University Institute of Lisbon

Department of Information Science and Technology

Using Genetic Algorithms for Real-time Dynamic Difficulty Adjustment in Games

João David Oliveira Pereira

A Dissertation presented in partial fulfillment of the Requirements
for the Degree of
Master in Computer Science

Supervisor
Prof. Dr. Sancho Moura Oliveira, Assistent professor
ISCTE-IUL

October 2019

"I was obsessed with the idea of sitting next to someone and playing a game that we were both competing in, and we were also competing with the computer. That was mind-blowing to me at that time. It was just so cool to think about the computer being able to play with us, and then also [for] us to compete."

Robin Hunicke

Abstract

Dynamic Difficulty Adjustment is the area of research that seeks ways to balance game difficulty with challenge, making it an engaging experience for all types of players, from novice to veteran, without making it frustrating or boring.

In this dissertation we propose an approach that aims to evolve agents, in this case predators, as a group and in real time, in a way that they adapt to a changing environment.

We showcase our approach after using a generic genetic algorithm in two scenarios, pitting the predators vs passive prey in one scenario and pitting the predators vs aggressive prey in another, this is done to create a basis for our approach and then test our algorithm in four different scenarios, the first two are the same as the generic genetic algorithm and in the next two we switch prey in the middle of the experience progressively from passive to aggressive or vice versa.

Keywords: Game Development, Dynamic Difficulty Adjustment, Genetic Algorithms, NEAT.

Resumo

Adaptação Dinâmica de Dificuldade é a área de pesquisa que procura formas de equilibrar a dificuldade do jogo com o desafio, tornando-o uma experiência envolvente para todos os tipos de jogadores, desde principiantes a veteranos, sem o tornar frustrante ou aborrecido.

Nesta dissertação propomos uma abordagem que visa evoluir os agentes, neste caso predadores, como um grupo e em tempo real, de forma a que estes se adaptem a um ambiente em mudança.

Nós mostramos a nossa abordagem depois de usar um algoritmo genético genérico em dois cenários, colocando os predadores versus presas passivas num cenário e colocando os predadores versus presas agressivas noutro, isto é feito para criar uma base para a nossa abordagem e depois testamos o nosso algoritmo em quatro cenários diferentes, os dois primeiros são os mesmos que o algoritmo genético genérico e nos dois seguintes trocamos as presas a meio da experiência progressivamente de passivas para agressivas ou vice-versa.

Palavras-chave: Desenvolvimento de jogos, Adaptação Dinâmica de Dificuldade, Algoritmos Genéticos, NEAT.

Acknowledgements

Quero agradecer ao Instituto de Telecomunicações por ter emprestado o espaço para o desenvolvimento desta tese.

Quero agraceder ao Professor Sancho Oliveira por me ter orientado em todos os passos deste longo caminho que é redigir uma tese, por ter tido interesse no tema, pela paciência, pelas critícas construivas e muito mais, certamente que sem ele esta tese não teria sido redigida.

Quero também agradecer ao grupo do ZooISCTE/Xaboitec aos quais pertecem, Bernardo Ribeiro, João Bernardo, Kevin Ramos, Rodrigo Almeida e Robbert DeHaven, pelos momentos partilhados, pelo apoio e motivação, pelas horas a fio a discutir o paradoxo de Fermi, canabalismo e outros tópicos menos próprios para um documento deste tipo.

Quero agradecer ao pessoal que me aturou constantemente as reclamações, as lamúrias e afins, que também me apoiou e me motivou a continuar com esta momumental tarefa, que em alguns momentos mais parecia uma representação realista da jornada de Dante na Divina Comédia, nomeadamente a primera parte. De entre essas pessoas gostaria de destacar Patrícia Santos, Luís Antunes e Pedro Gonçalves.

Quero por fim também agradecer à minha família pelo apoio, confiança, compreensão, paciência, amor e carinho.

A todos um muito obrigado por tudo.

Contents

List of Figures

Abbreviations

AI Artificial Intelligence

NPC Non-player Character

FPS First Person Shooter

TPS Third Person Shooter

NEAT NeuroEvolution of Augmenting Topologies

GA Genetic Algorithms

PCG Procedural Content Generation

TWEANN Topology and Weight Evolving Artificial Neural Network algorithms

rtNEAT Real-time NeuroEvolution of Augmenting Topologies

cgNEAT Content-Generating NeuroEvoltion of Augmenting Topologies

odNEAT Online and Decentralized NeuroEvolution of Augmenting Topologies

DDA Dynamic Difficulty Adjustment

Chapter 1

Introduction

The video game industry is one of the most profitable within the entertainment industries, having invoiced 121.7 billion dollars in 2017 and it is estimated that by 2021 it will reach 180 billion dollars [Newzoo, 2018], according to this there is an increasing demand from the public for this kind of entertainment [Newzoo, 2018].

There is a growing competition amongst companies to get attention to their titles from the consumer for as long as possible. With this objective in mind, different types of approaches were developed for the creation of games and content [Hendrix et al., 2018]. This growing innovation in a competitive market leads to the need to create content, faster and at a lower cost, without compromising its capacity to retain the players' interest. This need led to the use and creation of different approaches such as but not limited to, the content generated procedurally and adaptive content [Hendrix et al., 2018].

This thesis focuses on the interactions between non-playable characters (NPC's), controlled by the system, and the players. In particular in the evolution of this relation, where, while the players are learning how to play, the game collects information about the player's behaviour. This information may then be used to help the game to learn from the gameplay of the user and thereby adapt and improve the players' experience. This kind of approach to game creation is not limited to one type of game (First Person Shooter (FPS), Third Person Shooter

(TPS), strategy, among others [Apperley, 2006, Bontrager et al., 2016]). The idea of the game progressively adapting to the player to improve the rhythm and experience is a simple idea that has existed for many years, having first appeared in the game 'Gun Fight' in 1975 [Capcom, 1975]. This mechanic is called 'Dynamic game Difficulty Balancing' (or dynamic difficulty adjustment) and is a procedural way to evolve the game that aims to subtle and progressively change some parameters according to the ability of the player [Hunicke and Chapman, 2004], such as slowing down obstacles, giving extra life to the player, adjusting the maximum life of enemies [Gavin, 1996], or even the very way in which the events of the game are "narrated". An example of this last adjustment is the game 'Left4Dead' [Valve, 2008] by Valve, which uses an artificial intelligence (AI) called "The AI Director" that controls the way and time events occur ('Procedural Narrative') [Magazine, 2008]. From the players point of view it's less appealing to encounter enemies or obstacles that are always the same than to encounter enemies or obstacles that are slightly different, because if these things are always the same, it becomes a chore, not a challenge, and has a consequence it starts to get easy and boring [Lach, 2018]. This difference in design, forces the player to change their way of playing to progress. Such dynamic adaptation of the game allows the experience to be more immersive and challenging, or easier depending on the success that the player exhibits. This adaptive game flow is not a unique solution to the creation of games, due to the existence of a wide range of genres [Apperley, 2006] [Bontrager et al., 2016], platforms [Cowan and Kapralos, 2014] and also audiences [Llc, 2019]. An Example of this non uniqueness is the fact that for some players a static progression like the one found in the Soul games (Demon Souls, Dark souls 1, 2, 3 and Bloodborne [FromSoftware, 2018]) is more appealing, instead of games that are guided by narrative, like Journey [Thatgamecompany, 2012], or even games that present adaptive difficulty, like Left4Dead mentioned above and Alien Isolation [Assembly, 2014], where the AI presented in the alien takes centre place in how the game progress.

1.1 Objectives and research questions

The main objective of this thesis is to leverage genetic algorithms and concepts taken from NeuroEvolution of Augmenting Topologies (NEAT) to dynamically adjust the difficulty of games. More precisely, we will use these algorithms and concepts for the adaptation of creature behaviour in real-time based on players' actions to adapt and evolve said behavioural patterns so that the game tailors itself around the expertise of the player. This evolution is performed on the creatures' parameters that in turn affect its behaviour tree. In this case, we are going to use Unreal Engine 4 to create a test environment to test and evolve predator creatures according to environmental inputs. In this case the only environmental variables are the prey creatures. For this, we make use of Genetic Algorithms to evolve predator creatures first in a classical approach, save the results and then use a novel genetic algorithm approach to evolve the predator creatures and adapt the population continuously in real-time.

1.2 Structure and Organization

The remaining of this thesis is organized as follows: In Section 2 we review work previously done in topics related to this work, we then proceed in Section 3 to discuss how to achieve what we propose, that is followed by Section 4 where we present results of each experiment, on Section 5 we discuss the obtained results, in Section 6 we present some post review tests and discuss them and on Section 7 we draw conclusions and talk about future work.

Chapter 2

Related Work

In this section, we review and explain work previously done in areas of interest to this thesis. There are a couple of methods that can be used to approach this topic [Hendrix et al., 2018, Lach, 2018] such has: methods that rely on prior human knowledge and experience [Malla Osman et al., 2015, Magerko et al., 2006], Player Experience Models, Procedural Content Generation and Genetic Algorithms.

2.1 Player Experience Models

Player Experience Models is the field of study that deals with computation models of players applied to games. These models use information displayed through subconscious actions and behaviours [Yannakakis et al., 2013]. Player Experience Models are mainly used to monitor and collect information about the players' performance. By doing this, the game's difficulty will be tweaked to the player, according to what is shown in their gameplay data, instead of the user or playtesting [Hendrix et al., 2018].

Related work presented in [Hendrix et al., 2018] shows that various methods have been used to achieve this, such as adding or removing enemies and items [Hunicke and Chapman, 2004], adjusting the behaviour of the AI [Andrade et al., 2006]

and using case-based reasoning [Bakkes et al., 2009]. It is also stated in [Hendrix et al., 2018] that "... there is no standardized experience model that can be integrated seamlessly, mainly due to the large challenge of creating a one-size-fits-all mode given the number of different genres, platforms and technologies in use.", this claim that there are plenty variables to this problem is also supported by [Apperley, 2006].

Has stated in [Apperley, 2006] there are 3 main ways to approach player modelling, Model-based, Model-free and Hybrid.

In a Model-based approach researchers build a model based on a theoretical framework, where you first create a theory then test it to verify if it answers the problem. This type of approach may refer to different methods of analysis such as but not limited too emotional models, cognitive modelling.

In a Model-free approach instead of proposing an hypothesis first, researchers collect and analyse data first then generate model according to that data. In this type of approach, we see the use of machine learning procedures to generate the models.

A hybrid approach is an approach that is not a model-base approach and not a model-free approach, it's an approach that is situated "... between the two ends of the spectrum, containing elements of both approaches." [Yannakakis et al., 2013].

2.2 Procedural Content Generation

Procedural Content Generation (PCG) is a method of generating content procedurally from a reduced amount of resources, combining them randomly. As stated in [Hendrix et al., 2018] this method has two main types: offline and real-time. The offline variant refers to when content is generated to aid development and then it can be adapted or used as is by the developer, this is mainly used to save development time and costs. The real-time variant refers to when the game is being played. This way of generating content is mainly used to increase randomness, unpredictability and increase the re-playability factor of the game.

According to [Hendrikx et al., 2013], six levels of content can be procedurally generated, which are: game bits, game space, game systems, scenarios, game design and derived content. Game bits are the base materials which the a game world is built upon for example textures and sounds. Game space is the realm in which the game happens, for example, maps and dungeons. Game systems are ways to complement the world to make it more appealing, for example, ecosystems and roads. Scenarios are ways through which the game progresses for example puzzles and the story. Game design is the constrains set to the game and it can also include content from all six levels described here, examples of this are system design and world design. Derived content is byproduct content created by the game, examples of this are leaderboards and news.

2.3 Genetic Algorithms

Genetic Algorithms are a type of Evolutionary Algorithm and therefore belong to the group of Search Algorithms [Vikhar, 2016]. GAs were first proposed in 1970 by John Henry Holland in [Holland, 1970] and later expanded by David E. Goldberg in [Goldberg, 1989]. This algorithm works in a way that resembles the processes of natural selection described by Charles Darwin, wherein simpler terms individuals better suited for their environment have more chances to survive and therefore reproduce and pass on their genes. The aspects in which this evolutionary process resemble natural selection are selection, reproduction, crossover and mutation [Vikhar, 2016].

These algorithms are mainly used to solve optimization and search problems and are used in various fields of study [Freeman, 1998]. The fields we are most interested in for this thesis are robotics and games, mainly the NEAT and Dynamic Difficulty Balance methods.

2.3.1 Selection

Selection is the process of choosing the most suited individuals according to their proficiency to perform a task, this selection can be done in a couple of ways such has Roulette selection, Tournament selection, Stochastic universal sampling, Reward-based selection and Truncation selection [Jinghui et al., 2005, Baker, 1987, Loshchilov et al., 2011, Mühlenbein and Schlierkamp-Voosen, 1993, Deb et al., 2002].

Roulette selection functions in the same way a roulette wheel works, the only difference is that the more fitness an individual has the bigger the partition of the wheel he gets. Meaning if the sum of all the partitions is the accumulated fitness of the population then the partition an individual has correlates to a percentage of the total fitness. The way this selection works is: you need to calculate the total fitness your population obtained then calculate the relative fitness each individual obtained and assign a portion of the roulette according to the percentage of the total fitness achieved, then you spin the roulette and the selected individual is the one where the pointer lands, repeat the spins enough times to repopulate the population until the initial population size is reached [Jinghui et al., 2005].

Tournament selection functions similarly to a sports group tournament, where the one with more points wins, in this case, the individual with more fitness is selected. The way this selection works is you first randomly select several individuals according to the tournament size then compare the fitness of each participant and the one with more fitness is selected, repeat this process enough times to repopulate until the initial population size is reached [Jinghui et al., 2005].

Stochastic universal sampling functions similarly to roulette selection, it still partitions a roulette according to the portions of each individual but instead of spinning randomly it's gonna spin a fixed amount that is obtained by dividing the total fitness for the amount of individuals to repopulate which for clarity purposes we are going to call X, also the starting point for the spins is a random number obtained from 0 to X, the selected individual is where the pointer lands and also

the following starting point is also where the pointer lands, keep spinning until the initial population size is reached [Baker, 1987].

Reward-based selection is a multi-criteria form of selection that is particularly useful to solve the multi-armed bandit paradigm and also to optimize the approximation of the Pareto front. In this selection, the odds of an individual being selected to reproduce is the cumulative of the fitness earned and the rewards earned by its parents. This selection works in the following way: whenever an individual is selected he and his parents will receive a rewards and so on, there is a couple of ways in which this reward can be defined, for more detailed information consult [Loshchilov et al., 2011, Deb et al., 2002].

In the Truncation selection, the best x individuals are selected to mate with each other randomly until the number of children reaches the initial population size, while also preserving the best individual and making sure that the same parent does not mate with himself [Mühlenbein and Schlierkamp-Voosen, 1993].

2.3.2 Reproduction

After selection comes reproduction, where the selected individuals pass on their genes to the next generation. Not all genes are passed from parents to child, in the reproduction phase there are two of operations that chooses which genes are passed on and if they change or not, which are crossover and mutation.

