assertIn:** Assert an element a is contained in b (a in b) test fails if the element a is not part of b (i.e. a = 3, and b = [1, 5, 6, 8 ,10], value of a (3) is not contained in the list so the test fails).

‘

3.2 Version Control

When teaming with other people with common goals for developing any kind of project, one of the most important elements is communication. There should be a common agreement for a communication channel so every element of the team can be updated with any changes made during the project. Developing a project in computer programming is not the exception, and from some years back computer programmers have been using very efficient communication channels. A very useful tool are version control systems (VCS). The main goal from a VCS is to keep track of the development of a project with a very intelligent system that allows people (programmers) to track changes in a project through time. It keeps record of changes that have been made to a project and allows to retrieve the information from those other points in time. VCS basically can create a backups for every changes made to a project, revert specific files, the whole project to a previous state, compare changes over time, see who did last modifications that could be causing a problem, who introduced an issue, when among other (Chacon 2009)[42]. Version controlled systems can be local, centralized or distributed, to collaborate in a project nowadays the most popular and perhaps effective choice is to use distributed version control systems where the users fully mirror the set of files or directories under version control commonly known as repository (Chacon 2009)[42]. This means that each contributor has a copy of the full repository and if the main server has a malfunction and information is lost, it can be restored by any contributor.

The VCS used for this project was Git. Git was authored and developed by the creator of Linux kernel, Linus Trovalds in 2005 and many other developers from the Linux community after a commercial break down between the community that developed Linux kernel and the company that provided them DVCS called BitKeeper (Chacon 2009)[42] .

Git thinks of data like snapshots of a miniature filesystem. Git has a command that is called commit and this helps to record the different snapshots taken in different points of time, one can think of commits as some kind of milestones for the individual and the collective part of the project. An important fact about Git is that since all the history of the project being developed is in each users local disk, the speed to browse around the history of the project is very efficient. This also allows the user to work even if there is no access to the network where the repository is.

For this project only basic, but still essential features of Git were used. The webpage where the repository for the project was stored is Github, first a user name needs to be created in this webpage and create a repository. Which is relatively easy by following the instructions given in the webpage. It basically asks us to give a name a name and an optional description, the repository can be public or private, the repository for this project is public. It is advised to create a README for describing the project. At the end of the steps one selects create repository and the repository is created. Setting up Git in the pc is not hard, for this I used

Git bash which is a build environment shell for Windows and enables Windows to use the Git commands. There is a lot of information about how to set up Git bash in the web which involves generating a key, by adding a command in bash along with which I added the email address used when signing up in Github in this case and then follow the steps and it is set. To start working with a repository from Github either our own or someone else's. We copy an URL that appears on the right bar in the page from the repository we intend to work with. This URL was pasted into the Git bash in the same line after the following command **git clone** and with this a local clone of the repository is created. Once the local directory for the project is in the pc we start working on it. Every file that will be included in the project should be in that directory, the state of the files when they are first created is untracked. The following are the commands used during for Git during the project and a short description for each is provided:

- **git status:** Used to see the what the status on the working directory. It will display two sections, one with the staged changes that are ready to be committed and another section displaying the changes that have not been staged, along with a small legend on the left side of the name of the file, that indicates what changes have been done to that file (deleted, modified, new file, etc.)
- **git add:** Used to stage the changes we have made in the directory for the local repository clone specific files. Before using this command all changes that have been made in the directory are considered unstaged. This means that any new file added or any already existing file that has been modified has to be staged with this command in order to include them in the next commit, any changes made that have not been staged with **git add** will not be considered in the next commit.

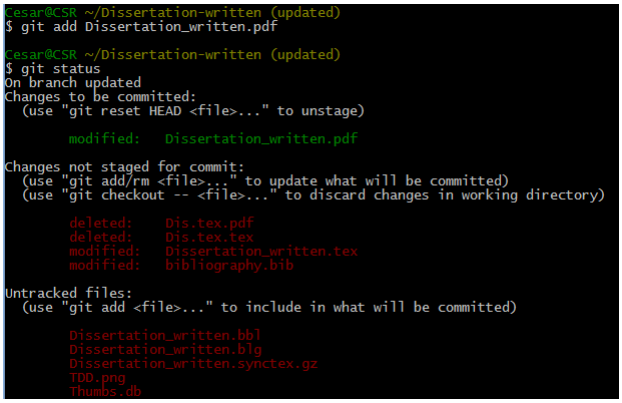

Figure 3.2: Example of one file staged with git add.

- **git commit:** Takes a snapshot of the status of the local cloned repository, of course the files that considers are the ones that $textbfgitadd$ was previously used on. Creates a new point in the history of the project. Once this command is run, it commonly opens a text editor where I introduced the changes that were made in the files that are being changed in this commit.

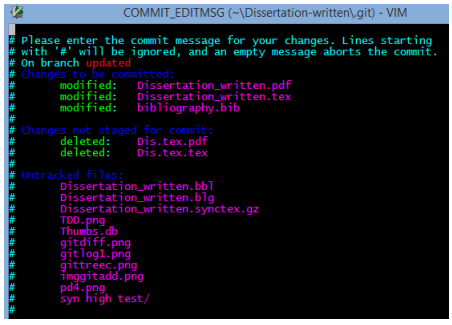

Figure 3.3: Example after using git commit asking for message for the commit.

- **git diff:** Using this command directly displays the content that has been changed in the project since the last commit and that are not staged for the next commit. If the command is run **git diff - - cached** it will display the changes that have been staged. If we wish to see staged and unstaged changes together we use **git diff HEAD** and if there is no interest in watching the specific changes but we want to display more information than just the one presented by **git status** we add **- - stat** after the different options we had with **git diff [options] - - stat** and this displays a summary of the changes.

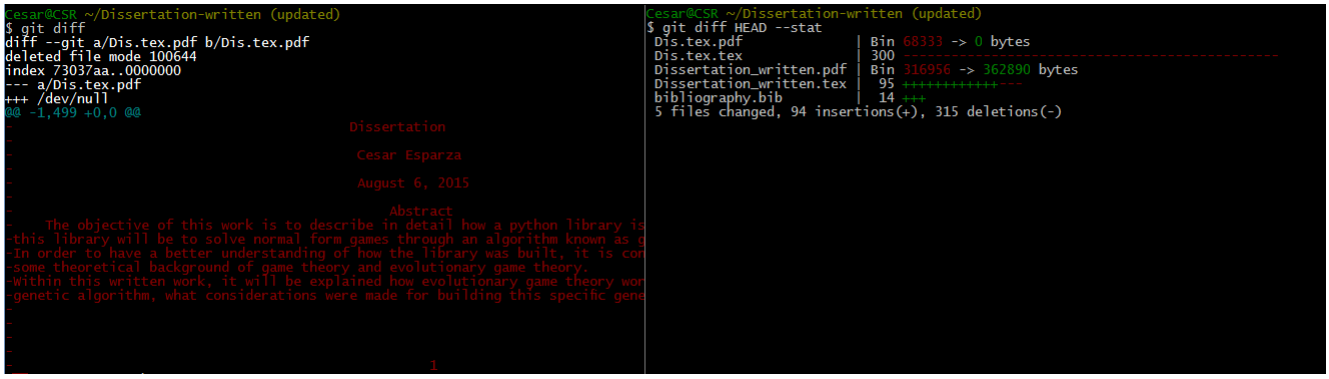

Figure 3.4: Example of general output with git diff and git diff HEAD - - stat (red color represent the deleted lines and green the added) .

- **git fetch [remote-name]:** This command pulls all the data from the remote project that is not contained the current copy we hold in our local repository clone. Basically used to update with all the changes other users (contributors) to the project have made, but it does not merge it with the work we have done because this has to be done manually. The name given to the remote repository we created a clone from is usually origin.
- **git branch:** Without any arguments the command gives a list of the existing branches in the local repository.The command followed with the name of the branch **git branch (branchname)** will create a branch with that given name out of the main project line (master), in the last commit made before the branch. So if we continue working on the master and then we switch to the branch all the changes made to that point will be reverted to the context where the branch was created. If we wish to delete a local branch we use **git branch -d (branchname)**. To delete a remote branch we use the command **git push (remote-name) :(branchname)**.
- **git checkout (branchname):** This command is used to change between the different existing branches. A way of creating a branch and changing to it at the same time is a shortcut provided by Git **git checkout -b (branchname)**.
- **git merge:** This command is used to merge the changes that have been made in a branch are complete. And what is usually done is change into the main project line (master) with **git checkout (name of the branch we w** and then use **git merge (name of the branch wished to be merged in)**. This way the changes, if there exists no conflict, will be merged into the main project line (master).
- **git pull:** Basically does first a **git fetch** immediately followed by a **git merge** from the tracked branch into the branch we are currently in. This command basically does the same as those two commands, but it can if there are changes that may cause a problem, it might make the process a bit more complex.

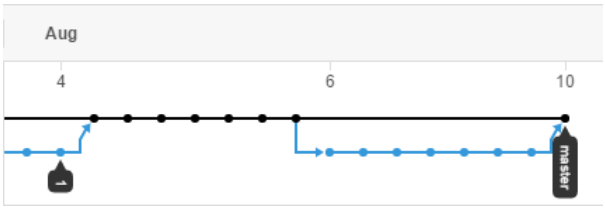

Figure 3.5: Example diagram from Github of a master branch with a branch 1 for being merged.

- **git push:** This command helps to update the local changes we have made to the remote repository. And the way to use it is **git push (name given to the remote) (name of the local branch)**. We just need to be careful to not overwrite changes.
- **git log:** This command displays all the commit messages that have been previously used in the state of the project we currently are in. There is an alternative command that displays how the project has been modified up to the point we currently are in and is **git log - - oneline - - decorate - - graph - - all**

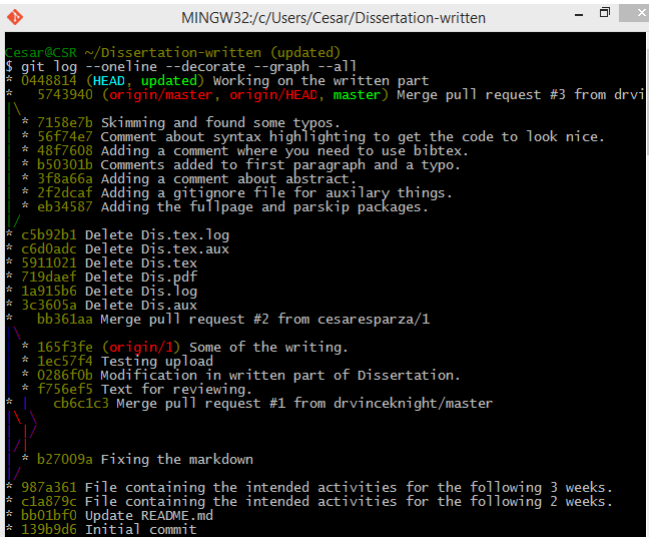

Figure 3.6: Basic example of git log oneline.

3.3 Library (Package)

The name chosen for the library is 'Ablearn' and it stands for agent-based learning, the intention is for different people to contribute to this library in the future for it to grow and therefore have a wider range in application at some point in time.

The library is the file that contains all the components of the code. Because of the object oriented properties that Python facilitates, the code is segmented into 6 modules. According to their function the order in describing each is not relevant.

3.3.1 Population module

The population module is intended to create agents that will be used in the simulation. The reason of being called agents instead of the classic name player, is because of the intended focus in agent-based modeling as mentioned before. It was build in a very generic way so it can be used, if possible, for any other type of interaction and algorithm in the future. The basic information contained in this class is very generic and it's attributes can be modified when using it. This module contains the instructions to create a class named Agent the class has an initializatin(__init__) method that takes as parameters strategies, utility and the possibility to add a label. After the initialization, another method is presented increment_utility which is set for incrementing the agents utility, the criteria for this increment_utility will be explained in another module.

- **Strategies:** Each created agent will be assigned a strategy
- **Utility:** The utility each created agent generates after each interaction with another agent.
- **Label:** The possibility of adding a label to each created agent, to track their performance.

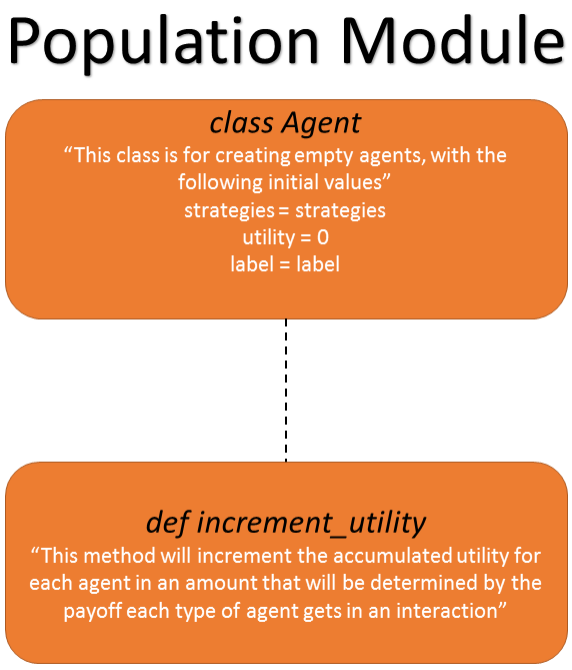

Figure 3.7: Diagram of population module.

3.3.2 Environments module

The environments module for this project is the representation of the environment in which the agents will interact.The environment created within this module for this project is **bimatrix_random_environment** and has only two characteristics. It is set to make two agents interact pairing them randomly and it also sets the rules which these paired agents use to interact. Some of the methods contained in this module make use of a module named random from python. This module is imported when the environment module is executed.

Bimatrix_random_environment creates a class BiMatrixRandomEnv, named after the characteristics bimatrix and random environment. This class has an initialization (**__init__**) method that takes as parameters **number_of_agents** and **bimatrix**, within the initialization some variables are defined, these variables along with the parameters will now be explained:

- **Number_of_agents:** Input by user, total population of agents regardless of the type of agent(i.e. row agent or column agent).
- **Bimatrix** Input by user, bimatrix of payoffs(can be symmetric or assymetric).
- **Number_of_row_agents:** Result from dividing by 2 the previously input value number_of_agents. And gives the number of row agents.
- **Number_of_col_agents:** Result from dividing by 2 the previously input value number_of_agents. And gives the number of column agents.
- **Number_of_row_strategies:** Number of strategies that will be available for row agents. Calculated by counting how many rows the bimatrix has.
- **Number_of_col_strategies:** Number of strategies that will be available for column agents. Calculated by counting how many columns the bimatrix has.
- **Row_strategies:** List containing the available strategies for row agents.
- **Col_strategies:** List containing the available strategies for column agents.

After the initialization, a method **interact** is defined. This method first defines a variable called **pairs** which is assigned to a function **randomly_pair_agents** that will be explained later. It also contains a **"for" loop** this loop within other things contains a variable which is set to a function **strategies_to_utilities** which will be explained, variables in the **"for" loop** are the following:

- **Ra**: Used in the code during for loops in exchange of **row_agents**.
- **Ca**: Used in the code during for loops in exchange of **col_agents**.
- **Pairs**: Variable created to represent the group of paired **row_agents** with **col_agents**.
- **Utility**: Variable used to obtain the utilities resulting from the interaction from each pair of agents (**row_agents and col_agents**).
- **Agent.increment_utility**: The increment utility function is defined in the population model. The structure for in this for loop is as follows:
  agent.increment_utility(utility[x]) and what it does is assign the function increment utility to an agent can be **ra (row_agent)** or **ca (col_agent)**, the **utility** in parenthesis was assigned in the previous variable, and it is only calling the value with the position x in the list. Given that we only have 2 types of players (we are using a bimatrix) x can be either 0 or 1.

Following **interact** the method previously mentioned **strategies_to_utilities** is defined. This method is in charge of obtaining the specific pair of utilities (assigned to row and column) from the **bimatrix**. It then

returns these values to the **utility** variable in the **interact** method and interact uses it to assign the utilities to each agent.

After **strategies_to_utilities** the method **randomly_pair_agents** is defined. This method is used by **interact** too, and what it does is that the previously created row and column agents that are contained in lists are randomly selected (one of each type) and then paired so they can interact.

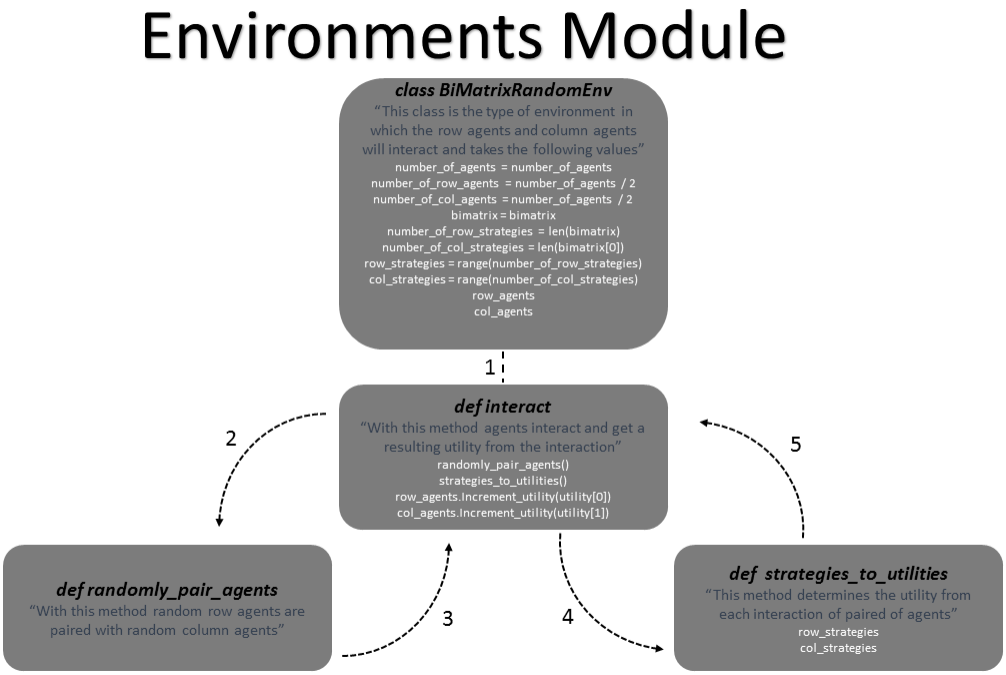

Figure 3.8: Diagram of environments module.

3.3.3 Algorithms module

The **algorithms module** is a module that is meant to contain different algorithms by which the created **agents** from **population module** will be processed. For this project **algorithms module** contains a **genetic algorithm**. The **genetic algorithm** is named **genetic** and it creates a class **Genetic**. In the initialization (**__init__**) takes the following parameters:

- **generations:** Are the number of generations (times) the whole code will be run.
- **rounds_per_generation:** Are the number of times the interaction of the agents will be given within each generation.
- **death_rate:** Represents the proportion of agents that will be eliminated after each generation has run.
- **mutation_rate:** This parameter is given as a limit for agents with the highest accumulated utility not being reproduced. A **mutation_rate** value is established and is compared against a **random.random()** number. Every time the **mutation_rate** is higher than the value given by the **random.random()** a mutation will occur. This comparison occurs every time a new agent is created when all the interactions within a generation have happened. This means that each new agent is passed under this condition. The numbers given by **random.random()** are decimal numbers, which means that the range for **mutation_rate** should also be in decimal numbers from **0 to 1**.
- **exploitation_rate:** Is a variable introduced to determine if the concept of *exploitation or exploration* will be used. Where a value of 1 (100%) represents a total exploitation, this means reproducing only the agent with the highest utility for each type of agent (**row_agents and col_agents**) and a value of 0 (0%) is used to consider the whole population of agents.

Following the initialization (**__init__**), the method **assign_strategies** is found. This method takes parameters **agents**, which according to the type of agent in turn will be represented by **row_agents** or **col_agents** and the parameter **agents_strategies**, also which according to the type of agent in turn will be represented by **row_strategies** or **col_strategies**. This method is used after the different types of agents are created in the **initialization** of the class **BiMatrixRandomEnv** of the module **environments**. The **assign_strategies** method assigns a random strategy for each agent according to the strategies available given their type through a **"for" loop** and **random.choice(agents_strategies)** this random choice selects from the **row_strategies or col_strategies** randomly according to the type of agent.

After the **assign_strategies** method, comes a method named **kill_agents**. This method takes as parameters **agents, number_of_agents, number_of_x_agents, and agents_strategies**. In this method a variable **number_of_deaths_per_generation** is created, which determines the number of agents that will be eliminated for each type of agent group (**row_agents or col_agents**) it does so by taking the integer value from the product of the multiplication of the variables **death_rate number_of_x_agents**(this last variable represents the total number of row_agents or col_agents). The value **number_of_deaths_per_generation** is used in a **"while" loop** to eliminate the agents while the condition is not satisfied. Within the "while" loop the agents (**row_agents or col_agents)** are sorted according to their accumulated **utility** from smallest to highest and the one with the lowest accumulated **utility** is deleted from the list of the corresponding type of agents.