2.3.2.1 Crossover

In the crossover process, genes from the parents can be selected according to one technique, the more common ones being single-point, 2-point, k-point, variable to variable and uniform crossovers [Srinivas M. & Patnaik LM, 2006, Hasançebi and Erbatur, 2000] The crossover happens according to an odd specified by the users which is also called the probability of crossover [Srinivas M. & Patnaik LM, 2006].

Single-point crossover is the simplest crossover method, where it gets the two parents sets of genes and selects a point along the genes to cut, after that point the genes from the parents are swapped [Srinivas M. & Patnaik LM, 2006, Hasançebi and Erbatur, 2000].

2-point crossover is the same has a single-point crossover, except instead of selecting one point it selects two to cut, and swaps either the inner part (between the cuts), or the outer part [Hasançebi and Erbatur, 2000].

K-point crossover is still the same as the previous two crossover processes except the genes are split in k parts and then each of these parts is assigned to a group, according to their order, like 1st, 3rd, etc, and 2nd, 4th, etc, and finally one of the two groups is swapped [Hasançebi and Erbatur, 2000].

In the Variable to variable crossover method, the genes are paired one on one and then a single-point crossover is applied [Hasançebi and Erbatur, 2000].

in the Uniform crossover it's created a crossover mask that stipulates which part of the gene from which parent is passed on to the next generation, the following child to be made is made either with the original mask or with a new mask [Hasançebi and Erbatur, 2000].

2.3.2.2 Mutation

After crossover, the mutation process happens and one or more genes may change depending on the method used, examples of methods used to mutate genes are: Flip Bit, Boundary, Uniform and Gaussian [Goldberg, 1989, Rajakumar, 2013]. The mutation also happens according to an odd specified by the user.

Flip bit mutation is used in bit sequences and it changes the selected genes bit from 1 to 0 or vice versa [Goldberg, 1989].

Boundary mutation is used in either integer or float variables and this operation changes the selected genes to either minimum or maximum limits of the solution space [Rajakumar, 2013].

Uniform mutation is only used in either integer or float variables and this operation changes the value of the gene to a random value between the minimum and maximum values of the solution space.[Rajakumar, 2013]

Gaussian mutation is used only used in either integer or float variables and this operation changes the value of the gene to a random Gaussian distributed value [Rajakumar, 2013].

2.3.3 NeuroEvolution of Augmenting Topologies

NeuroEvolutiion of Augmenting Topologies (NEAT) was first presented in 2002 by Kenneth Stanley and Risto Miikkulainen in [Stanley and Miikkulainen, 2002] and its purpose was to solve problems common to all Topology and Weight Evolving Artificial Neural Network algorithms (TWEANN), "(1) Is there a genetic representation that allows disparate topologies to cross over in a meaningful way? (2) How can topological innovation that needs a few generations to be optimized be protected so that it does not disappear from the population prematurely? (3) How can topologies be minimized throughout evolution without the need for a specially contrived fitness function that measures complexity?" [Stanley and Miikkulainen, 2002]. The first questions answer is the tracking of genes through historical markings, with this all the genes and connections are mapped and kept in a list, the genes then can be matched and paired up to be crossed over meaningfully. The answer to the second question is protecting innovation through speciation, with this an organism instead of competing directly with the whole population, it competes with similar organisms, organisms with the same topology / of the same species, this way the organisms optimize themselves by competing in a niche. This grouping is done in each generation and each species is represented by a random organism from that species and from the previous generation. It's also worth mentioning that the NEAT uses an explicit fitness sharing method for reproduction, because of this the number of elements in each species cannot grow too large and take over the population. Finally the answer to the third question is to minimize dimensionality

through incremental growth from minimal structure, this means that NEAT instead of starting the initial population with random topologies, it starts off with a uniform one and with minimal topologies, has mutations occur and the population evolves, only the fittest solutions thrive and because the population starts with this way then the research space is also minimal which gives NEAT a performance edge while compared to other methods used [Stanley and Miikkulainen, 2002]. There are also a couple of extensions worth mentioning such has rtNEAT, cgNEAT and odNEAT.

2.3.3.1 rtNEAT

rtNEAT stands for real-time NeuroEvolution of Augmenting Topologies, it is an extension of NEAT that conducts evolution in a real-time fashion instead of generation by generation, this method of evolution was used in the game NERO [Stanley et al., 2005] and later in another one called Globulation 2 [Olesen et al., 2008].

2.3.3.2 cgNEAT

cgNEAT stands for Content-Generating NeuroEvoltion of Augmenting Topologies, it is an extension of NEAT specialized in the creation of tailored game content according to user preferences. It was used in the game called Galactic Arms Race Video Game to evolve the particle system of weapons according to their usage [Hastings et al., 2009].

2.3.3.3 odNEAT

odNEAT stands for online and decentralized NeuroEvolution of Augmenting Topologies, it is an extension of NEAT created to evolve multi-robot systems. It is used in robots to make them adapt to environment changes and to learn new behaviours while performing the assigned tasks [Silva et al., 2015].

2.4 Dynamic Difficulty Adjustment

Dynamic Difficulty Adjustment (DDA) or Dynamic Game Balancing is the process of making automated changes to the games' environment, attributes or even behaviours in real-time, according to the players' skill in order to secure the players' attention for as long as possible without making the game too easy, hence boring, or too difficult, hence frustrating. "As players work with a game, their scores should reflect steady improvement. Beginners should be able to make some progress, intermediate people should get intermediate scores, and experienced players should get high scores ... Ideally, the progression is automatic; players start at the beginner's level and the advanced features are brought in as the computer recognizes proficient play." by Chris Crawford embodies what DDA wants to achieve. DDAs main premise is to use players direct and indirect inputs to tailor the game to the user, this said there are a number of areas where this type of adaptation is researched [Lach, 2018] and has mentioned in chapter 1 although not very common, there are a good amount of titles that already use DDAs. The research conducted in the area can be grouped up in 3 areas: automation, it ranges from adjusting NPC's to the use of PCG, from adjusting elements of the content to creating adjusted content, perceived challenge, how the player perceives difficulty and how subjective it is and finally Player Experience Models as mentioned in 2.1 [Lach, 2018].

Chapter 3

Developed Work

In this section, we discuss how to create a GA that tailors himself around player inputs. To approach this problem we use two different algorithms, Template Generation and Real-time Evolution. In the Template Generation we used a GA to evolve the creatures and create templates to bootstrap the Real-time Evolution. This bootstrapping is done to provide a solid basis for the evolution process and therefore shorten the amount of time needed to reach the optimal solution for the presented problem. In the Genetic algorithm we use Roulette selection, K-point crossover and Boundary mutation. The Real time Evolution uses concepts inspired in how rtNEAT and odNEAT works. Before introducing the problems that the Real-time Evolution GA needs to solve, for clarity and readability purposes, two concepts must be introduced: family and community. A family is a group of individuals of the same species that are genetically very similar, having the same predominant genes. A community is a population of agents of one species.

The Real time Evolution needs to solve the following problems:

- In each iteration it must select a suitable candidate for reproduction, according to their suitability to solve the problem presented;
- In each iteration we must give time for new families, to thrive or prove themselves not suited to solve the current problem;

- we must prevent a family from dominating the community;
- we need to give other possibilities to extinct families to reappear, without compromising the current performance of the community, and at the same time maintaining some diversity in the community.

To address this last point the solution we opted for is to add an increment value to the odd of choosing the family that is going to reproduce, but this type of approach also comes with its problems, mainly:

- must not be too big because otherwise it will cause the creatures selected to not follow a pattern thus becoming random;
- needs to reflect the number of families that exist and have existed;
- and it also must discriminate families according to their historical performance.

To implement the algorithm and display the results we used Unreal Engine 4 and created a custom environment, where the predator creatures are trained to progressively, as a community, adapt to the environment that they are subjected too. This custom environment consists of a square plain with no walls nor obstacles, measuring around 1 km^2.

3.1 Agents

Each predator has a set of correlated and non-correlated attributes. The set of correlated attributes is composed of pairs of attributes, these pairs are subjected to a balancing system that prevents them from getting maxed out without a trade-off. This balancing system works in the following way: one attribute is randomly assign a percentage from 1 to 100, while the other attribute gets 100 minus the random percentage assign to attribute one. This percentage is then used to find the corresponding numerical value of the attribute. This balancing system makes

the correlated attributes inversely proportional, for example in the pair A - B, A is inversely proportional to B. The correlated attributes are physical traits, which will have the most impact in the behaviour demonstrated, and the non-correlated attributes are psychological traits. The names of the families are generated according to the following pattern: Cor, meaning that the next letters refer to correlated genes, first 2 letters of the best gene in each comparison; NCor, meaning that the next letters refer to non-correlated genes and then the first 2 letters of each gene followed by H or L, meaning that it's higher or lower than half its max value. This means that creatures with the same predominant genes will have the same family name, although they might not have the same values in each gene, these values vary between 0 and 1. This also means that different families will have different behaviours and this scales with how much genes are different. The different genes trained in this experiment are:

- Correlated: Health – Attack, Hunger – Metabolism, Speed – Life time;
- Non-Correlated: Aggressiveness , Patience, Conserving Nature, Search Radius.

For the correlated genes, the Health gene gives the predator is total maximum hit points of life, the attack gene gives the damage that each blow gives to the target, the hunger gene assigns the maximum value that the predators' energy bar has, the metabolism gene determines the rate at which the hunger bar is depleted, the speed gene is the maximum speed that the predator can achieve when moving around and the life time gene determines the initial lifespan of the predator.

For the non-correlated genes, the aggressiveness gene determines the maximum distance value that can exist between the predator and the prey before the predator stops chasing, the patience gene determines the maximum amount of time that the predator will spend chasing a prey, the conserving nature gene determines the % of the hunger bar from which the metabolism and speed are halved and the search radius gene determines the maximum search radius for a point to move to.

The mentioned creatures each have a decision tree that acts as a controller for the predators, which can be seen in fig. 3.1. This decision tree is composed by five branches, with the leftmost branch having higher priority of activating and the rightmost the lowest. This means that if the condition for a branch that as higher priority is fulfilled, then the action that is being taken is cancelled and the higher priority one proceeds to be done. From left to right, the first branch returns the predator to its random movement routine and is activated when the predator spends too much time chasing the prey, which is dictated by his Patience gene, or the prey manages to create a gap in distance that is larger then what the predators Aggression gene allows. The second branch has two sub-branches and this branch is activated if the predator finds a prey. The first sub-branch makes the predator chase a prey and when a prey is reached the second branch applies damage to it. The third branch has one sub-branch and is activated if the predator hears noise. The sub-branch moves the predator to the general area where the noise originated. The fourth branch is activated when the predators' energy bar reaches half of its original value and it halves the maximum speed and metabolism of the predator by half while it is not chasing any prey. The fifth branch does not have any activating function and it applies a random movement routine.

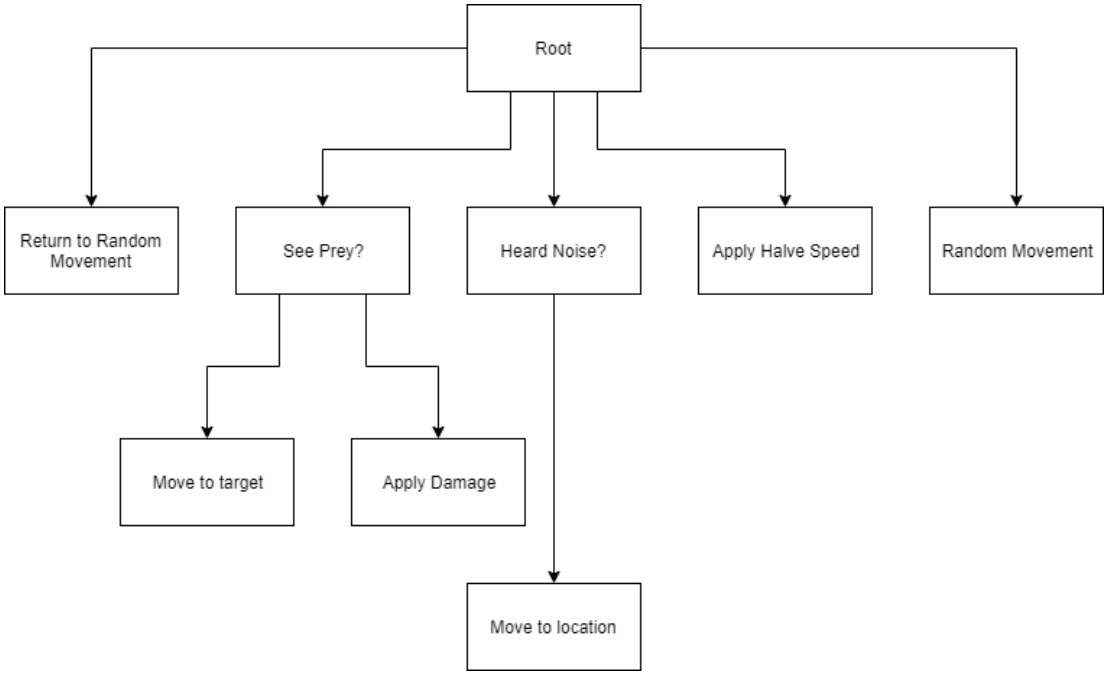

FIGURE 3.1: Predators' Behaviour Tree

Either of the types of attributes that are evolved impact the performance of each individual predator. The correlated attributes impact their physical performances, the damage they can take, the damage they can give, how fast can they reach the prey, in sum these impact their visible performance, on the other hand the non-correlated attributes affect their decision making, how far do they chase, for how long do they chase, how well do they manage their energy. With this we can see that even individuals with the same predominant genes will be different and therefore have slightly different performances.

3.2 Algorithm

3.2.1 Template Generation

In order to obtain the real-time evolution that we want to achieve, we first run a classic genetic algorithm, to generate templates according to two generic scenarios played against other bot creatures, prey. The two scenarios are: Predators are aggressive and preys are passive; Predators are aggressive and preys are also aggressive; In these two scenarios, the creatures were trained for 100 generations, with a community of 20 predators and 10 preys, where when a prey dies it is immediately replaced by another. In each scenario and in each generation, after all the creatures die, the top 1/3 of the population is selected to be the parents of the next generation. With roulette selection, we select two parents to reproduce with crossover, the child then may suffer a mutation in one or more genes. This process is repeated until the number of children matches the initial population. Once the last generation dies, the remaining families present are saved has templates for the Real-time experiment. With the previous step, the initial templates are generated according to any number of given scenarios, two in this case. In fig. 3.2 we can see the process described above.

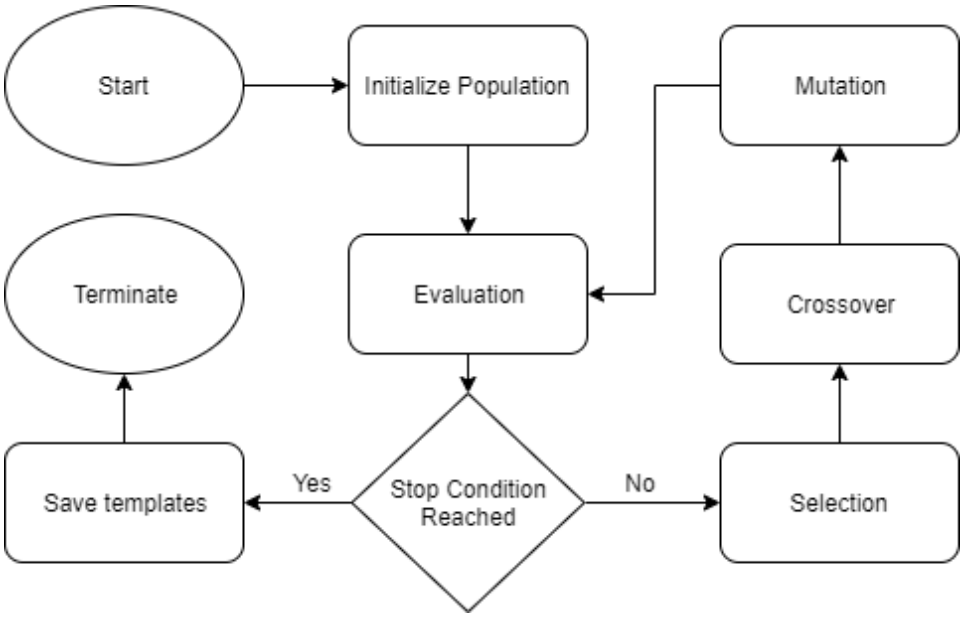

FIGURE 3.2: Template Generation Genetic Algorithm

3.2.2 Real-Time Evolution

Now for the community of creatures to adapt in real-time to inputs provided by the environment, we need to repopulate each time a creature dies and evaluate the performance of the community. This repopulation must solve the problems mentioned above in 3 where a suitable candidate must be selected according to the community's performance, this selection must give a chance for recently added families to thrive or to prove themselves not suited to solve the current problem, it must also prevent a family from dominating the community and it must also give chances to retry bad solutions and re-evaluate their performance without compromising the current performance of the community.

With these constraints in mind and using NEAT concepts as a base, because it already solves the need to protect recently added families problem, the solution we reached is the following:

$$P_i = \frac{mFfi + \delta}{\sum mFf + \sum \delta} \tag{3.1}$$

Where mFfi is the mean of the family being evaluated and mFf is the sum of the means of all families.

Where δ, is the increment given to all families except the previously spawned given by:

$$\delta = cur\delta + pre\delta \tag{3.2}$$

Preδ and curδ are the previous and current δ obtain by the family being evaluated, respectively. Curδ is obtained by

$$cur\delta = \frac{GR \times (Nf - R)}{Nf} \tag{3.3}$$

Where GR is the growth rate of the community, Nf is the number of families that are in the templates and R is the rank of the family. R functions as a discriminator between families that performed good and families that didn't and the values it can take vary from 0 to Nf-1, meaning that a well placed family will receive a better increment than a worse one, but with due time the increment will be high enough for a bad family to spawn and be reevaluated.

The GR serves to verify the need for new solutions and is obtained by

$$GR = \frac{Ft + Ftpre}{Ft} \tag{3.4}$$

where Ft is the total fitness, Ftpre is the previously total fitness. If Ft is higher than Ftpre then GR is going to be small, which makes δ smaller. This means that the community is evolving in a good direction and there is little reason to change course. While if Ft is smaller than Ftpre then δ is going to be larger which means

that a new family is likely to be spawned, this in short will broaden the search space in detriment of optimization.

The predators both in the template generation and real time evolution gain fitness according to the following equation:

$$F = Ppk + Agg + Surv \tag{3.5}$$

where Ppk is the points per kill and is obtain by:

$$Ppk = (Nplk \times 50) + (Nprk \times 15) \tag{3.6}$$

Where Nplk is the number of players killed and Nprk is the number of prey killed.

Agg from 3.5 means aggression and is obtained by:

$$Agg = (Ntdd \times (Atk \times 0.1)) \tag{3.7}$$

Where Ntdd is the number of times that damage was dealt and Atk is the Attack gene value.

Lastly from 3.5 Surv means survivability and is obtained by:

$$Surv = (\frac{Ta}{10} \times 0.6) + (Dist \times 0.6) \tag{3.8}$$

Where Ta is the time alive and Dist is the distance traversed.

Each time a predator dies, it is replaced by another using 3.1 to evaluate and using roulette selection to select a family to spawn. After the family selection we verify each member of that family to pick the best one, then we proceed to compare this member to the representative of that family. The representative of a family is the best performing individual of that family in that experiment. If the family has no representative, then the member we picked earlier becomes the representative. Then we apply mutation to the representative. This process is repeated until the experiment ends.

By using mean fitness's to evaluate the population we protect recently added families enough for them to show their worth. By using an increment allied with the mean fitness we can maintain a some diversity in the population, which prevents a family from totally dominating a community. The increment also allows previously bad solutions to be re-evaluated further in the experience and this is also affected by how the community is performing.

After the template generation previously mentioned, we can initiate the real time evolution. In this experience predator's behaviour is firstly tested against prey bots and in future work tested against players (people).

The Real Time Evolution was tested in four different experiments, in the first two we tried to replicate the results obtained in the template generation, to prove that the real-time evolution algorithm behaved in a similar way to the template generation one. The last two experiments we try to prove that there is an adaptation to a changing environment.

3.2.2.1 Hostile Predators vs Passive Prey and Hostile Predators vs Aggressive Prey

Like in the template generation algorithm the first and second tests done in the Real-time Evolution are of the same type, testing only against one type of prey, either passive or hostile.

The tests are conducted with 20 predators and 10 prey, same has in the template generation and it was ran for 5000 iterations, each iteration corresponds to one death of a predator.

3.2.2.2 Swap Experiment and Reverse Swap Experiment

These experiments both were done with the same settings has the previous ones. On both of these experiments the prey start as either passive or aggressive and at half point in the experience they progressively switch from one behaviour to another when they die. To clarify this means that when one prey dies, it is replaced by one with the other behaviour, if the middle of the experience as been reached. This provides a progressively changing environment that will allow us to verify if there is an adaptation of the community of predators to the new behaviours in the prey. The half point in each experiment is the 2500 iteration, but time wise this varies due too variance in creatures lifespan and death rate.

Chapter 4

Experiments and Results

4.1 Template Generation

Using the approach presented in the previous chapter, on the Template Generation we want to obtain distinct families in the first and second scenario to obtain different behaviours. The expected behaviour is has follows: in the first scenario a predator clearly more attack focused while on the second scenario a more defensive predator.

4.1.1 Hostile Predators vs Passive Prey

The results obtained from the scenario 1 match the expectations, where the surviving families were:

Trial	Genes						
	Correlated			Non-Correlated			
	Health/Attack	Hunger/Metabolism	Lifetime/Speed	Aggressiveness	Patience	Conservative Nature	Search Radius
1	Attack	Hunger	Lifetime	Low	High	Low	High
2	Attack	Hunger	Lifetime	High	High	Low	Low
3	Attack	Metabolism	Lifetime	Low	High	High	High
3	Attack	Metabolism	Lifetime	Low	High	Low	High