After the method **kill_agents** comes the method **reproduce_agents**. This method takes as parameters **agents, number_of_agents, number_of_x_agents and agents_strategies**. A variable **choice** is created by taking the integer value of (**((number_of_agents / 2) (exploitation_rate - 1) + 0.05) -1** this value is used within the "while" loop that follows. The "while" loop used in this method works while the existing number of row or col agents is less than the number of row or col agents created at the beginning of the simulation. Within this loop, the agents are sorted according to their accumulated **utility** values from low to high, and a **copy.deepcopy** function is used to create a copy of an agents chosen with **random.choice(agents[choice:])** where the variable **choice** will indicate the start of the range of agents

that can be chosen within the list of agents. After this within the same loop and "if" statement is presented which condition is that if the **mutation_rate** value is greater than a **random.random()** number, the agent that will be reproduced can be any agent (row or col agent) with any value of accumulated **utility** except for the agent (row or col agent) with the highest accumulated **utility** value of that generation.

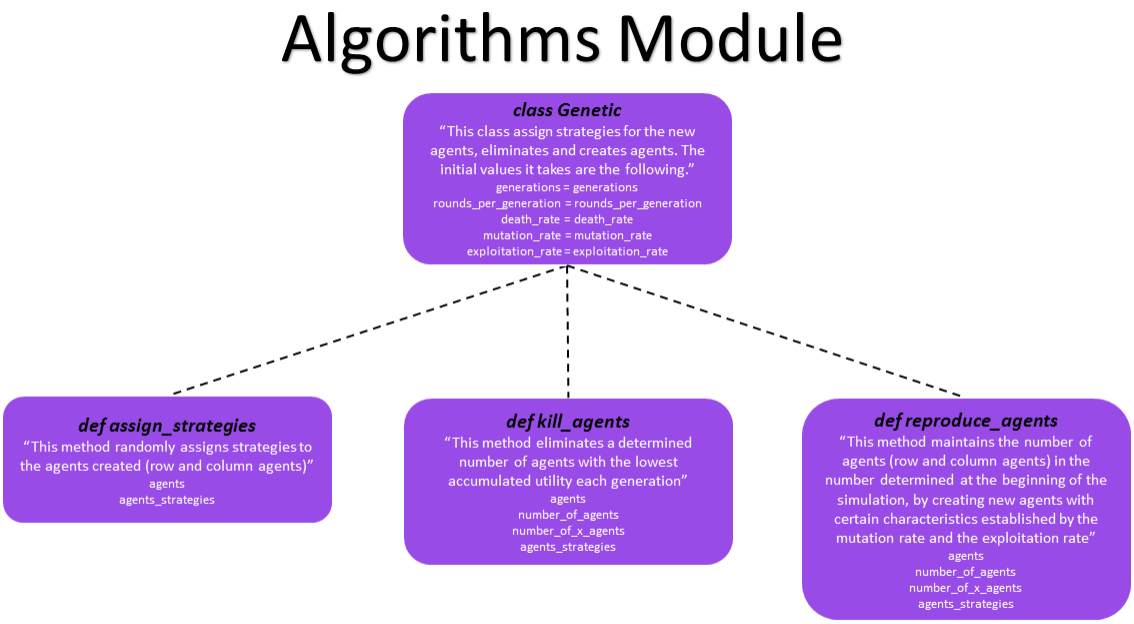

Figure 3.9: Diagram of algorithms module.

3.3.4 Simulation module

The **simulation module** is a module that is makes all the other modules interact with each other. For this project **simulation module** is named **simulation** and it creates a class **Simulation**. In the initialization (**__init__**) takes the parameters used in the other modules in addition to another parameter named **tog** which stands for "type of game". There are 7 predetermined games within the code, which can be selected by typing the correct value of **tog**. **tog** and the other parameters taken by the class **Simulation** have the following characteristics:

- **tog:** Stands for "type of game" and it serves to pick one of the available choices of bimatrix that have been coded in. The values that can be selected have to go between single quotation marks and are the following: **'pd'** an example of prisoner's dilemma, **'psr'** an example of the game paper-scissors-rock, **'mp'** an example of matching pennies, **'bos'** an example of battle of sexes game, **'hd'**, **'sh'** an example of stag hunt game, **'cs'** an example of choosing sides game and **'axl'** an example emulating the first tournament from Robert Axelrod with 8 out of the 14 original strategies. This strategies are taken from the axelrod package available for python and with the following strategies: *Tit for Tat, Grofman, Shubik, Grudger, Davis, Feld, Joss, Tullock and a Random strategy.*
- **na:** Stands for the value of **number_of_agents** explained in the **bimatrix_random_environment** module.

- **ge:** Stands for the value of **generations** explained in the **genetic** algorithm module. And is the number of generations (times) the whole code will be run.
- **rpg:** Stands for the value of **rounds_per_generation** explained in the **genetic** algorithm module. Is the number of times the interaction of the agents will be given within each generation.
- **dr:** Stands for **death_rate** explained in the **genetic** algorithm module. And represents the proportion of agents that will be eliminated after each generation has run.
- **mr:** Stands for **mutation_rate** explained in the **genetic** algorithm module. And this parameter is given as a limit for agents with the highest accumulated utility not being reproduced. A **mutation_rate** value is established and is compared against a **random.random()** number. Every time the **mutation_rate** is higher than the value given by the **random.random()** a mutation will occur. This comparison occurs every time a new agent is created when all the interactions within a generation have happened. This means that each new agent is passed under this condition. The numbers given by **random.random()** are decimal numbers, which means that the range for **mutation_rate** should also be in decimal numbers from **0 to 1**.
- **er:** Stands for **exploitation_rate** explained in the **genetic** algorithm module. And it is a variable introduced to determine if the concept of *exploitation or exploration* will be used. Where a value of 1 (100%) represents a total exploitation, this means reproducing only the agent with the highest utility for each type of agent (**row_agents and col_agents**) and a value of 0 (0%) is used to consider the whole population of agents.

In the initialization (__init__) method an instance of the class **Genetic** from the genetic module is created, an instance for the class **Agents** from the population module is created also and when selecting the **tog** value, an instance of the class **BiMatrixRandomEnv** from the environment module is created. After the instances are created within the initialization method, the function **ga.assign_strategies()** from the genetic algorithm module is used to assign the values for the strategies for each type of agent (**row_agents and col_agents**) that have been created through the instance of the module environment.After pair of lists with the names **row_accumulated_strategies** and **col_acummulated_strategies** are created respectively and the purpose for each list is the following:

- **row_accumulated_strategies**: Creates a list, that within itself contains a certain number of individual lists according to the length of **row_strategies** which is the number of strategies available for the **row_agents** obtained from the module environment.
- **col_accumulated_strategies**: Creates a list, that within itself contains a certain number of individual lists according to the length of **col_strategies** which is the number of strategies available for the **col_agents** obtained from the module environment.

After the initialization method has created the different instances from the other modules, a method with the name **run(plot=False)** is presented. This method is used to start running the whole simulation (all the modules). And this method take an attribute **plot** which is set by default as **False**, if using the method **run(plot=True)**, the simulation will plot the results from each generation in a graph. The graph is produced with the module **matplotlib** from python and the functions used are **pyplot and cm(color map)**. The graph is set in interactive mode by the function **ion( )**. The limit for the y axis is set **ylim(0, 1)**, the limit for the x axis is the number of **generations**. The x axis is labeled **xlabel("Generations")** and the y axis is labeled **ylabel("Probability")**. For the coloring the lines representing the different strategies for each type of agents color map is used for the **row_accumulated_strategies**[ ] the color map copper is used and for the **col_accumulated_strategies**[ ] the color map winter is used, how these **accumulated_strategies** values were obtained will be explained later, the thing to know about them is that each of them represent the proportion within a type of agent that is using a certain strategy. The graph indicates how the population of the different types of agents with strategies varied through time, the value of interest from this is the strategy the types of agents are using.

After the code for creating the graph a function **generations_passing( )** is presented, this function takes attributes **generations**, **rounds_per_generation**, **number_of_agents**, **row_agents**, **col_agents**, and **plot** and takes us to the next method.

After the **run** method, there is a method **generations_passing** that takes attributes **generations**, **row_agents**, **col_agents**, **number_of_agents**, **plot**. In this method there is a "for" loop that loops until the **generations** value is reached, within this loop exist two other "for" loops. he first one loops around the **row_agents** and **col_agents** setting for each agent the **utility** value to 0. The second "for" loop loops according to the **number_of_rounds** chosen at the beginning of the simulation. Within this loop we find the function **interact( )** from the module environment, explained before, which in general terms makes the **row_agents and col_agents** interact with each other and as a result obtains the payoff for each type of agent and an attributed **utility**. The loop from the rounds causes this process to repeat several times and when the last round is played all the different agents have an accumulated utility product of the repeated interactions. When the second loop concludes a function **distributions( )** follows. This function is for the method **distributions** that takes attributes **number_of_agents**, **row_agents**, **col_agents**, **row_strategies**, **col_strategies**, and **plot**.

Before the method **distributions** there is a method **proportion_classified_strategies** that takes attributes **agents and strategies**. This method will do some calculations when the code is using the method **distributions**. The method first creates a varibale **frequency_per_strategy_per_agent** with an empty list. Then it has a "main" "for" loop that loops around the variable **strategies** (this variable can be **row_strategies** or **col_strategies** according to which values are being used) of the **agent** in turn and appends a value of 0 to the list. Then an "secondary" loop is contained within the previous loop of **strategies**, this loops around the agents in turn (can be **row_agents** or **col_agents**) and contains an "if" statement, which conditions if the strategy from the **agent** in turn is equal to the counter in the "main" loop (**strategies**) the last value of the list **frequency_per_strategy_per_agent** will increment by 1. When both loops finish the function returns a list with all the values that are contained in the **frequency_per_strategy_per_agent**, each of those values are devided by the number agents. This values are the proportion that strategy represents in the type of agents in turn. It is important to note that the variable **agents** taken by this function can be either **row_agents** or **col_agents** and the variable **strategies** can be either **row_strategies** or **col_strategies**.

When the method **distributions** starts first it creates a variable **row_strategies_distribution** this variable contains a function **proportion_classified_strategies( )** that takes as attributes **row_agents** and **row_strategies**. And then has a "for" loop which loops around the number of **row_strategies** and appends into the list created at the initialization **row_accumulated_strategies** the resulting value from the **row_strategies_distribution**. According to which strategy it belongs, it appends the value to the corresponding list. Then the method creates another variable which has the same process as the previous, but with **col_agents**. The variable it creates is **col_strategies_distribution** this variable contains a function **proportion_classified_strategies( )** that takes as attributes **col_agents and col_strategies**. Then has a "for" loop which loops around the number of **col_strategies** and appends into the list created at the initialization **col_accumulated_strategies** the resulting value from the **col_strategies_distribution**. According to which strategy it belongs, it appends the value to the corresponding list. After the two variables, and the append of **row_strategies_distribution**[ ] to **row_accumulated_strategies**[ ] and **col_strategies_distribution**[ ] to **col_accumulated_strategies**[ ] is finished. The method **prints** for lines the first one contains the following text *"Row players' strategy distribution:"*, the second line prints the list **row_strategies_distribution**, the third line prints the text *"Column players' strategy distribution:"*, and the fourth line prints the list **col_strategies_distribution**. After these lines come two lines, each with the function **kill_agents( )** from the genetic algorithm module. One of which is for eliminating **row_agents** and the other for eliminating **col_agents**. Then we have other two lines, each with the function **reproduce_agents( )** from the genetic algorithm module. One to create the **row_agents** that were previously eliminated until reaching the required number of **row_agents** and the other to create **col_agents** until reaching the required number of **col_agents**. After the last **reproduce_agents( )** function, a code to update

the graph is run (if it was chosen to graph at the beginning of the simulation). The whole cycle is repeated until the simulation reaches the number of **generations** that was established at the beginning.