Table 4.1: Surviving Families Experiment 1

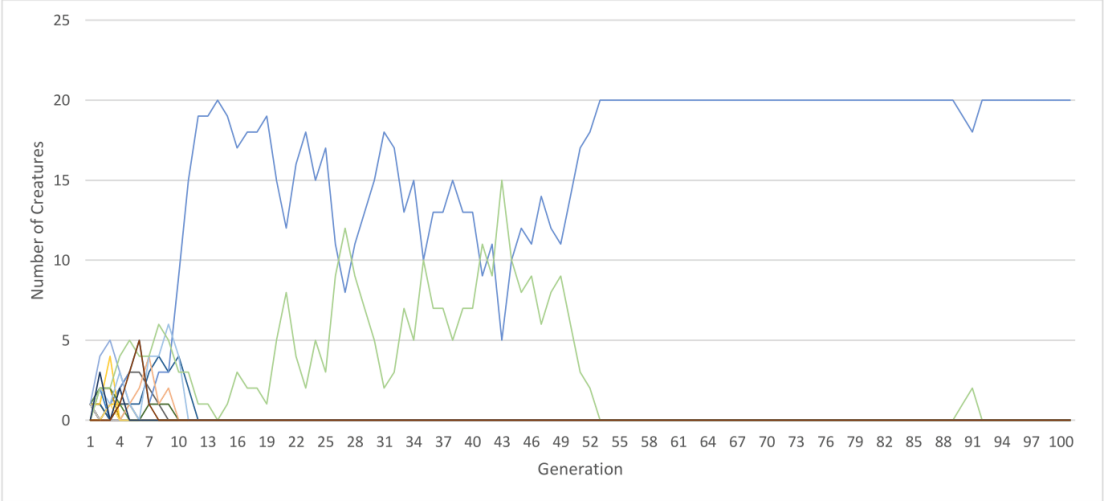

FIGURE 4.1: Number of Creatures per Family by Generation Trial 1

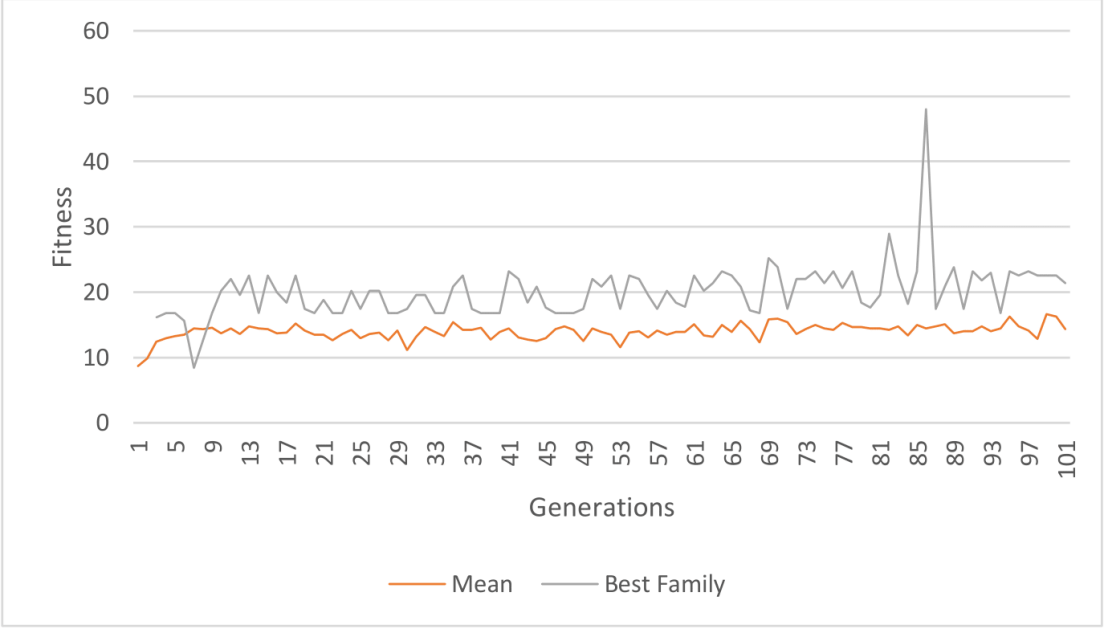

FIGURE 4.2: Mean Fitness Trial 1

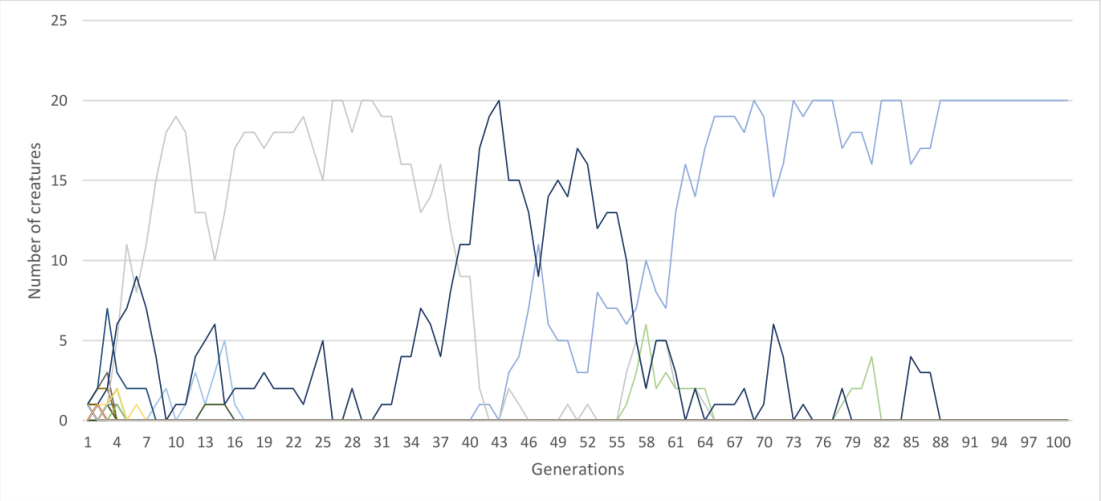

FIGURE 4.3: Number of Creatures per Family by Generation Trial 2

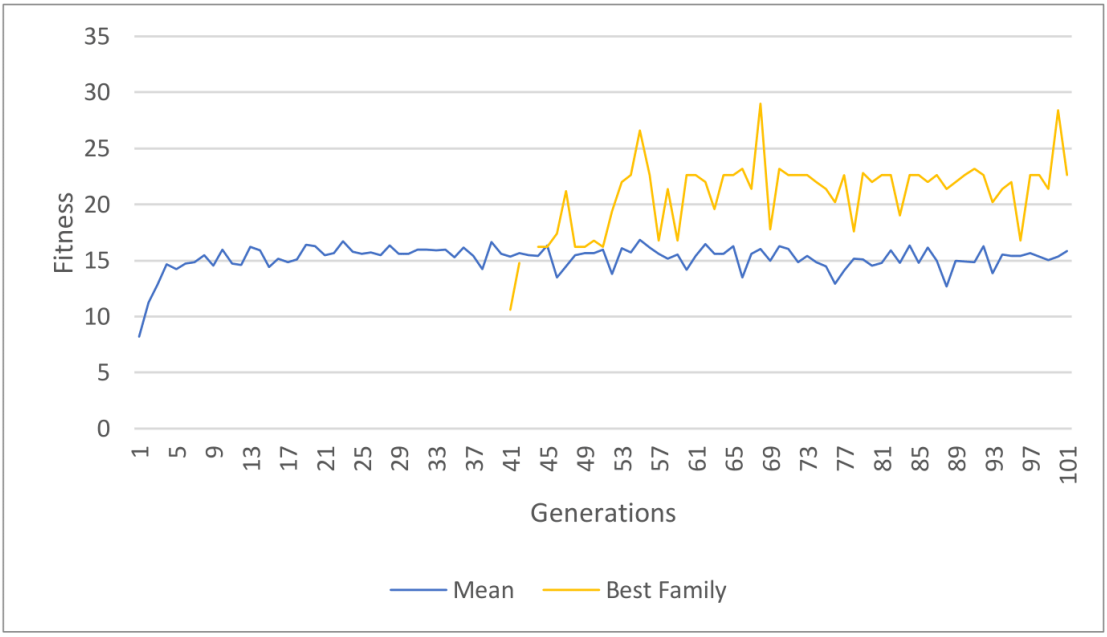

FIGURE 4.4: Fitness trial 2

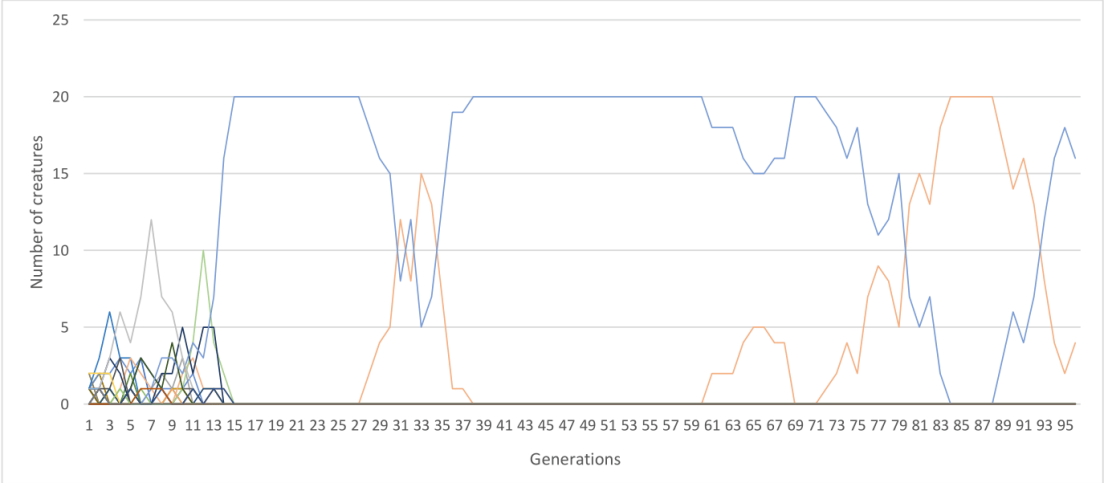

FIGURE 4.5: Number of Creatures per Family by Generation Trial 3

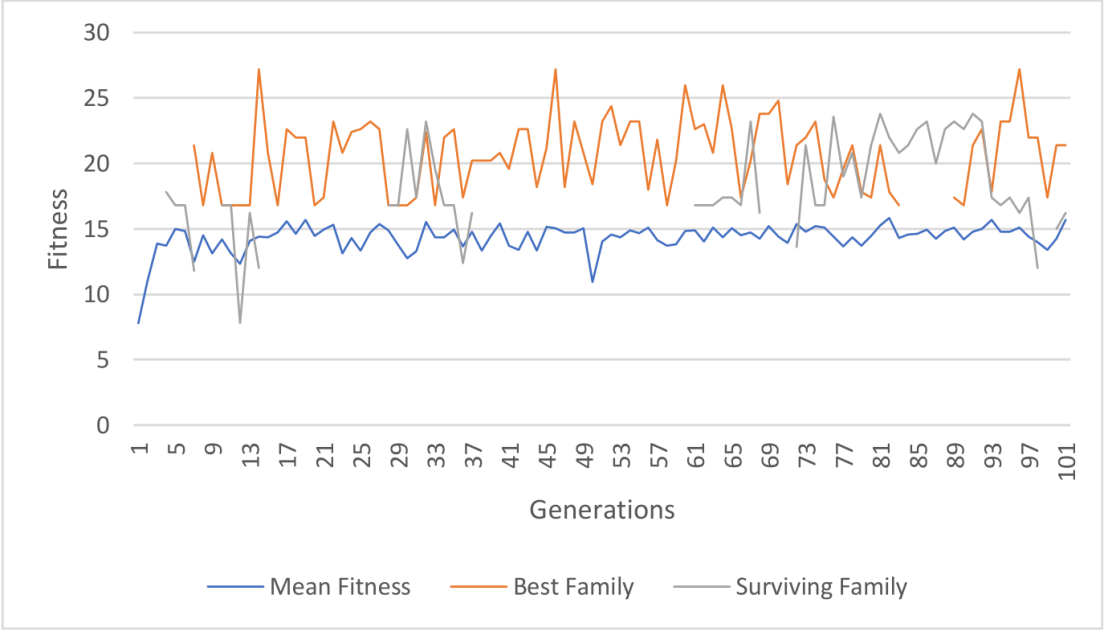

FIGURE 4.6: Fitness Trial 3

We can see that in all three trials (4.1 to 4.6) there is a considerable number of families present in the community but after a couple iterations one or more families take a hold. In all trials we can see that in the first couple of generations there is a reasonable amount of different families and as the experiment unfolds we can see that only a couple of families survive and fight for supremacy, with an occasional mutation. As a reminder the conditions for these three trials are Hostile Predators vs Passive Prey. In trial 1 we can clearly see that after the

initial generations there are two families that distinguish themselves and until the middle of the experiment the family represented in light green (4.1) shows good progression towards dominance but the light blue family mutated slightly, remaining in the same family but with better stats in their spectrum, which gave it the edge and proceeded to dominate this trial. A similar situation happens in trial 2 and 3, were in trial 2 the grey family declines and subsequently goes extinct, the family that takes her place (deep blue) after some generations also declines and goes extinct and then a light blue family (4.3) (which is a different family from the previous trial) takes hold, we can further verify that this family appears due to a mutation (4.4) in one of the other competing families, while in trial 3 only two families go back and fourth. We can see in all three trials that the best families achieve a higher than mean fitness but seem to reach the maximum possible given the scenario, around 20 fitness, except for one case in which a creature achieved 48 fitness in trial one (4.4). This outlier predator obtained this amount of fitness by managing to kill weakened prey.

4.1.2 Hostile Predators vs Aggressive Prey

The results obtained on the second scenario revealed that when presented with an hostile environment they tend to evolve in a more defensive manner, opting for genes that prolong their lifespan, such has Health and Metabolism. The surviving families in this scenario can be seen in 4.2.

Trial	Genes						
	Correlated			Non-Correlated			
	Health/Attack	Hunger/Metabolism	Lifetime/Speed	Aggressiveness	Patience	Conservative Nature	Search Radius
1	Health	Hunger	Speed	High	High	High	High
1	Health	Hunger	Speed	High	High	Low	High
2	Health	Metabolism	Speed	High	High	High	High
2	Health	Metabolism	Lifetime	High	High	High	High
2	Health	Metabolism	Speed	High	High	High	Low
3	Health	Metabolism	Speed	High	Low	High	High
3	Health	Metabolism	Speed	Low	Low	High	High
3	Health	Metabolism	Lifetime	High	Low	High	High

TABLE 4.2: Surviving Families Experiment 2

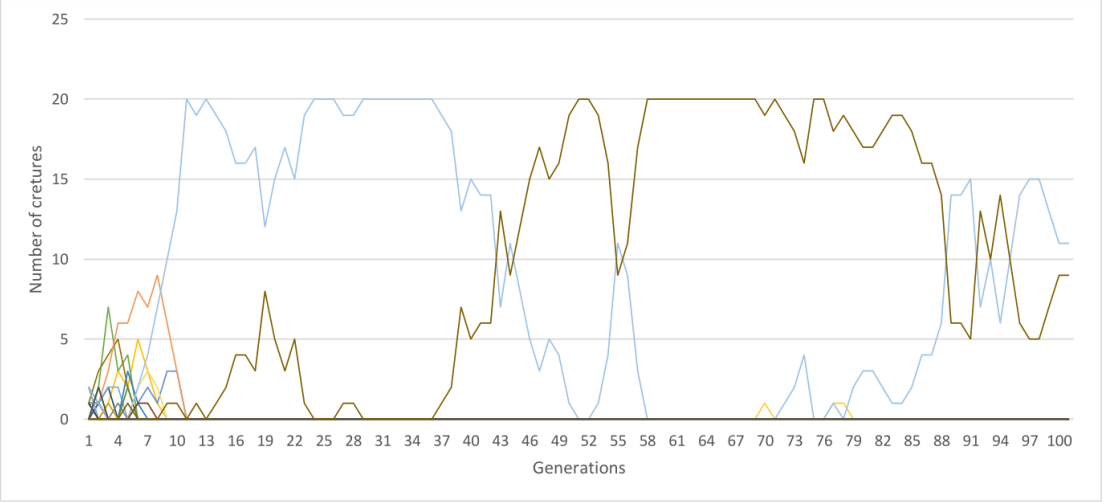

FIGURE 4.7: Number of Creatures per Family by Generation Trial 1

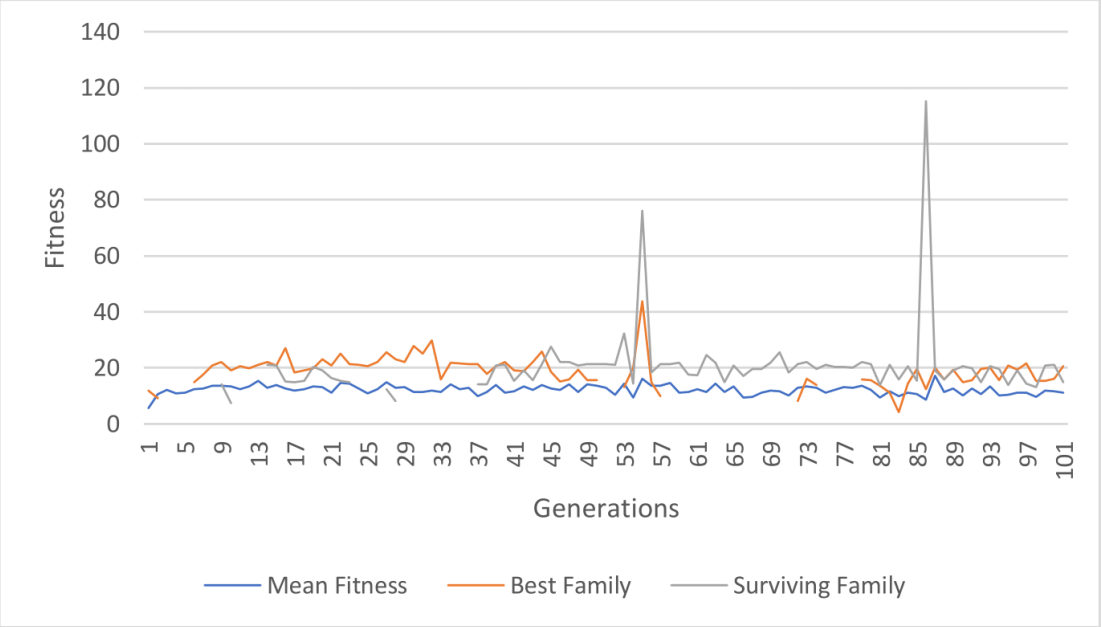

FIGURE 4.8: Fitness trial 1

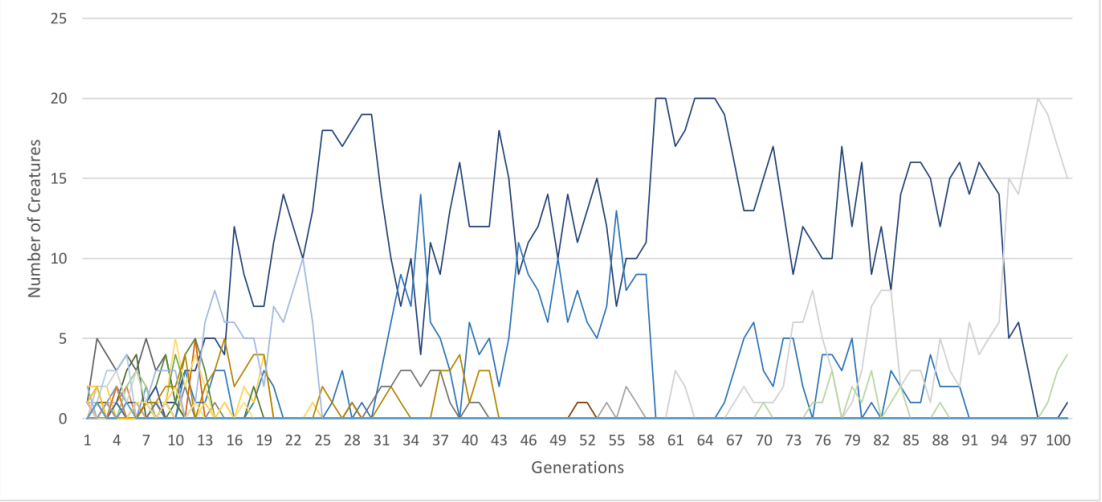

FIGURE 4.9: Number of Creatures per Family by Generation Trial 2

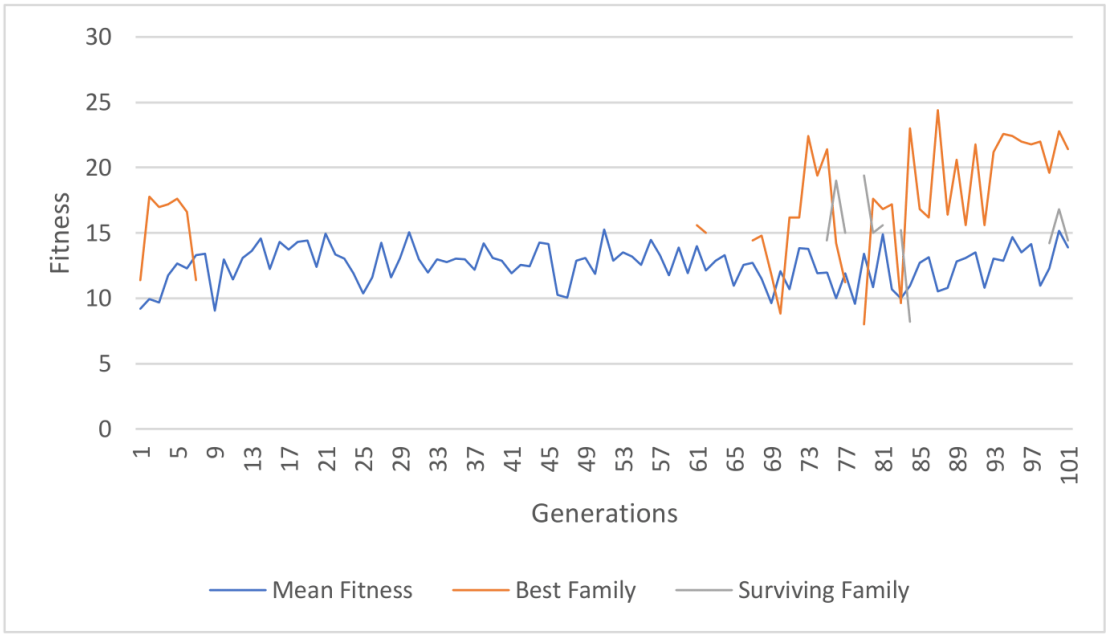

FIGURE 4.10: Fitness trial 2

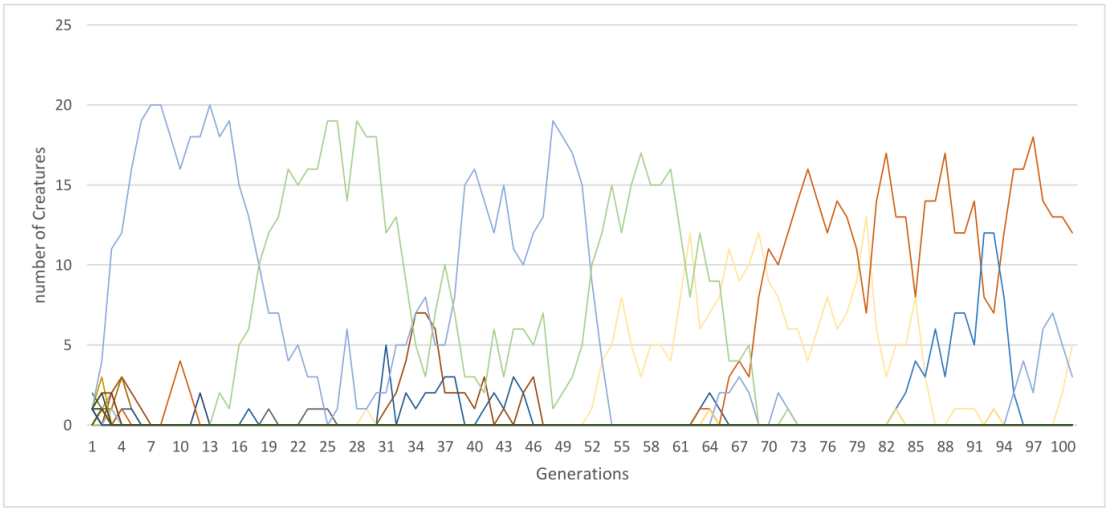

FIGURE 4.11: Number of Creatures per Family by Generation Trial 3

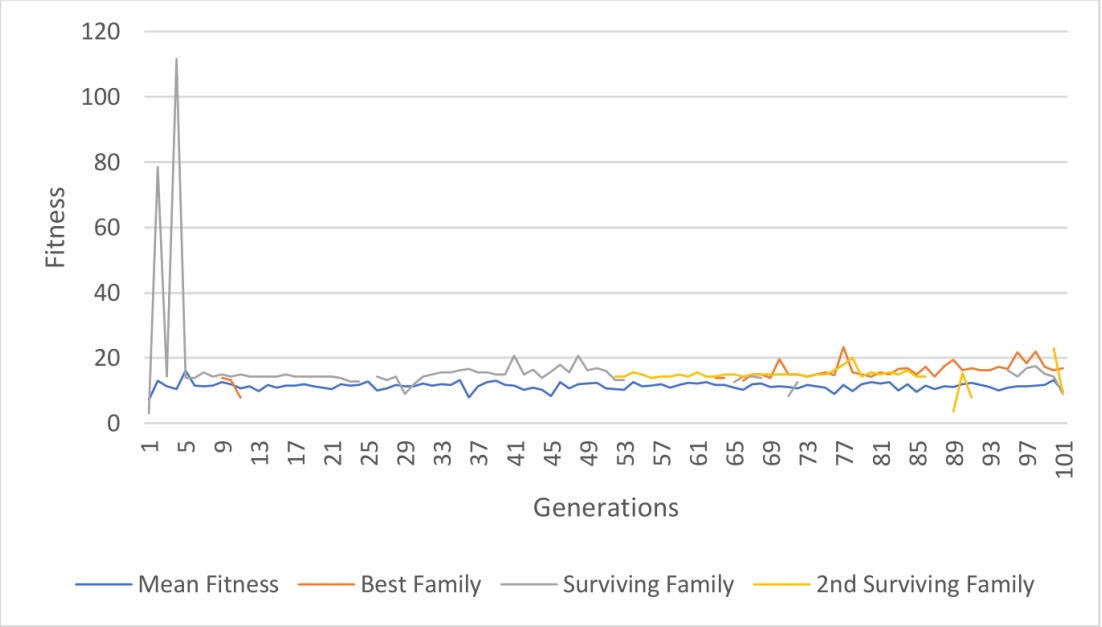

FIGURE 4.12: Fitness trial 3

We can once again see emergence of many families in the early generations and then they are trimmed out has the experiment progresses, but this time we see from the charts that the evolution doesn't show a clear best solution to the environment, instead it broadens its search by applying a diverse array of solutions. In trial 1 4.7 we can see that two families cyclically share the hold on the community, this can have two explanations: that one of them, like explained in the previous scenario, suffered a slight mutation that improved their potential or that in this experiment something like a grouping behaviour was presented by the prey. By grouping we mean that the prey grouped up against single individuals at a time, which may influence their performance. Also this grouping behaviour is most likely caused by the proximity between predator agents and prey agents, because the aggressive prey lock on the nearest predator and with the progression of the experiment as they get nearer to each other, the preys tend to be more likely to lock on the same predator. The grouping behaviour is present in all of these three trials. In the second trial 4.9, the same cyclical pattern is present but, in this trial, we can see a family that was extinct in the early generations coming back and be the most successful. In trial 3 4.11 the emergence of families continues almost until half of the experiment with some predominance of two families that later are replaced

by two families that mutated from them. Much like in the previous experiment the best and surviving families both have most of the time more fitness than the mean, and we can also see from these charts that in this experiment a higher than normal fitness could be achieved (4.8, 4.10, 4.12) and we can also clearly see where the emergence and resurgence of the families happen.

4.2 Real Time Evolution

In these executions we show the families that appeared in each instant of time and also the type of family that represents it. In this representation we group the families according their first two genes and labelling them has atkT, atk, def and defT, meaning total attacking, moderate attacking, moderate defensive and total defensive. This type of labelling allows us to better visualize the behaviour tendencies of the community over time with more precision and unambiguously.

4.2.1 Hostile Predators vs Passive Prey

This execution is expected to yield results similar to the first scenario in the Template Generation experiment, where more aggressive families are the norm.

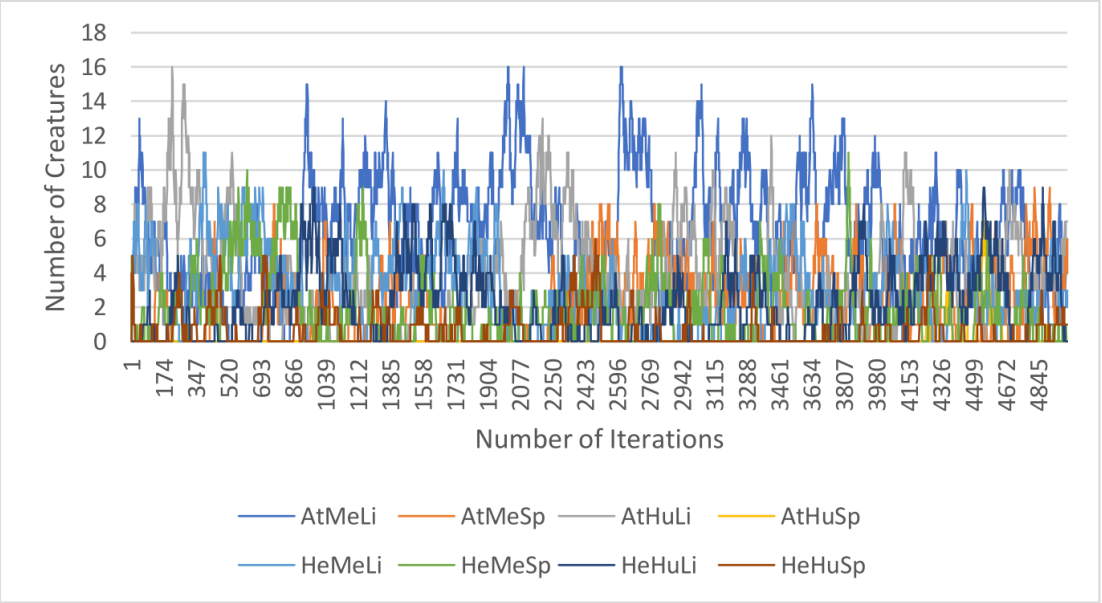

FIGURE 4.13: Number of Creatures per Family per Iteration

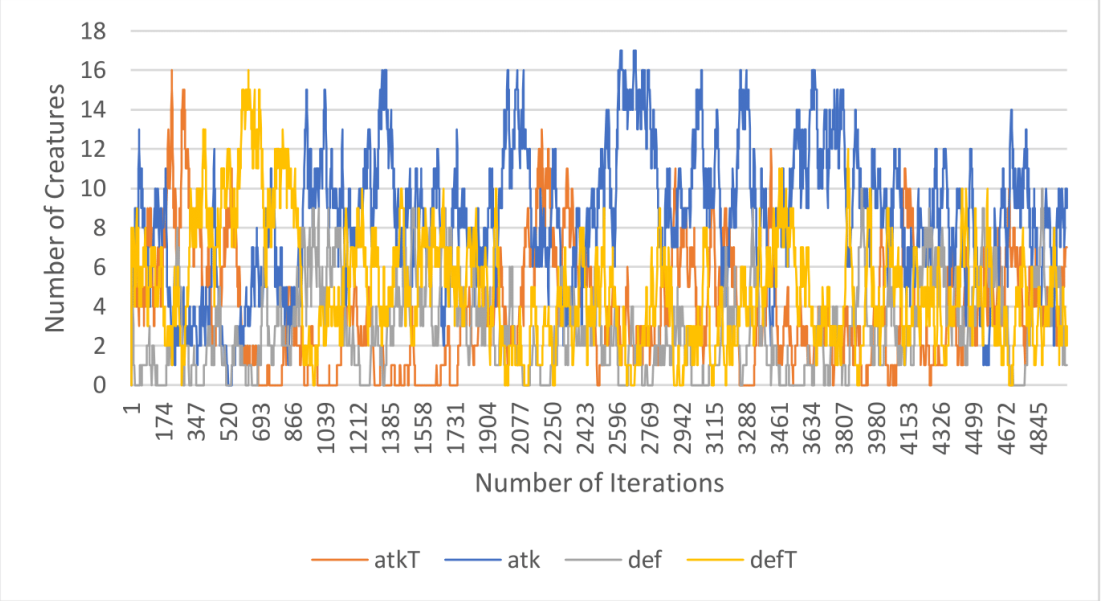

FIGURE 4.14: Number of Creatures per Type of Family per Iteration

And has it can be verified in 4.13, the families that are predominant are the same that are observed in the Template Generation for the same scenario, which are the AtMeLi and AtHuLi families.

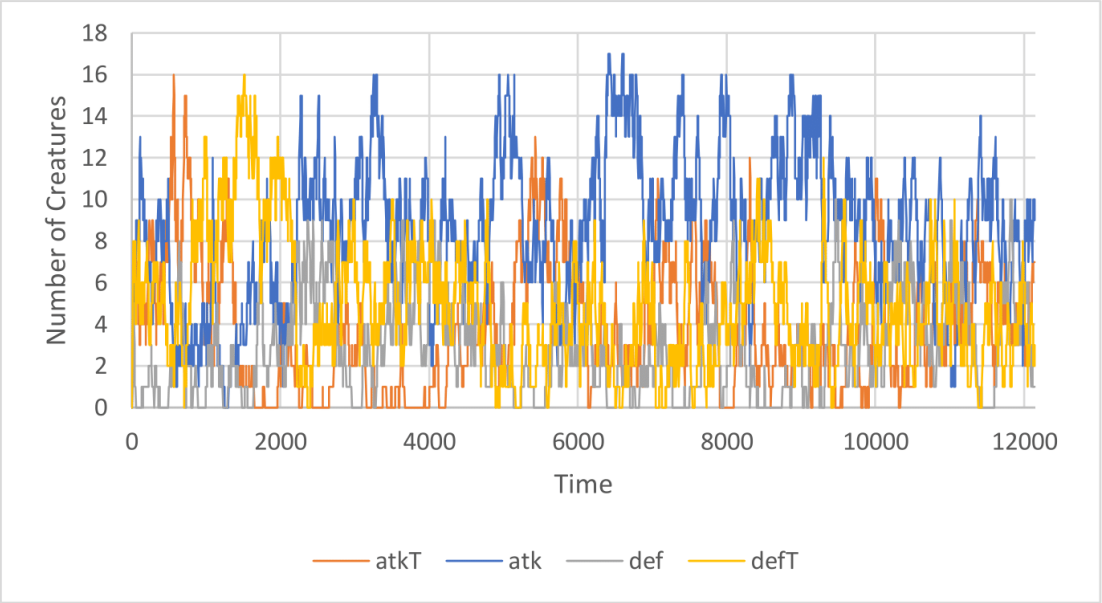

FIGURE 4.15: Number of Creatures per Type of Family per second

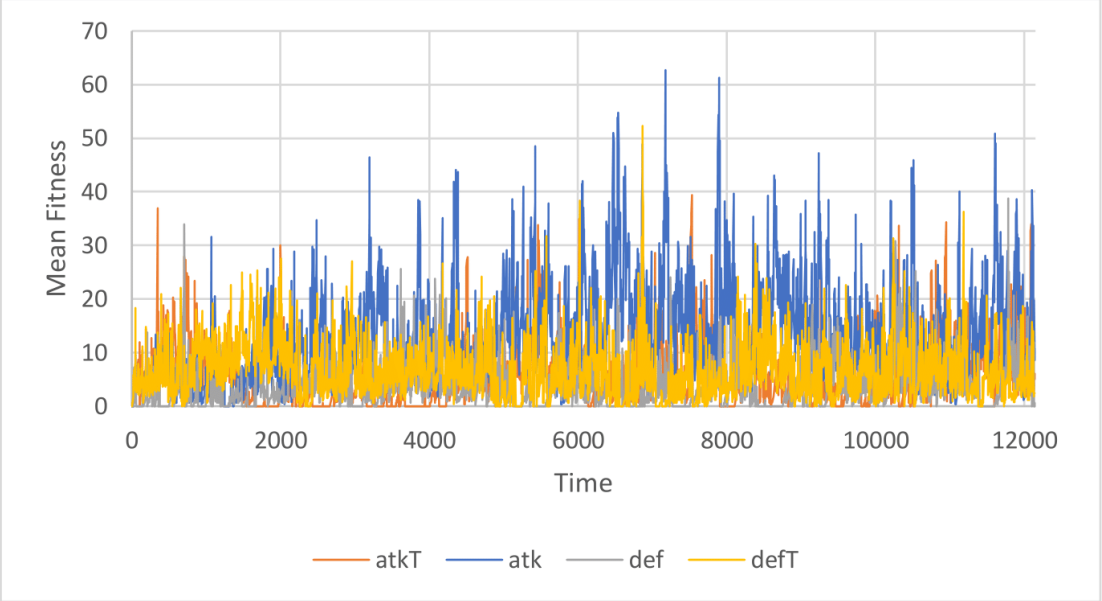

FIGURE 4.16: Mean Fitness per Type of Family

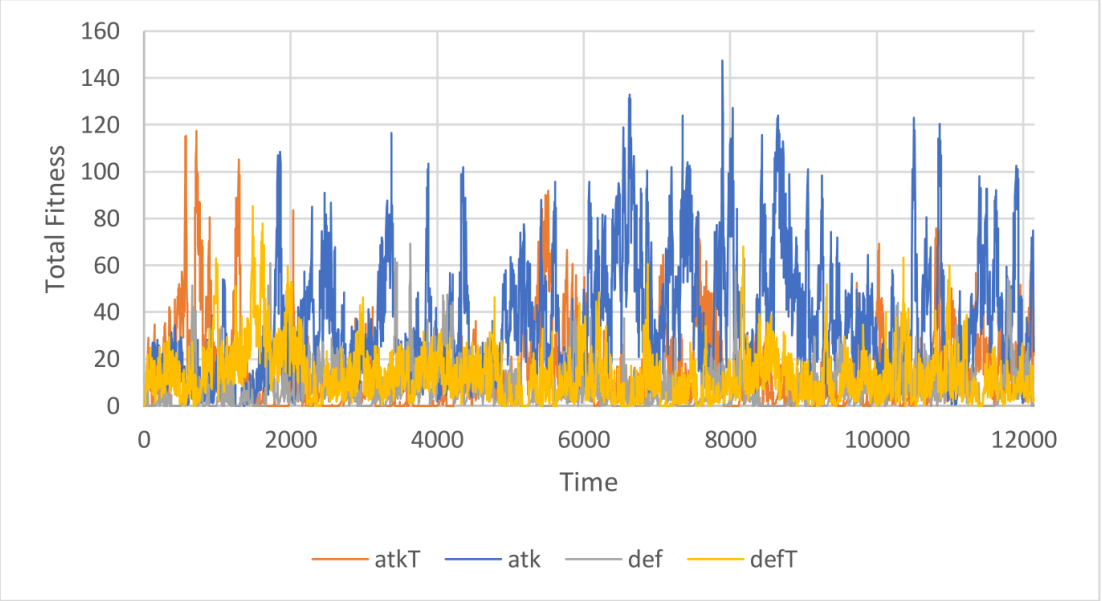

FIGURE 4.17: Total Fitness per Type of Family

We can also further verify in 4.14 and 4.15 that aggressive families indeed dominate the experiment during the majority of the iterations/time, where we can see that defensive families only have around thirteen times where they have more mean fitness than the rest and of those spikes only between the 335 and 836 iterations do we see this type of families dominate. This can be further proven by observing the mean and total fitness values of the types of families in 4.16 and 4.17, as well as observing 4.14 for the spike observation.