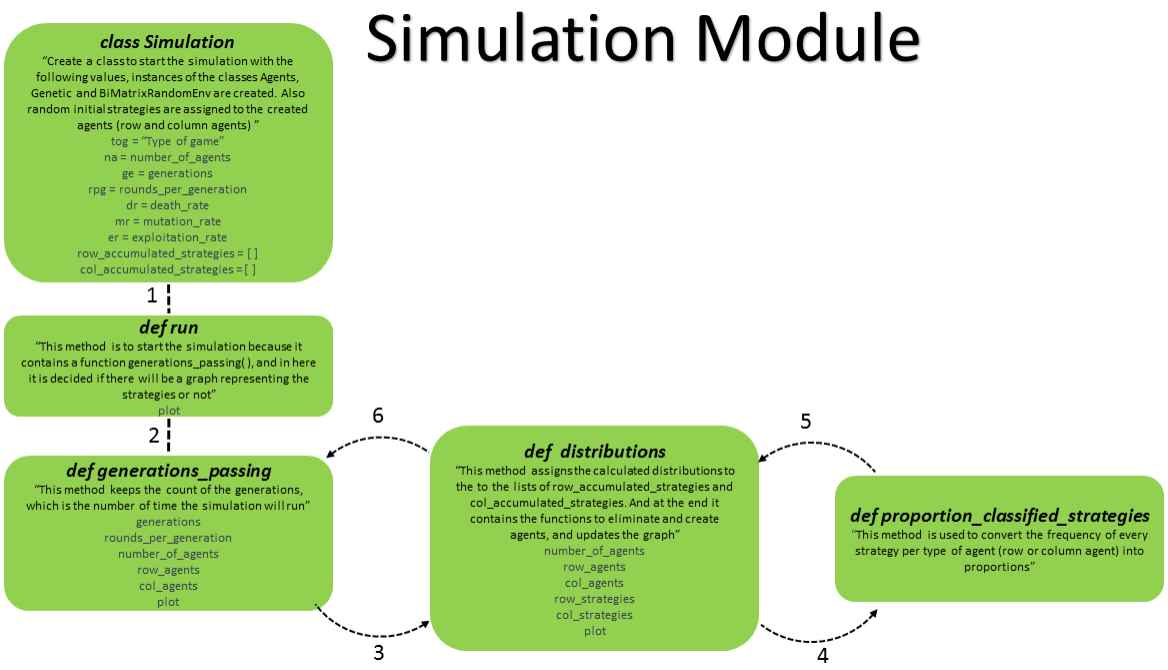

Figure 3.10: Diagram of simulation module.

4 Application of the code

Abstract

In this section the runs of the program will be discussed. What was consider when running a simulation. An explanation of what games were simulated, and of the results against the known techniques to indicate how the program outputs our expected results when running it with known classic game theory games. A comparison of the results given by the library 'Axelrod' from python with results from this code 'Ablearn'.

4.1 How the simulation was built.

After building the code the objective is to determine if the program gives us the expected results. We know that an evolutionary stable strategy is a strategy that survives through time. And as we have mentioned before there are certain conditions that we can verify in order to identify and evolutionary stable strategy, as a reminder those conditions are the following:

- *If X (any strategy s) is not a Nash equilibrium, then X is not evolutionarily stable.*
- *If X (any strategy s) is evolutionarily stable, then X is a Nash equilibrium.*
- *If X (any strategy s) is a Nash equilibrium, but $a = c$ and $b < d$ then X is not evolutionarily stable.*

The following tables represent some of the most well known examples of games in the normal form.

Perhaps the most known is the the prisoner's dilemma which was mentioned before and it basically has two Nash equilibria strategies one in which both cooperate and the other in which non cooperate.

		Prisoner 2	
		Cooperate	Defect
Prisoner1	Cooperate	3, 3	0, 5
	Defect	5, 0	1, 1

Figure 4.1: Prisoner's Dilemma

Matching pennies which is a pure conflict zero-sum game in which the winner of the game takes all and the loser ends up losing her share. We see that there is no scenario where both players can agree in a strategy. Player 1 preference of choice is from tossing a coin involves getting 2 heads or 2 tails, whilst player 2 prefers having a mixed combination. The equilibrium for this game is a mixed Nash equilibrium.

		Player 2	
		Heads	Tails
Player 1	Heads	1, -1	-1, 1
	Tails	-1, 1	1, -1

Figure 4.2: Matching Pennies

The battle of sexes game is a coordination game in which both agents cannot exchange information about what option out of two to choose, they both have a preferred strategy, but if they both choose their preferred strategy they have no payoff from it because they rather choose the same strategy and concur, even if it has a higher payoff for one of them than the other. In the example from the table there is a representation of preferences, and we should assume that it is a couple trying to decide where to go, the female agent prefers going to the opera, whilst the male agent prefers going to watch football. But we see that if they both end up choosing different strategies from each other they get a payoff of 0.

Female

Male		Opera	Football
	Opera	1, 2	0, 0
	Football	0, 0	2, 1

Figure 4.3: Battle of Sexes

And the hawk-dove game. This game is often used in evolutionary game theory. And it represents the 2 strategies an agent can choose, and the result of the interaction. There is an aggressive strategy which is the hawk strategy and a passive strategy which is the dove. When both agents choose to play hawk, the payoff they get is 0, the explanation is that since they are both aggressive, the possible payoff they could have had from the resource they are competing for is not greater than the cost they pay for playing this strategy against each other. When they both choose dove, they split in equal parts the resource and the payoff they receive is the same for both. When one plays hawk and the other dove, the agent using hawk strategy gets a higher payoff than the one using the dove strategy.

	Hawk	Dove
Hawk	0, 0	3, 1
Dovel	1, 3	2, 2

Figure 4.4: Hawk-Dove

The following table indicates how the simulations for prisoner's dilemma were run.

Num.	Population	Generations	Rounds per Gen.	Death Rate	Mutation Rate	Exploitation Rate
1	1000	50	50	0.1	0.1	1
2	1000	50	25	0.1	0.1	1
3	1000	50	5	0.1	0.1	1
4	1000	50	5	0.5	0.1	1
5	1000	50	5	0.9	0.1	1
6	1000	50	5	0.9	0.5	1
7	1000	50	5	0.9	0.9	1
8	1000	50	5	0.1	0.5	1
9	1000	50	5	0.1	0.9	1
10	1000	50	5	0.1	0.1	0.5
11	1000	50	5	0.5	0.1	0.5
12	1000	50	5	0.9	0.1	0.5
13	1000	50	5	0.1	0.5	0.5
14	1000	50	5	0.5	0.5	0.5
15	1000	50	5	0.9	0.5	0.5
16	1000	50	5	0.1	0.9	0.5
17	1000	50	5	0.5	0.9	0.5
18	1000	50	5	0.9	0.9	0.5
19	1000	50	5	0.1	0.1	0.1
20	1000	50	5	0.5	0.1	0.1
21	1000	50	5	0.9	0.1	0.1
22	1000	50	5	0.1	0.5	0.1
23	1000	50	5	0.5	0.5	0.1
24	1000	50	5	0.9	0.5	0.1
25	1000	50	5	0.1	0.9	0.1
26	1000	50	5	0.5	0.9	0.1
27	1000	50	5	0.9	0.9	0.1

Table 4.5: Configuration of simulations for prisoner's dilemma.

4.2 Prisoner's Dilemma

For Prisoner's dilemma I do not consider relevant to vary the number of generations to test for any changes. This is because the interaction of the agents and their strategies is given in the level of rounds. In other words, the rounds give the proportion of strategies for each generation. A variation in the number of generations may be considered when a clear pattern in the data is not easy to identify. However, I do consider important to determine if the number of rounds make a difference in the results. Hence the simulation will be tested with different number of rounds. It will be done under the conditions 1, 2 and 3 of the table 4.5 (Configuration of simulations).

The simulations start by assigning the following proportions to row and column agents to the strategies cooperate and defect

	Cooperate	Defect
Row Agents	0.49	0.51
Column Agents	0.502	0.498

Figure 4.6: Initial Prisoners Dilemma distribution for simulation 1.

	Cooperate	Defect
Row Agents	0.489	0.514
Column Agents	0.524	0.476

Figure 4.7 : Initial Prisoners Dilemma distribution for simulation 2.

	Cooperate	Defect
Row Agents	0.512	0.488
Column Agents	0.49	0.51

Figure 4.8: Initial Prisoners Dilemma distribution for simulation 3.

The 3 simulations start with very similar proportions for each strategy for agent type. They all start close to 0.5 so there is not a significant difference at this stage.

By generation 5 it can be seen that the strategy 'Defect' is dominating.

	Cooperate	Defect
Row Agents	0.348	0.652
Column Agents	0.24	0.76

Figure 4.9: Generation 5 Prisoners Dilemma distribution for simulation 1.

	Cooperate	Defect
Row Agents	0.25	0.75
Column Agents	0.378	0.622

Figure 4.10: Generation 5 Prisoners Dilemma distribution for simulation 2.

	Cooperate	Defect
Row Agents	0.35	0.65
Column Agents	0.314	0.686

Figure 4.11: Generation 5 Prisoners Dilemma distribution for simulation 3.

For simulation 2 in generation 5 it can be seen that the proportions are slightly different than first simulation. For row agents the proportion increased and is very close to the proportion of the column agents in generation

5 from the previous simulation 1 and for the column agents decreased and is very close to the proportion for row agents in the simulation 1. In simulation 3 for this generation the value for strategy 'Defect' for row agents is slightly higher than simulation 1 and slightly smaller than simulation 2. And for the same strategy in column agents compared to simulation 1 it decreased slightly and to simulation 2 it increased. The differences presented between simulations do not appear significant for this generation.

By generation 10 it can be seen that the dominance of strategy 'Defect' is almost absolute in the 3 simulations.

	Cooperate	Defect
Row Agents	0.054	0.946
Column Agents	0.074	0.926

Figure 4.12: Generation 10 Prisoners Dilemma distribution for simulation 1.

	Cooperate	Defect
Row Agents	0.018	0.982
Column Agents	0.094	0.906

Figure 4.13: Generation 10 Prisoners Dilemma distribution for simulation 2.