4.2.2 Hostile Predators vs Aggressive Prey

In this execution the results that are expected are the opposite of the previous one and in line with the ones obtain in the second scenario in the Template Generation experiment.

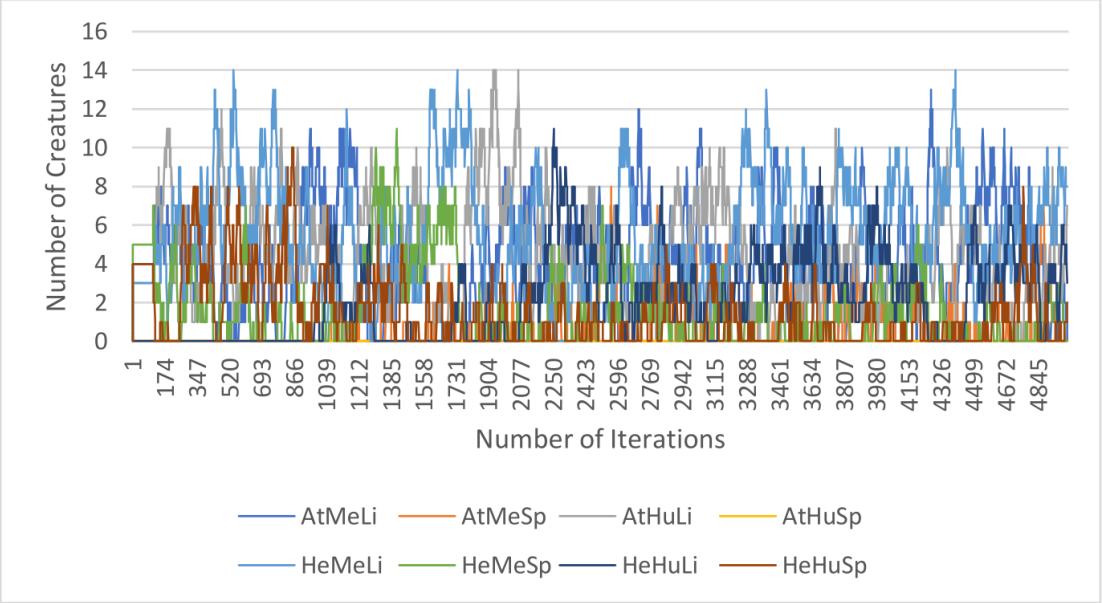

FIGURE 4.18: Number of Creatures per Family per Iteration

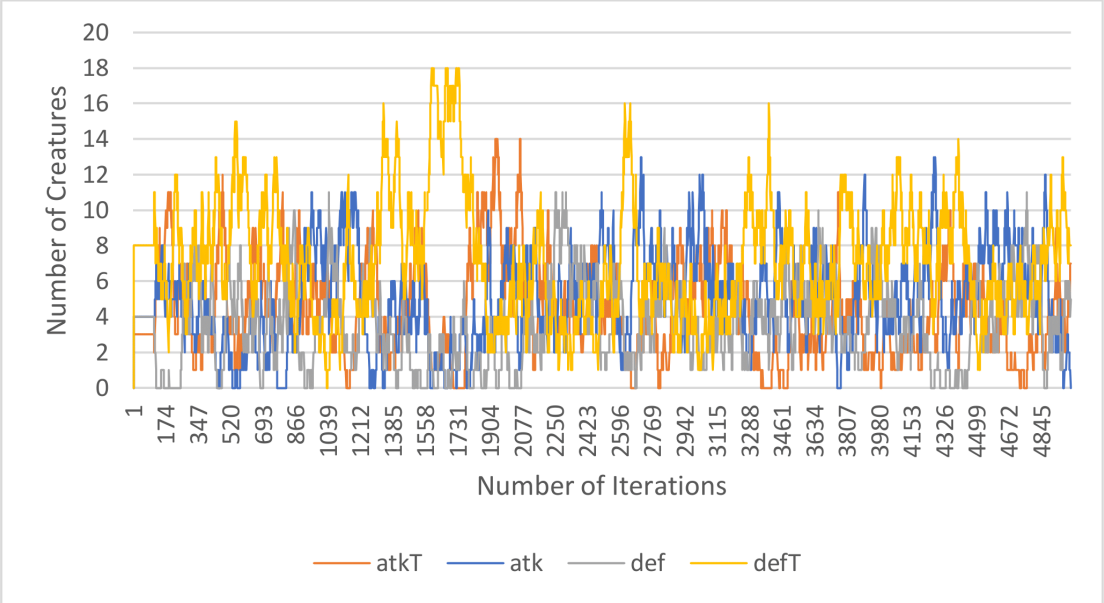

FIGURE 4.19: Number of Creatures per Type of Family per Iteration

The results match the expected ones, and this can be seen in 4.18 where we notice a predominance of a couple of families which belong to defensive types of families, def and defT, seen in 4.19 and 4.20.

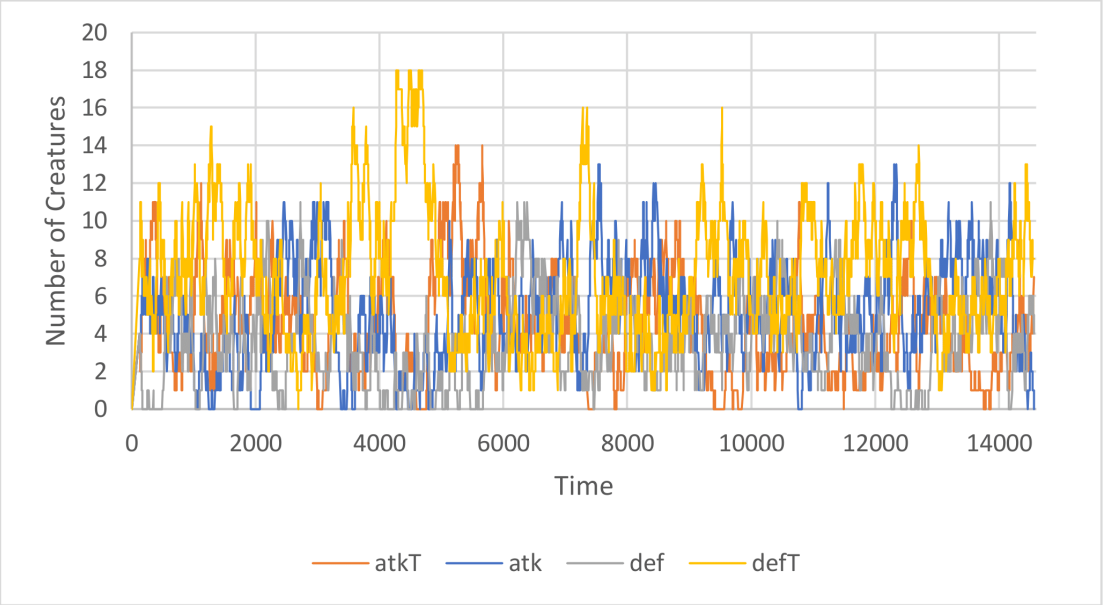

FIGURE 4.20: Number of Creatures per Type of Family per second

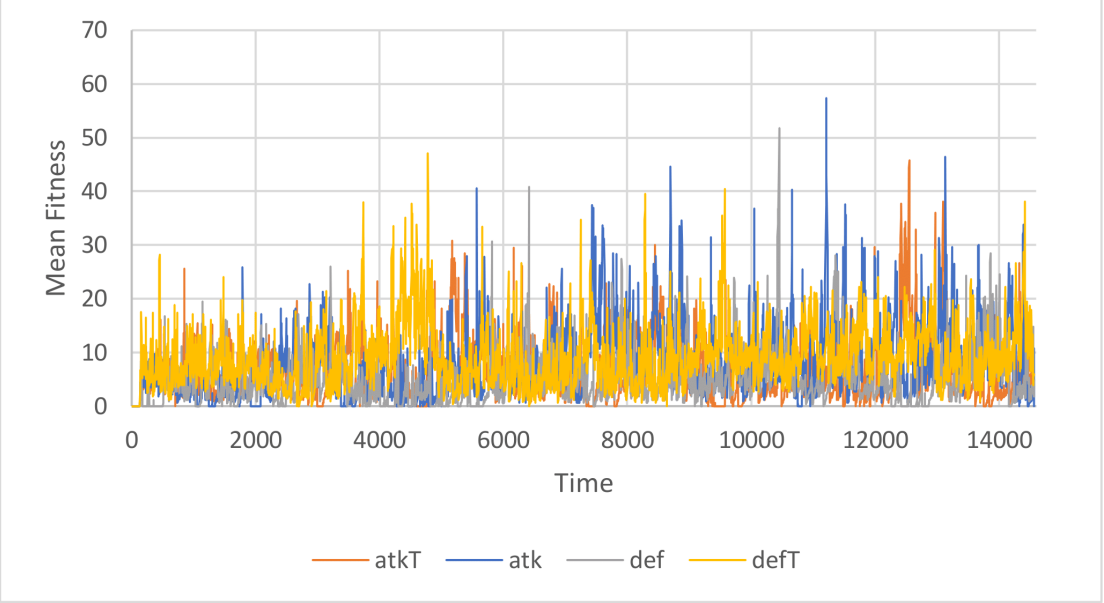

FIGURE 4.21: Mean Fitness per Type of Family

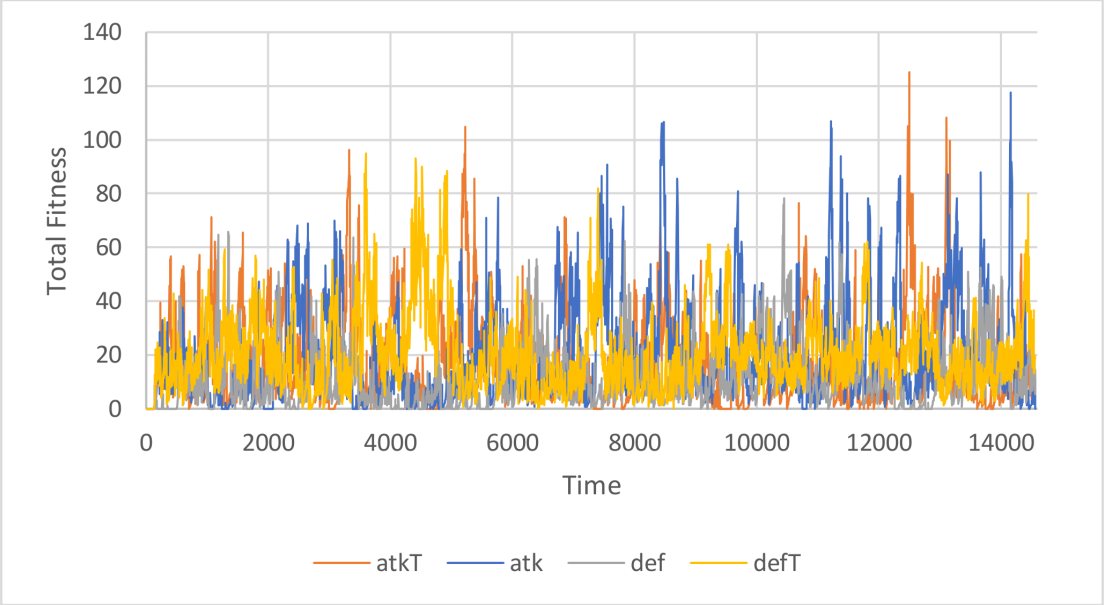

FIGURE 4.22: Total Fitness per Type of Family

Although the majority of the experiment is dominated by a total defensive type family (defT) there are zones where moderated defensive type ones appear and even moderate aggressive and total aggressive type ones. We can verify with the fitness graphs, 4.21 and 4.22 that those appearances are more than lucky spikes, these appearances seem to be an unexpected byproduct of the contest between predator and prey, that is also present in the following swap experiments and that will be further discussed in section 5 of this thesis.

4.2.3 Swap Experiment

For clarity purposes its worth to mention that the half point of this experiment is 2500 iteration which is around the 5800s mark.

In this experiment we test swapping progressively the prey from passive to aggressive at half point. The expected results are dominance of aggressive predators in the first half of the experiment and after a transition period a swap from the aggressive types to defensive ones.