	Cooperate	Defect
Row Agents	0.082	0.918
Column Agents	0.07	0.93

Figure 4.14: Generation 10 Prisoners Dilemma distribution for simulation 3.

In simulation 2 for generation 10 with respect to simulation 1, a small increase in the strategy 'Defect' for row agents, and a small decrease in the same strategy for column agents. For simulation 3 strategy 'Defect' for row agents is slightly decreased with respect to the value in simulation 1 and 2. For the same strategy in column agents it increased slightly when compared to simulation 1 and 2. In this generation for the 3 different simulations, the value for strategy 'Defect' are above 0.9, but again there is no obvious pattern so the variations may be given by the effect of the randomized variables for each simulation.

In generation 25 it can be seen that the condition persists and the strategy 'Defect' for both types of agents dominates in frequency.

	Cooperate	Defect
Row Agents	0.068	0.932
Column Agents	0.054	0.946

Figure 4.15: Generation 25 Prisoners Dilemma distribution for simulation 1.

	Cooperate	Defect
Row Agents	0.045	0.955
Column Agents	0.047	0.953

Figure 4.16: Generation 25 Prisoners Dilemma distribution for simulation 2.

	Cooperate	Defect
Row Agents	0.072	0.928
Column Agents	0.054	0.972

Figure 4.17: Generation 25 Prisoners Dilemma distribution for this simulation 3.

In simulation 2, the proportions increased slightly for strategy defect for both agents in relation to simulation 1. For simulation 3 the proportions are slightly smaller for the strategy 'Defect' for row agents than in simulations 1 and 2. For same strategy in column agents is slightly higher than values from simulations 1 and 2. Still all the values for strategy 'Defect' are above 0.9.

For generation 50, the last generation we see that strategy 'Defect' dominates the strategy 'Cooperate' for all simulations.The graphs from the 3 different simulations are presented.

	Cooperate	Defect
Row Agents	0.01	0.99
Column Agents	0.044	0.956

Figure 4.18: Generation 50 Prisoners Dilemma distribution for simulation 1.

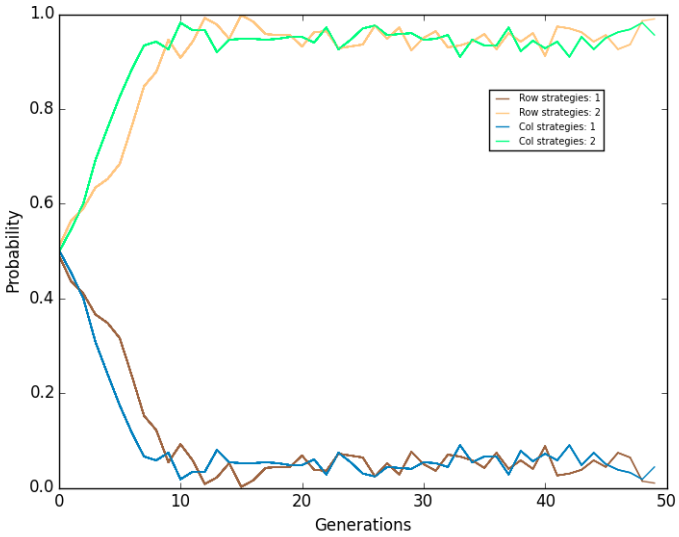

Figure ????: Prisoner's dilemma 50 rounds per generation.

	Cooperate	Defect
Row Agents	0.024	0.976
Column Agents	0.08	0.92

Figure 4.19: Generation 50 Prisoners Dilemma distribution for this simulation.

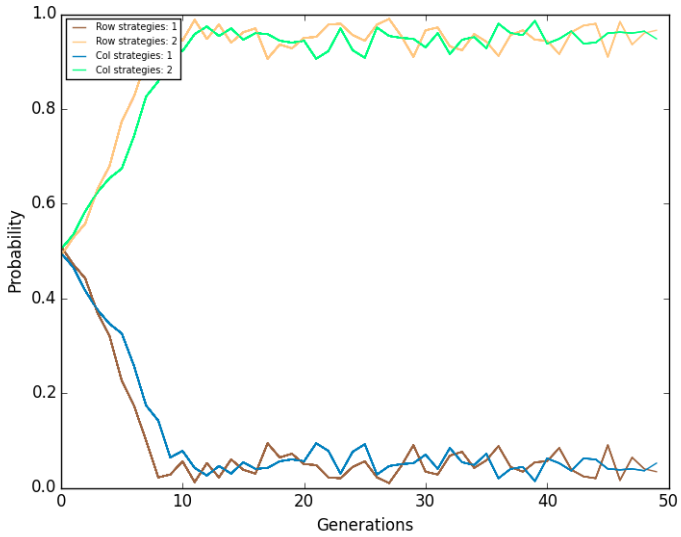

Figure 4.20: Prisoner's Dilemma 25 rounds per generation.

	Cooperate	Defect
Row Agents	0.02	0.98
Column Agents	0.028	0.972

Figure 4.20: Generation 50 Prisoners Dilemma distribution for this simulation 3.

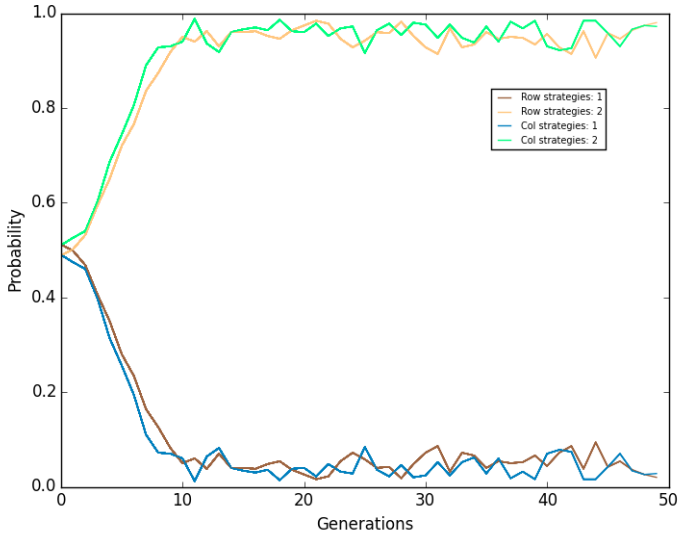

Figure 4.21: Prisoner's Dilemma 5 rounds per generation.

Simulation 2, both proportions decreased slightly. Since the changes in the proportions are very small I do not consider there is significant difference between using 50 or 25 rounds. For simulation 3 the value for row agent in strategy 'Defect' is slightly higher compared to simulation 2 and smaller from the one in simulation 1. For column agents for the same strategy the value is higher in comparison to the values from simulations 1 and 2.

After the 3 runs for the simulations, we can see in the graphs from each simulation that they all have a similar pattern. And after analysing the different specific points (generations), the variations between them do not appear to be caused by the number of rounds. The strategies through the generation have a similar behaviour, the strategy 'Defect' for row and column agents dominate the strategy 'Cooperate'. The strategy 'Cooperate' in the 3 different simulations starts declining from as early as the second generation, and around generation 10 it reaches the lowest values continuing to oscillate between 0.02 and 0.06 for the rest of the simulation. Again it is important to mention that the length of the simulations in terms of generations do not appear to be very relevant, since we can see that the behaviour for both strategies is very similar from generation 10 to generation 50 (when the simulation ends). On the other hand, we can see that the behaviour for the strategies is cyclical, with not a very clear pattern. For this reason in prisoner's dilemma simulations the configuration with 5 rounds per generation will used. When there is a small increment in the 'Cooperate' or 'Defect' is given to the mutation rate, that is allowing more of either to appear, but since it is very low (0.1) it does not change the behaviour significantly. This is how the graphs behave without mutation rate.

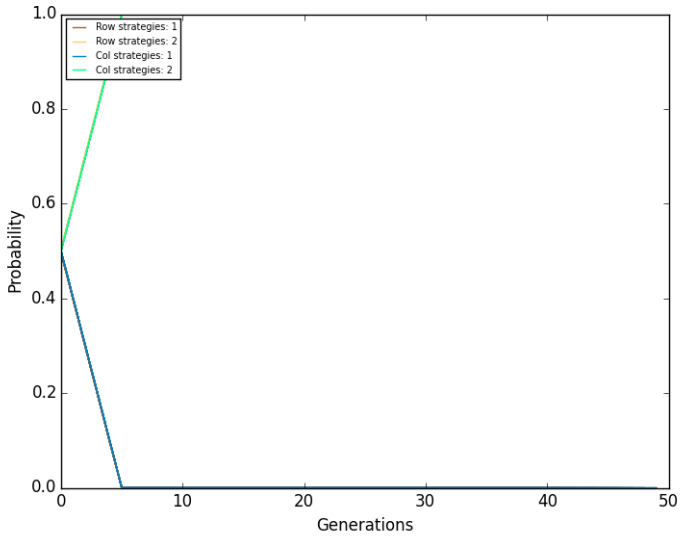

Figure 4.22: Prisoner's Dilemma 50 rounds per generation mutation rate 0.

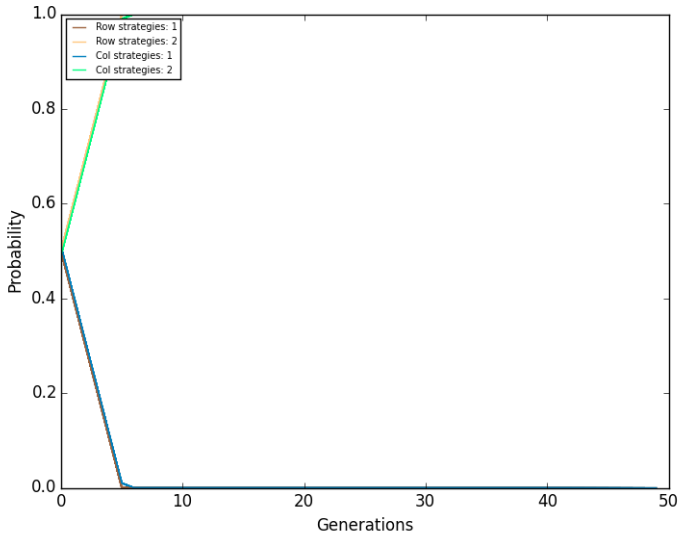

Figure 4.23: Prisoner's Dilemma 25 rounds per generation mutation rate 0.

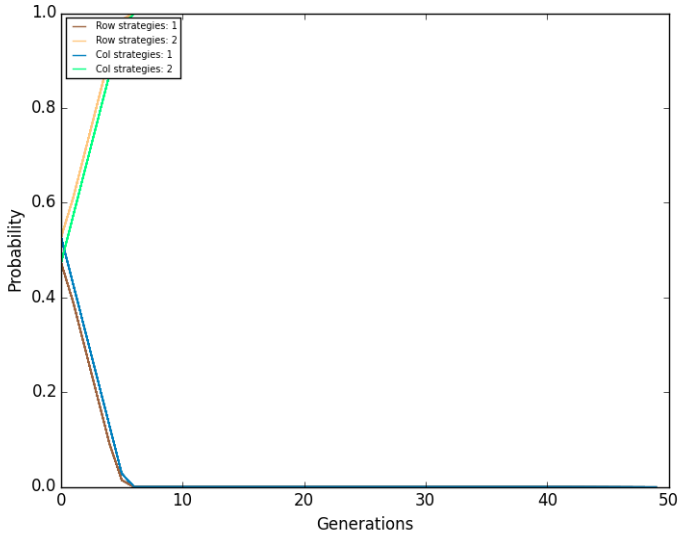

Figure 4.24: Prisoner's Dilemma 5 rounds per generation mutation rate 0.

These graphs show how mutation rate allows the less effective strategy 'Cooperate' to be present throughout the generations, and without mutation they will just disappear as early as the 5th generation. Let us continue to try the different combinations of values that are shown in the table 4.5 (Configuration of simulations).

5 Results

"Prisoner's Dilemma" For simulations 3, 4 and 5 we have constant values for mutation rate with 0.1 and exploitation rate of 1. The resulting graphs from simulations 3, 4, and 5 are presented. It is important to note that in these first 3 simulations the value that varies is death rate, and that the smaller the death rate, the gap between strategies 'Defect' and 'Cooperate' is larger.

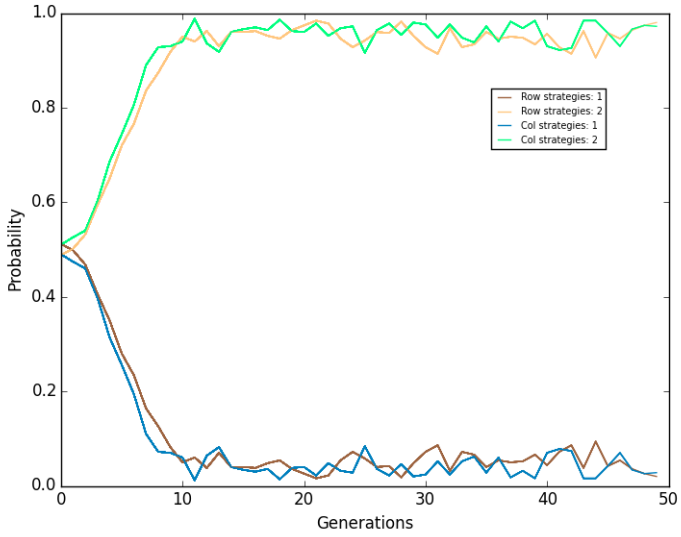

Figure 5.1: Prisoner's Dilemma simulation 3.

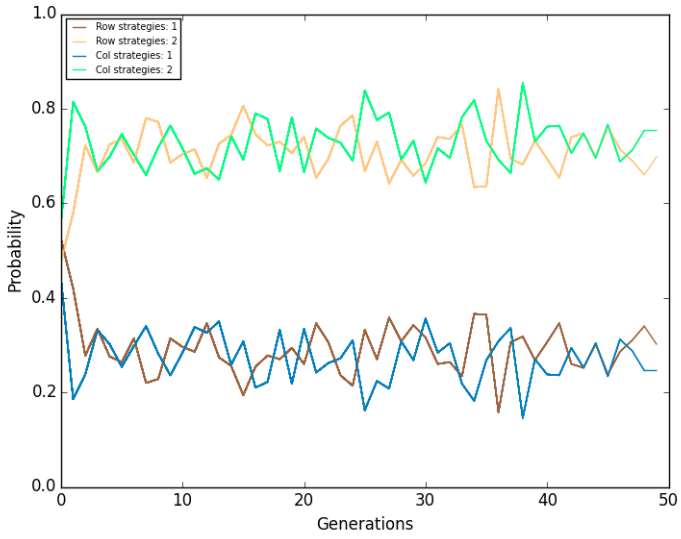

Figure 5.2: Prisoner's Dilemma simulation 4.

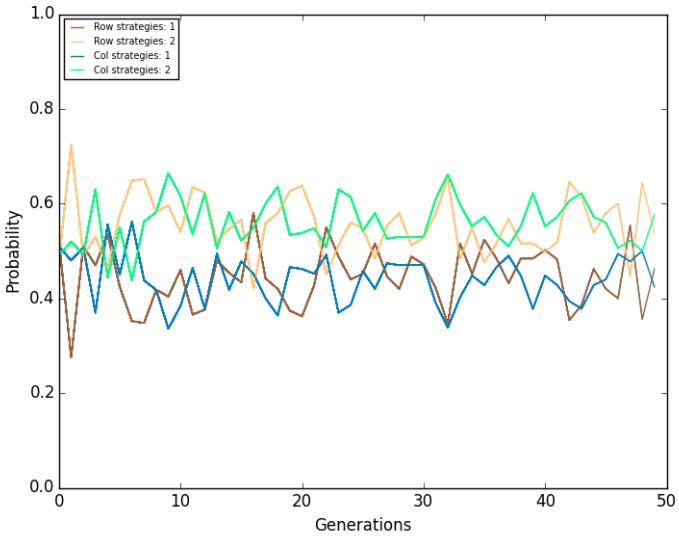

Figure 5.3: Prisoner's Dilemma simulation 5.

Graphs 5, 6 and 7 have a high death rate of 0.9, this means that it allows the creation off 90% of new agents each generation while eliminating the least efficient. These three graphs have mutation rates of 0.1, 0.5 and 0.9 respectively with an exploitation rate of 1. And the behaviour can be seen in the following graphs.

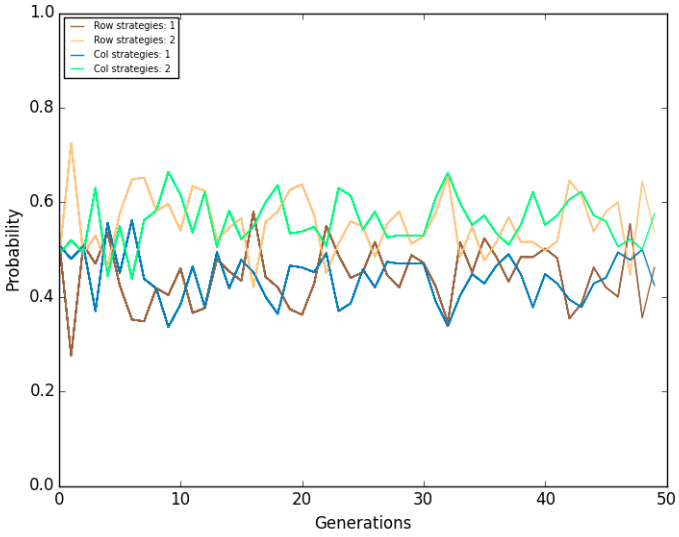

Figure 5.4: Prisoner's Dilemma simulation 5.

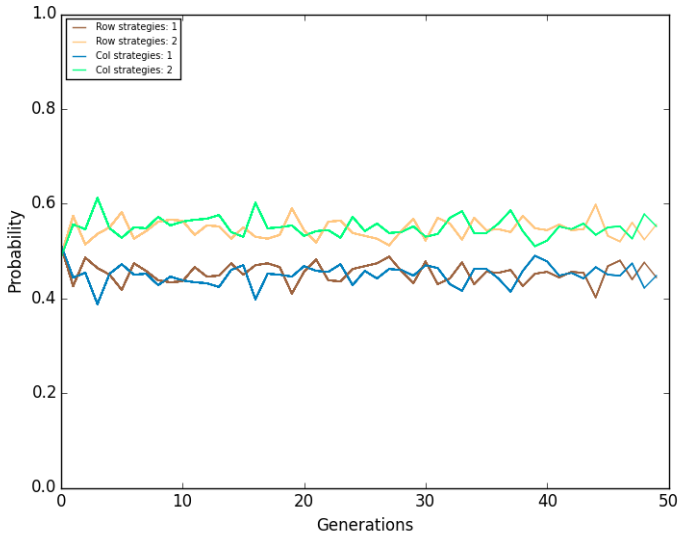

Figure 5.5: Prisoner's Dilemma simulation 6.

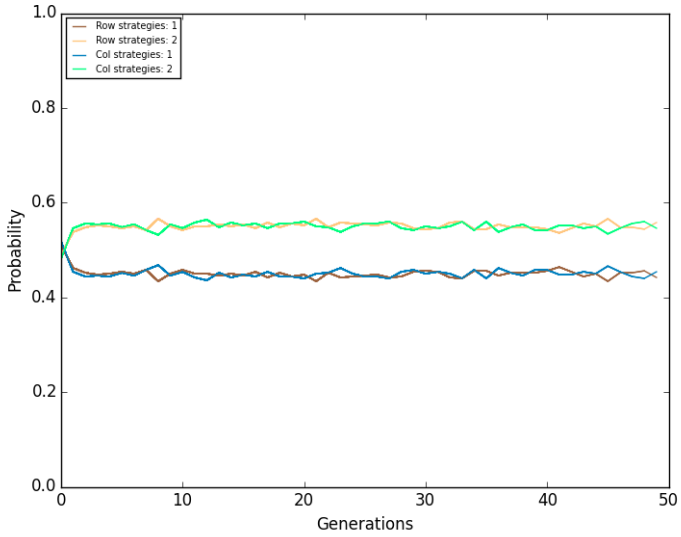

Figure 5.6: Prisoner's Dilemma simulation 7.

It can be seen that when the mutation rate is lower the variation is higher, even if the proportions are close to 0.5. There are evident high peaks and low valleys for both strategies. In the three simulations the strategy 'Defect' dominates in general but in simulation 5 for generations 5, 7, 17, 23, 27, 34, 36 and 48, the strategy 'Cooperate' briefly dominates 'Defect' in fact it only dominates in this generations in the previous of next generation the values are lower. This brief dominance from 'Cooperate' may be given to a higher interaction between this type of strategy between each other, in consequence they accumulate a higher utility which allows them to reproduce in the following generation. Because of the high death rate they reproduce quickly. But this means that in the following generation there is a higher probability for 'Defect' to encounter with 'Cooperate', so 'Defect' accumulates more utility and 'Cooperate' quickly dies out. In the other 2 simulations we see how as the mutation level increases the proportion of the strategies become more stable. By looking

at the graphs 1, 8 and 9 have a similar behaviour where as the mutation rate increases, less peaks and valleys can be seen.

For graphs 10, 11 and 12 we have an incrementing death rate of 0.1, 0.5 and 0.9 respectively. For all 3 simulations there are constant values for mutation rate of 0.1 and exploitation rate of 0.5. As the death rate increases, as expected the gap between the number of strategies representing 'Defect' and 'Cooperate' reduces. But it can also be seen that as the as death rate increases given the value of 0.5 for exploitation rate the peaks and valleys increase, they do not get to the point where 'Cooperate' dominates 'Defect', but it is important to note that with a low mutation rate and a lower exploitation rate than 1 which in these cases is 0.5 strategies that earn lower utility during the interaction are allowed to reproduce in the same probability as the one with highest utility.