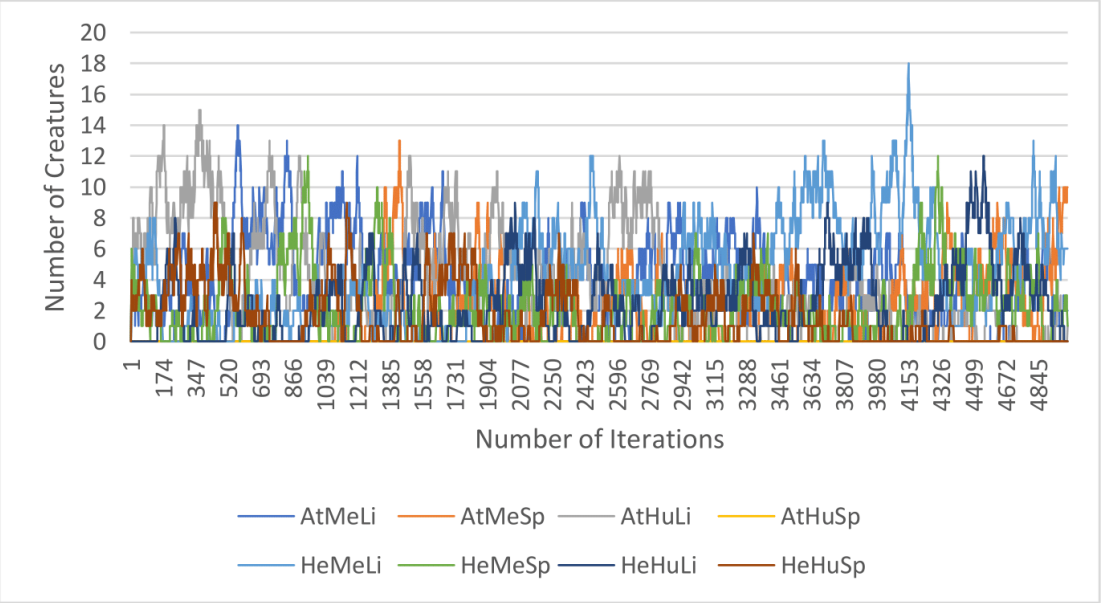

FIGURE 4.23: Number of Creatures per Family per Iteration

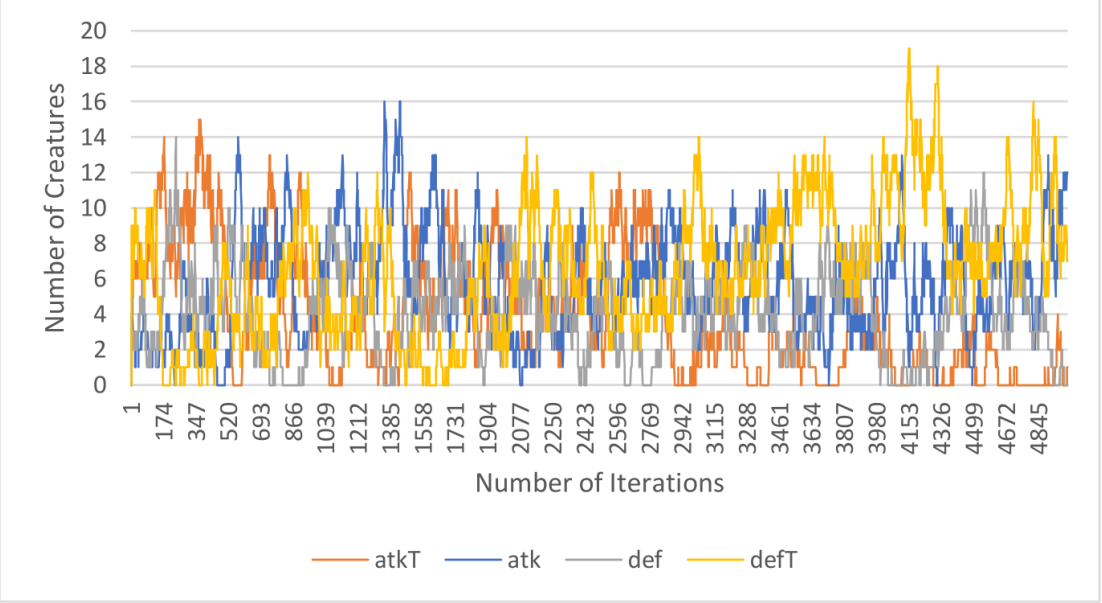

FIGURE 4.24: Number of Creatures per Type of Family per Iteration

The obtained results that are within the expectations, taking into account the results previously obtained in the Hostile Predators vs Passive Prey and Hostile Predators vs Hostile Predators experiences. From 4.23 we can see that the predominant families in each half of the experiment are different and in 4.24 and 4.25 we can verify that the types of families switch according to changes in the environment.

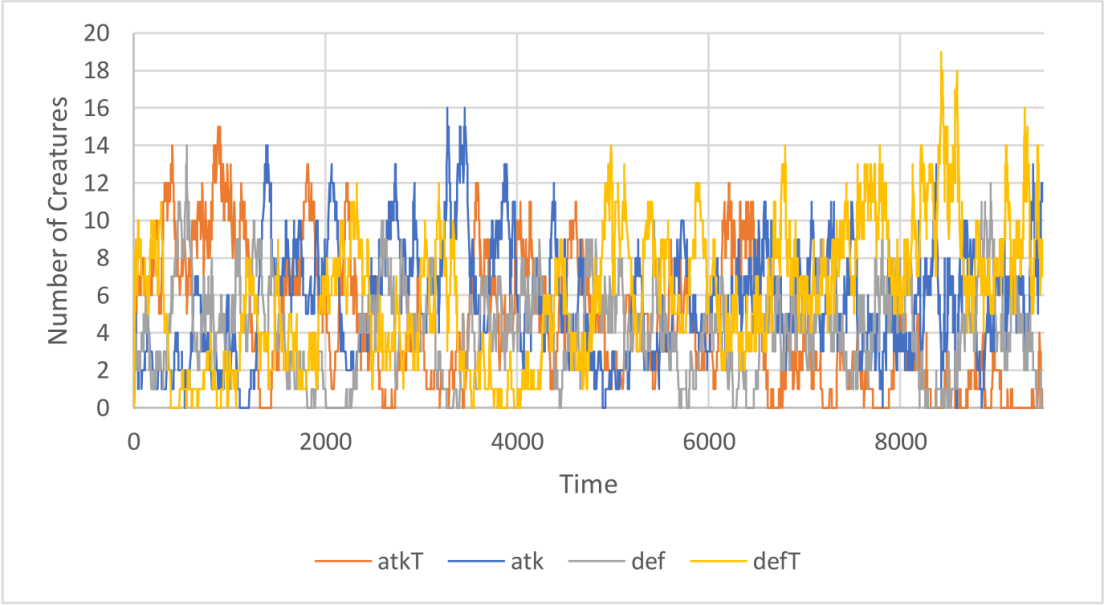

FIGURE 4.25: Number of Creatures per Type of Family per second

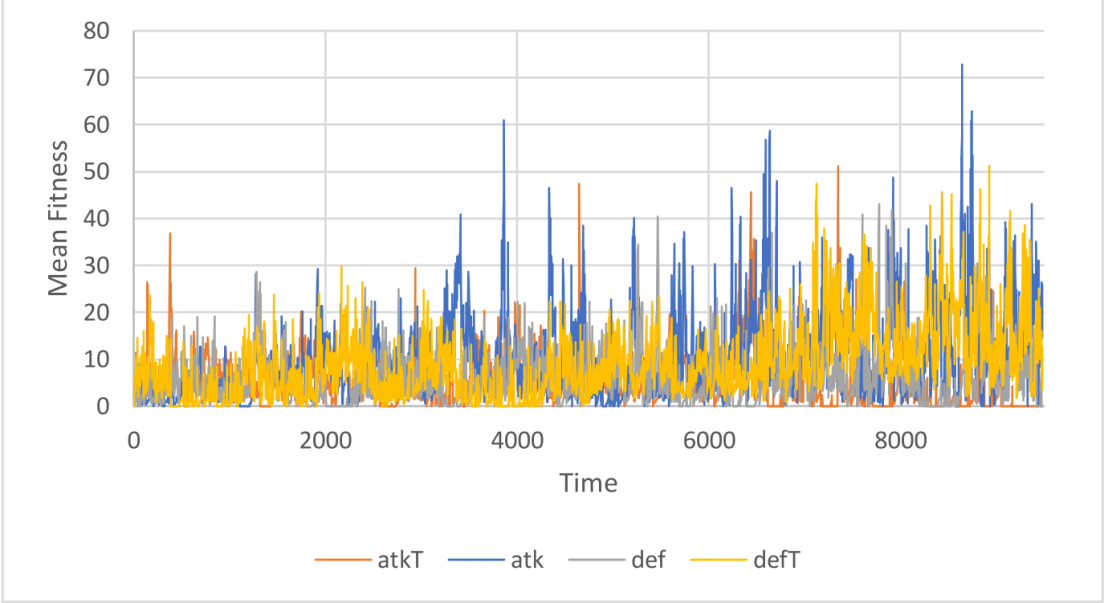

FIGURE 4.26: Mean Fitness per Type of Family

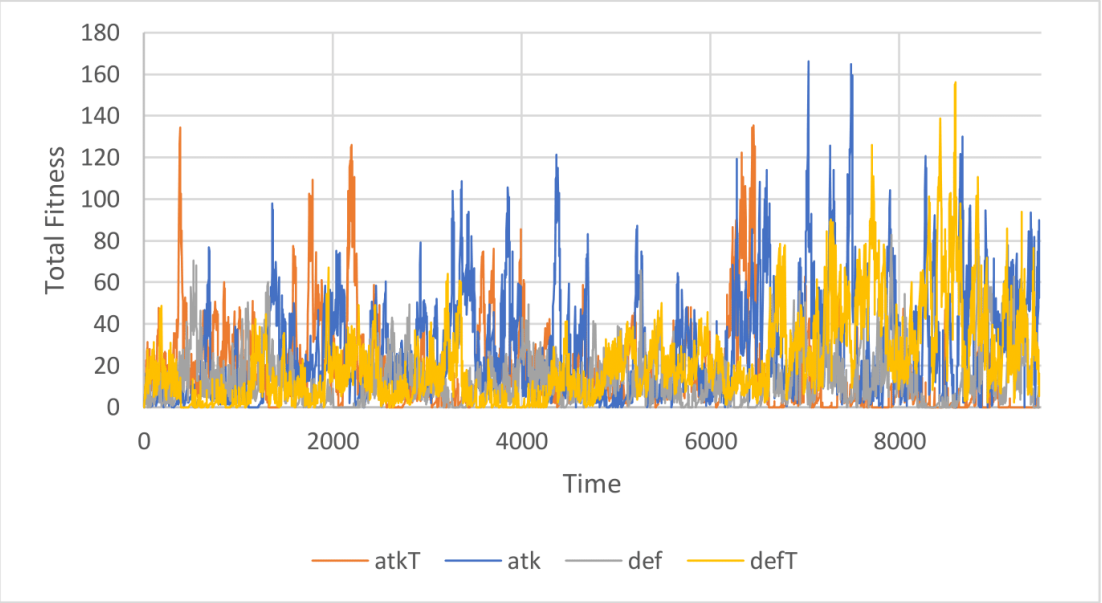

FIGURE 4.27: Total Fitness per Type of Family

In 4.26 we can also see during the first half of the experiment that the mean fitness was only slightly better for aggressive predators in contrast to what happens after where there is an increase in the mean fitness overall and the more defensive types emerge. Although from 4.27 we can see the aggressive type families more clearly dominating the first half and the second half being dominated by defensive typed families.

4.2.4 Reversed Swap Experiment

For clarity purposes its worth mentioning that the half point in this experiment is the 2500 iteration which is around the 8400s mark.

In this experiment we test swapping progressively the prey from aggressive to passive at half point. The expected results are dominant defensive predators in the first half and after a transition period a swap from defensive types to aggressive ones in the second half.

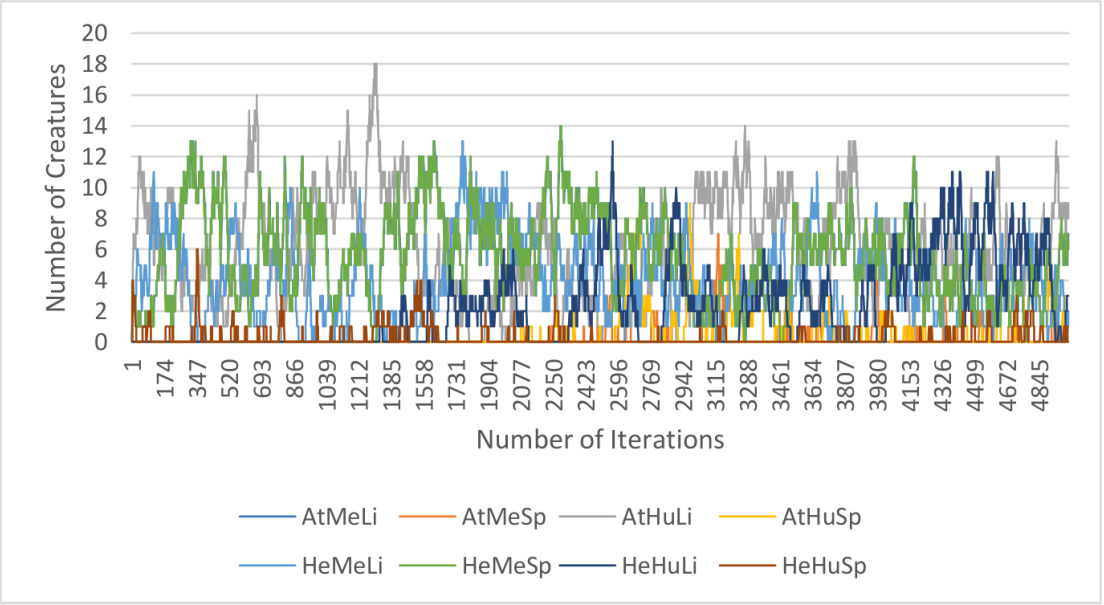

FIGURE 4.28: Number of Creatures per Family per Iteration

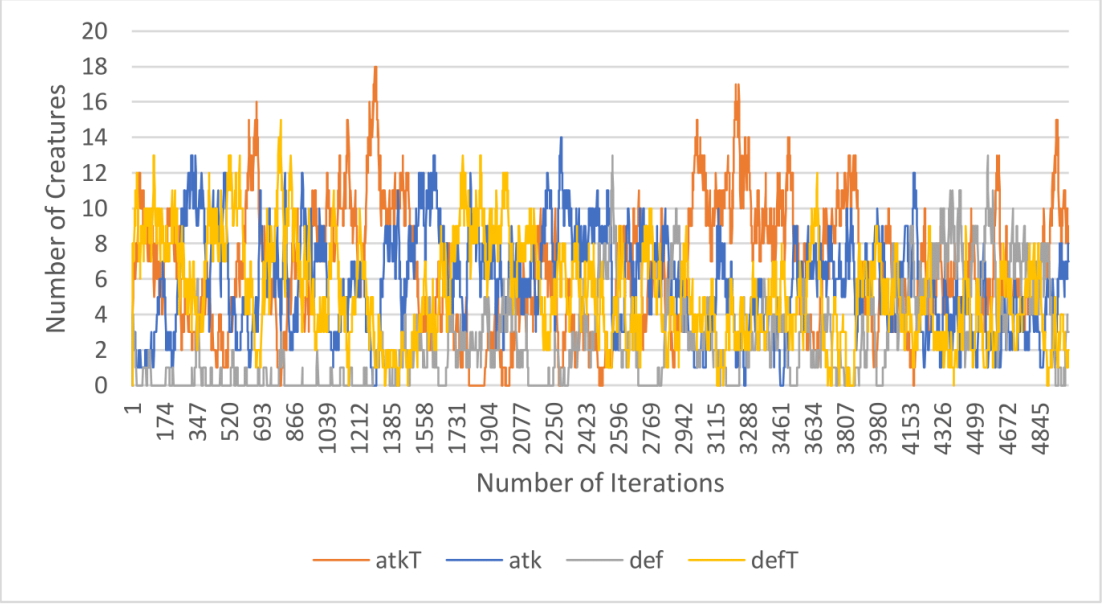

FIGURE 4.29: Number of Creatures per Type of Family per Iteration

The obtained results are not fully within the expectations taking into account all the previously done experiments, from 4.28 we can see that two types of families, AtMeLi and AtHuLi in the first half and HeHuLi in the second half, appear and thrive in a environment that according to previous experiments is not suitable for them. What is observed here in the first half of the experiment is the same that happens in 4.2.2, where due to the competition between prey vs predator and even

predator vs predator, some aggressive predator manage to gain more fitness than expected and subsequently are selected for reproduction.