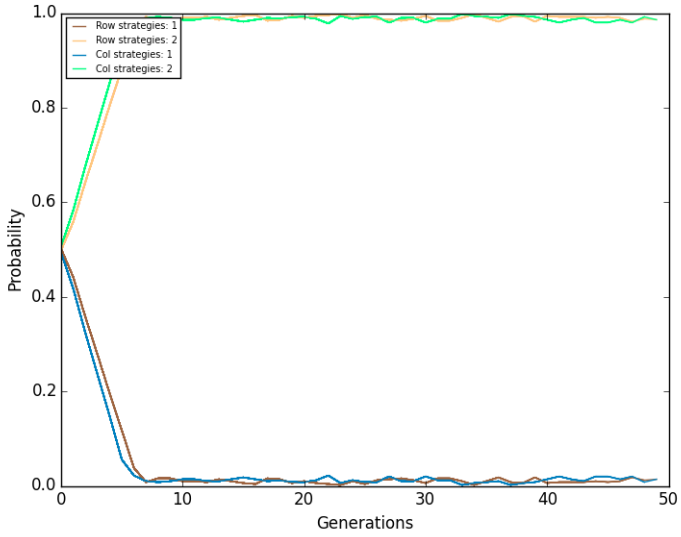

Figure 5.7: Prisoner's Dilemma simulation 10.

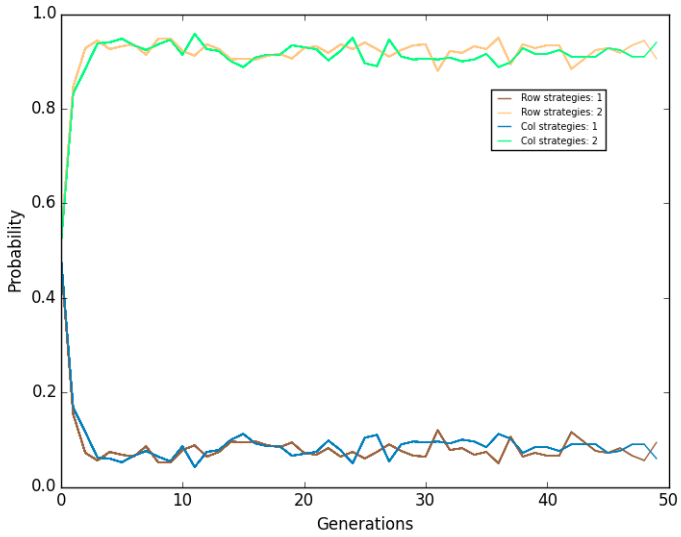

Figure 5.8: Prisoner's Dilemma simulation 11.

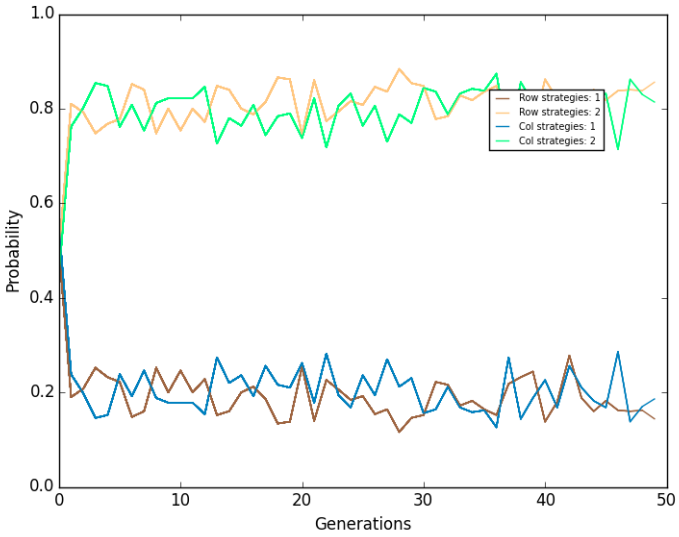

Figure 5.9: Prisoner's Dilemma simulation 12.

When comparing these graphs with the graphs from simulation 3, 4 and 5 which have an increasing value of death rate and a death rate of 0.1 they behave in a similar way. But when comparing graphs from simulation 5 and 12 with each other. They both have a death rate of 0.9 and a mutation rate of 0.1, but a different exploitation rate value of 1 and 0.5 respectively. We can see that since a lower exploitation rate makes other strategies reproduce with the same probability as the strategy with the highest accumulated utility from the generation, a larger gap between strategies is present. Meaning that even if 'Cooperate' accumulates a high utility, if there is 'Defect' strategy within the 50% of the population that is allowed to reproduce, it has a possibility to reproduce. This 'randomization' gives an equilibrium between the presence of strategies in the population. Simulations 19, 20 and 21 have the same values as these other groups of simulations, but an exploitation rate of 0.1, which makes makes graph from simulation 21 behave very similar to the graph from simulation 12.

For the graphs from simulations 16, 17 and 18 the value that changes is death rate 0.1, 0.5 and 0.9 respectively. With constant values of mutation rate with 0.9 and exploitation rate 0.5. As we have seen before the effect of the death rate reflects with a reducing gap between the type of strategies as death rate increases.

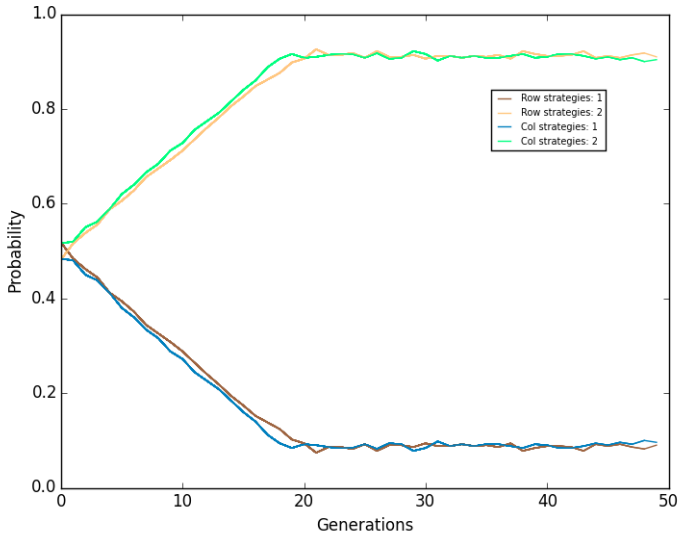

Figure 5.10: Prisoner's Dilemma simulation 16.

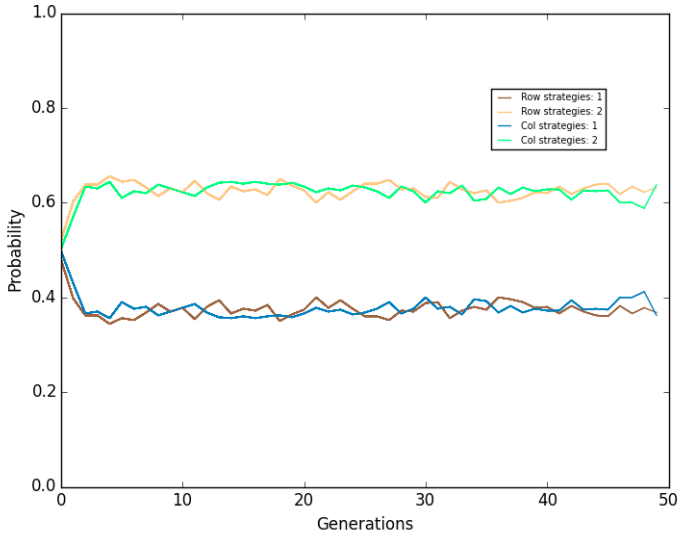

Figure 5.11: Prisoner's Dilemma simulation 17.

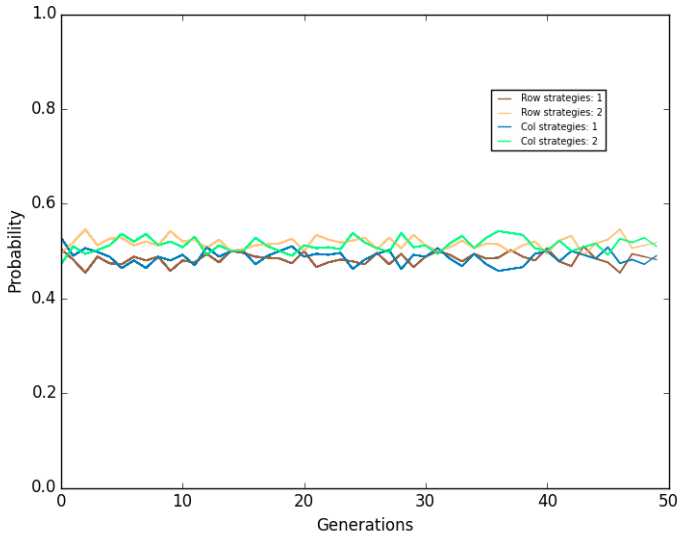

Figure 5.12: Prisoner's Dilemma simulation 18.

But we can see a particular characteristic in the graph for simulation 16, it reaches the lowest value for 'Cooperate' strategy later than other simulations with the same death rate. Graphs from simulations 3, 8, 9, 10, 13, 19 and 22 they all reach the lowest value for 'Cooperate' around generation 10. Simulation 16 reaches it closer to generation 20, we see a similar behaviour for simulation 25. What this two simulations have in common other than death rate value, is their mutation rate is 0.9. Since mutation rate allows any strategy to reproduce except for the agent with strategy with that accumulated the highest utility, this delays the 'Cooperate' strategy for reaching its lowest value. The remaining simulations, behave in a similar way to the one that have been discussed so far. It can be seen that the variable that acts as an enabler for the other variables to have an effect is the death rate.

As we calculated at the before in this work. We can find the Nash Equilibrium for the prisoner's dilemma game, by looking at the normal form table and select the best responses for each player. We can see that the results from the simulations gives us the strategy "Defect" as stable and this is also a Nash equilibrium as we can see in the normal form table for the game.

Player 2

		Cooperate	Defect
Player 1	Cooperate	3, 3	0, 5
	Defect	5, 0	$\underline{1}$, $\underline{1}$

Figure ???.???: Nash equilibrium with best responses.

Now let us find the ESS for the prisoner's dilemma game. Earlier it was explained how to calculate the evolutionarily stable strategy (ESS) when a mutant strategy appears in the population in the proportion ϵ. We will assume the strategy 'Cooperate' is the population and the invader is the strategy 'Defect'. Thus we have that the payoff for the strategy 'Cooperate' is the following:

$$(1-\epsilon)3 + 0\epsilon = 3 - 3\epsilon \tag{6}$$

And the payoff for the strategy 'Defect' is:

$$(1-\epsilon)5 + 1\epsilon = 5 - 4\epsilon \tag{7}$$

We can see that the strategy 'Defect' appears to be greater than 'Cooperate'. Now since our simulation every time gives almost equal starting proportions to each strategy we will assume that the 'invading' strategy 'Defect' appears in a proportion of ϵ=0.5. And we have the following:

$$3 - 3(0.5) = 1.5 \tag{8}$$

And for the strategy 'Defect' is:

$$5 - 4(0.5) = 3 \tag{9}$$

So we have that the payoff for 'Defect' is greater than the payoff for 'Cooperate' thus we can say that 'Cooperate' is not an evolutionarily stable strategy. Now we check if the strategy 'Defect' is evolutionarily stable. And therefore we assume that 'Cooperate is the invader. We can see that it appears that 'Defect' might be evolutionarily stable given the previous test. But we should check this in the same way. We first have the expected payoff of the strategy 'Defect':

$$(1 - \epsilon)1 + 5\epsilon = 1 + 4\epsilon \tag{10}$$

And the payoff for the strategy 'Defect' is:

$$(1 - \epsilon)0 + 3\epsilon = 3\epsilon \tag{11}$$

Again it appears that the strategy 'Defect' has a greater expected payoff than 'Cooperate'. We assume that the 'invading' strategy 'Cooperate' appears in a proportion of ϵ=0.5. And we have the following for the strategy 'Defect':

$$1 + 4(0.5) = 3 \tag{12}$$

And for the strategy 'Cooperate' is:

$$3(0.5) = 1.5 \tag{13}$$

So we see that again the strategy'Defect' has a higher expected payoff, so it is an evolutionarily stable strategy (ESS). With this we can see that the strategy 'Defect' is both a Nash equilibrium and an ESS. And this supports the results we have from the simulation.

"Matching Pennies"

We can find the Nash Equilibrium for the matching pennies game, we cannot calculate it just by looking at the normal form table and selecting the best responses since this game has a mixed Nash equilibrium.

Player 2

Player 1		Cooperate	Defect
	Cooperate	1, -1	-1, 1
	Defect	-1, 1	1, -1

Figure ???.???: Matching pennies bimatrix payoff.

So we calculate the mixed Nash equilibrium by making equal the expected payoff when playing heads to the expected payoff of playing tails. We will assume that the probability for player 1 of playing heads for player one is σ and for playing tails 1 - σ, resulting in the following:

$$(-1)(\sigma) + (1 - \sigma) = (1)(\sigma) + (-1)(1 - \sigma) \tag{14}$$

$$-4\sigma = -2 \tag{15}$$

$$\sigma = 1/2 \tag{16}$$

This means that player 1 will play heads 1/2 of the times, and also means she will be playing tails 1/2 also. And now we calculate the probability of playing heads for player 2 with probability σ and tails with probability 1 - σ.

$$(1)(\sigma) + (1-\sigma)(-1) = (-1)(\sigma) + (1)(1-\sigma) \tag{17}$$

$$4\sigma = 2 \tag{18}$$

$$\sigma = 1/2 \tag{19}$$

This means that player 2 when playing against player 1 will play heads 1/2 of the times and in consequence play tails in the other 1/2 of the times. This means that the Nash equilibria is when:

$$((1/2, 1/2), (1/2, 1/2)) \tag{20}$$

Now let us find the ESS for the matching pennies game. In the same way we calculated the prisoner's dilemma ESS. We will assume the strategy 'Heads' is the population and the invader is the strategy 'Tails'. Thus we have that the payoff for the strategy 'Heads' is the following:

$$(1-\epsilon)(1) + (-1)\epsilon = 1 - 2\epsilon \tag{21}$$

And the payoff for the strategy 'Tails' is:

$$(1-\epsilon)(-1) + (1)\epsilon = 2\epsilon - 1 \tag{22}$$

We can see that both strategies appear to be equivalent unless there is a very small value of ϵ.But since our simulation every time gives almost equal starting proportions to each strategy we will assume that the 'invading' strategy 'Tails' appears in a proportion of ϵ=0.5. And we calculate as follows:

$$1 - 2(0.5) = 0 \tag{23}$$

And for the strategy 'Defect' is:

$$2(0.5) - 1 = 0 \tag{24}$$

So we have that the expected payoff for both is 0 thus we can say that 'Heads' is not an evolutionarily stable strategy, Given that it does not dominate 'Tails'. Now we check if the strategy 'Tails' is evolutionarily stable. And therefore we assume that 'Heads' is the invader. We will check the same way. We first have the expected payoff of the strategy 'Tails':

$$(1-\epsilon)(-1) + (1)(\epsilon) = 2\epsilon - 1 \tag{25}$$

And the payoff for the strategy 'Heads' is:

$$(1-\epsilon)(1) + (-1)(\epsilon) = 1 - 2\epsilon \tag{26}$$

Again it appears that they are equal. So we can see that 'Tails' is not an ESS either. But we calculate it anyway assuming that the 'invading' strategy 'Heads' appears in a proportion of ϵ=0.5. And we have the following for the strategy 'Tails':

$$1 - 2(0.5) = 0 \tag{27}$$

And for the strategy 'Heads' is:

$$2(0.5) - 1 = 0 \tag{28}$$

So we see that again the strategy 'Tails' is not an ESS either. And this explains the behaviour in our graph, how through time both strategies appear to interact and oscilate in the middle (0.5)

References

[1] Joel Watson. *Strategy: An introduction to game theory*. 2013.

[2] Robert Gibbons. *A primer in game theory*. Harvester Wheatsheaf, 1992.

[3] Nolan McCarty and Adam Meirowitz. *Political game theory: an introduction*. Cambridge University Press, 2007.

[4] Amnon Rapoport. Game theory: contributions to the study of human cognition. , 6(2):142–167, 1999.

[5] Andrew M Colman. Cooperation, psychological game theory, and limitations of rationality in social interaction. *Behavioral and brain sciences*, 26(02):139–153, 2003.

[6] M Hykšová. Several milestones in the history of game theory. *inJubiläenChance oder Plage*, pages 49–56, 2004.

[7] Emile Borel. La théorie du jeu et les équations intégrales à noyau symétrique. *Comptes Rendus de lAcadémie des Sciences*, 173(1304-1308):58, 1921.

[8] Emile Borel. On games that involve chance and the skill of the players. *Econometrica: Journal of the Econometric Society*, pages 101–115, 1953.

[9] J von Neumann. On the theory of parlor games. *Mathematische Annalen*, 100:295–320, 1928.

[10] John Von Neumann and Oskar Morgenstern. *Theory of games and economic behavior*. Princeton university press, 2007.

[11] John Nash. Non-cooperative games. *Annals of mathematics*, pages 286–295, 1951.

[12] Charles Darwin and Encyclopaedia Britannica. The origin of species by means of natural selection. 1872.

[13] Mark Pallen. *The rough guide to evolution*. Dorling Kindersley Ltd, 2009.

[14] Karl Sigmund. John maynard smith and evolutionary game theory. *Journal of theoretical biology*, 68:7–10, 2004.

[15] J Maynard Smith. Evolution and the theory of games: in situations characterized by conflict of interest, the best strategy to adopt depends on what others are doing. *American Scientist*, pages 41–45, 1976.

[16] John M McNamara and Franz J Weissing. Evolutionary game theory. *Social behaviour: Genes, ecology and evolution*, pages 109–33, 2010.

[17] Fiona Carmichael. *A guide to game theory*. Pearson Education, 2005.

[18] Thomas L Vincent and Joel S Brown. *Evolutionary game theory, natural selection, and Darwinian dynamics*. Cambridge University Press, 2005.

[19] J Maynard Smith and GR Price. The logic of animal conflict. *Nature*, 246:15, 1973.

[20] Larry Samuelson. *Evolutionary games and equilibrium selection*, volume 1. Mit Press, 1998.

[21] Alexander J McKenzie and Edward N Zalta. Evolutionary game theory. 2009.

[22] David Easley and Jon Kleinberg. *Networks, crowds, and markets: Reasoning about a highly connected world*. Cambridge University Press, 2010.

[23] Martin J Osborne. *An introduction to game theory*, volume 3. Oxford University Press New York, 2004.

[24] Uri Wilensky and William Rand. *An Introduction to Agent-Based Modeling: Modeling Natural, Social, and Engineered Complex Systems with NetLogo*. MIT Press, 2015.

[25] Marco A Janssen and Elinor Ostrom. Empirically based, agent-based models. *Ecology and Society*, 11(2):37, 2006.

[26] Paul Rendell. A turing machine in conway's game life. *http://www. cs. ualberta. ca/˜ bulitko F*, 2, 2001.

[27] J Maynard Smith. The theory of games and the evolution of animal conflicts. *Journal of theoretical biology*, 47(1):209–221, 1974.

[28] Rober Axelrod. *The Evolution of Cooperation*. Basic Books, Inc., Publishers New York, 1984.

[29] Robert M Axelrod. *The complexity of cooperation: Agent-based models of competition and collaboration*. Princeton University Press, 1997.

[30] Joshua M Epstein. Agent-based computational models and generative social science [generative social science studies in agent-based computational modeling]. *Introductory Chapters*, 2007.

[31] Charlotte Bruun. Agent-based computational economics-an introduction. *Dept of Economics, Politics and Public Administration, Aaiborg University, Denmark*, 2004.

[32] Bhanu Prasad Pokkunuri. Object oriented programming. *ACM Sigplan Notices*, 24(11):96–101, 1989.

[33] Richard P. Ten Dyke and John C. Kunz. Experiences with object-oriented group support software development. *IBM Systems Journal*, 28(3):465–478, 1989.

[34] Francesco Luna and Benedikt Stefansson. *Economic Simulations in Swarm: Agent-based modelling and object oriented programming*, volume 14. Springer Science & Business Media, 2012.

[35] Margaret Ann Boden. *Mind as machine: A history of cognitive science*. Oxford University Press, 2006.

[36] Dan Simon. *Evolutionary optimization algorithms*. John Wiley & Sons, 2013.

[37] David E Golberg. Genetic algorithms in search, optimization, and machine learning. *Addion wesley*, 1989, 1989.

[38] Mitchell Melanie. An introduction to genetic algorithms. *Cambridge, Massachusetts London, England, Fifth printing*, 3, 1999.

[39] James G March. Exploration and exploitation in organizational learning. *Organization science*, 2(1):71–87, 1991.

[40] Kent Beck. *Test-driven development: by example*. Addison-Wesley Professional, 2003.

[41] Guido Van Rossum and Fred L Drake. *Python language reference manual*. Network Theory, 2003.

[42] Scott Chacon. *Pro git*. Apress, 2009.