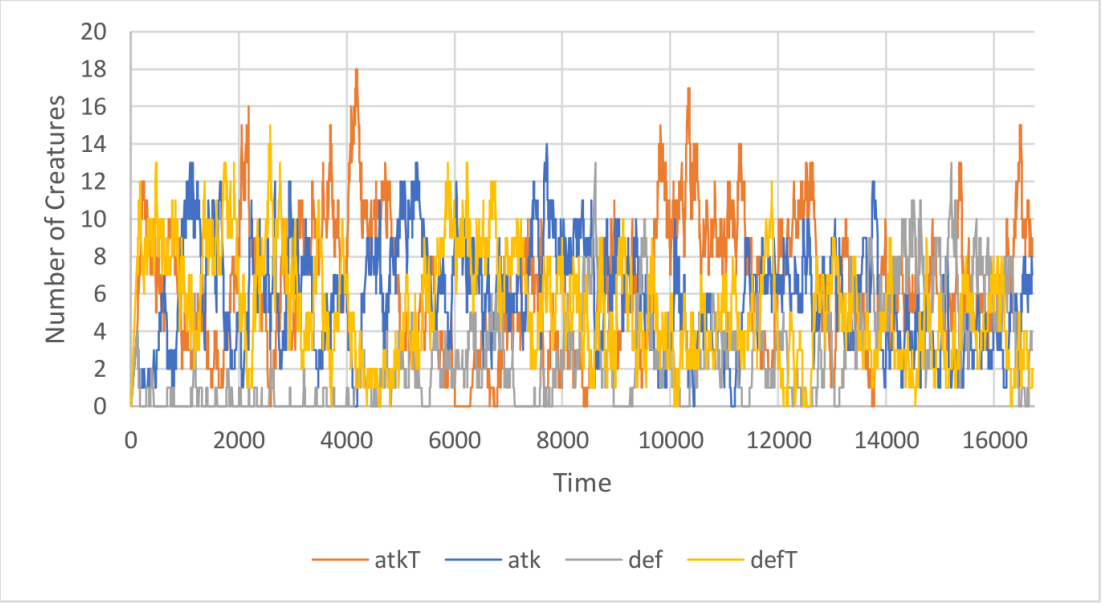

FIGURE 4.30: Number of Creatures per Type of Family per second

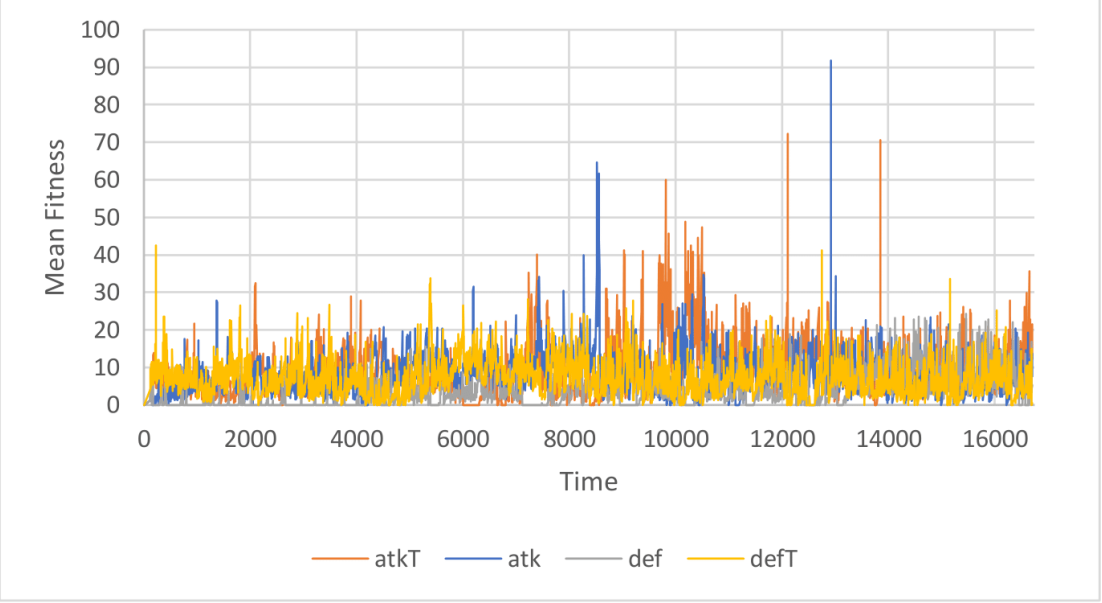

FIGURE 4.31: Mean Fitness per Type of Family

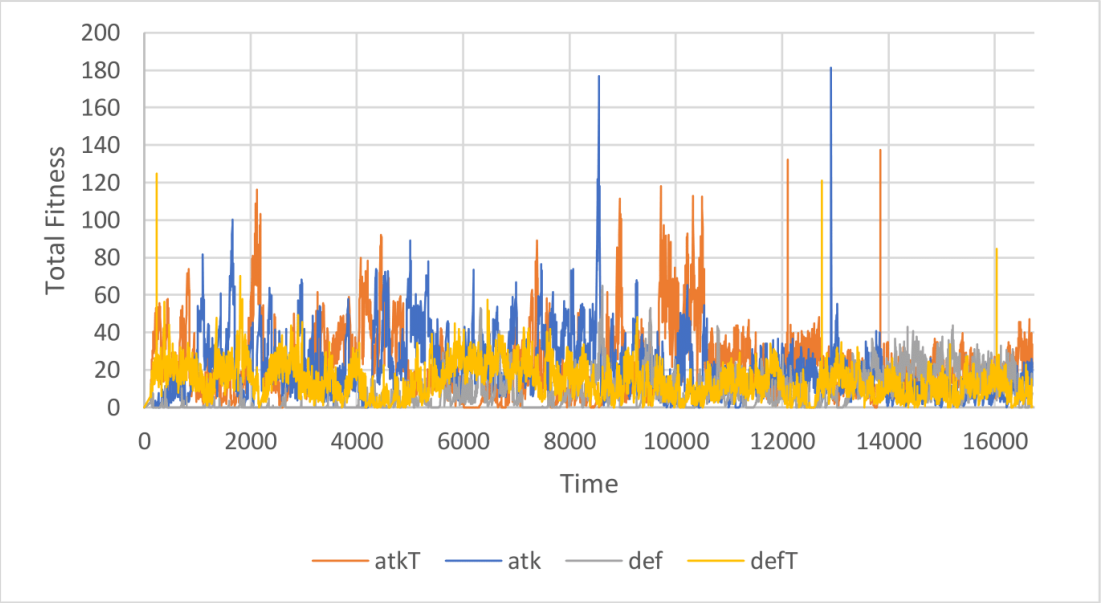

FIGURE 4.32: Total Fitness per Type of Family

And we cannot say that it was in a short amount of time, as we can see in 4.30. Although we can verify that during that period those families dominated in number, their mean fitness is only slightly superior to the ones that are supposed to thrive 4.31, on the other hand, the total fitness 4.32 shown leads us to think that only one or two individuals performed completely abnormally and obtained results far superior to the expected ones, leading to them reproducing a lot during their lifetime.

Chapter 5

Discussion

The results obtained from the Template Generation experiment are within the expectations with the families that prevail in each scenario, mostly showing what we think would be the most important genes to prevail. For example, in scenario 1, where prey is passive and only runs, from the physical genes the most important genes would be Attack (At) and Hunger (Hu). Although there exists the possibility of appearing families with some gene differences this is beneficial overall, because it allows a better range of problem solving for the continuous execution.

In the Real-Time Evolution, the results as stated in the previous section, for the two experiments that aim at replicating the results of the generic GA, the Template Generation, the results obtained were the expected ones. On the other hand the results from the swap and reverse swap experiments were not as clean cut as expected. On the swap experiment, although results were the expected ones, the spikes in total fitness 4.27 lead us to think that the Fitness function is either not sufficiently complex to deal with the present scenarios, the ones where the environment changes, or the weights presented are not the most optimal although to achieve the proposed weights there was extensive testing and a couple of the results can be checked in the Annex's A. These types of spikes also show up in the Reverse Swap experiment and influence the results in an unexpected way. One more thing to note is that when the environment is hostile, there is always a couple

of aggressive families present that seem to slightly thrive against expectations, this also seems to indicate that the fitness functions weights are unbalanced.

As a way to demonstrate how the algorithm works we suggest the following example scenario, where we use the formulas presented in 3.1, 3.2, 3.3 and 3.4, applied to an example iteration that we'll call iteration *X*, that has a total fitness of Ft = 359 and a previous total fitness of Ftpre = 300 in the community. In this iteration the current families in the templates are AA with Ft = 0, AB with Ft = 100, BB with Ft = 50, BA with Ft = 75, CC with Ft = 30, CA with Ft = 15, CB with Ft = 70 and CD with Ft = 19. These families have also have the following total number of creatures: AA - 0, AB - 5, BB - 3, BA - 7, CC - 4, CA - 4, CB - 5 and CD - 2. The family selected to spawn in the previous iteration was the AB family, therefore it receives 0 increment, meaning that preδ and curδ are 0. The preδ values given to the example families are equal to the current one for simplicity purposes. This gives us the following results:

	Families							
	AA	AB	BB	BA	CC	CA	CB	CD
Total Fitness	0	100	50	75	30	15	70	19
Number of Creatures	0	5	3	7	4	4	5	2
Mean Fitness	0	20	16.6	10.7	7.5	3.75	14	9.5
preδ	0.2075	0	1.4525	1.0375	0.6225	0.415	1.245	0.83
curδ	0.2075	0	1.4525	1.0375	0.6225	0.415	1.245	0.83
P	0.00448	0.21610	0.21075	0.13803	0.09448	0.04948	0.17817	0.12058

Table 5.1: Results of Example

With P meaning probability of being picked for reproduction, we can see that the formula allows previously extinct families to get a cumulative chance to reappear without compromising the current evolutionary direction.

Chapter 6

Post Review, Tests and Results

After obtaining the previous results and discussing them as depicted in the previous chapter we where not satisfied with them and we wanted to narrow down the causes of these unclear results. For that purpose using a different computer setup we manage to change the following values of the experiment: field test size from $1Km^2$ to $10Km^2$, the predator community size from 20 to 200, the prey population from 10 to 100 and we also increase the number of iterations to 15000. This allowed us to obtain the following results in the swap and reverse swap experiments.

6.1 Results

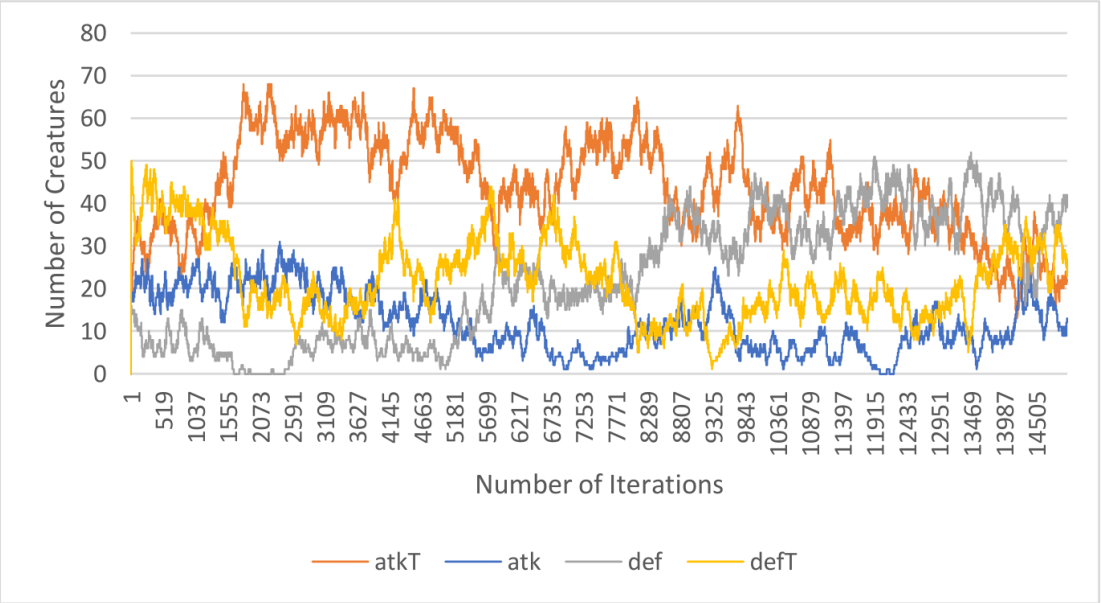

FIGURE 6.1: Number of Creatures per Type of Family per Iteration

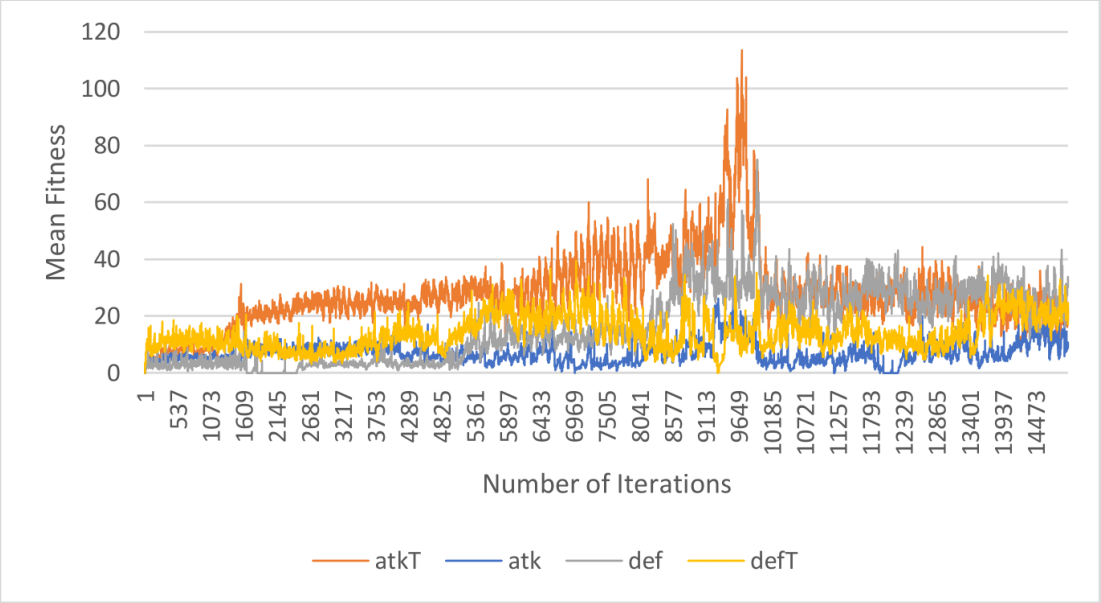

FIGURE 6.2: Mean Fitness per Type of Family per Iteration

The above are the results of the swap experiment with the new setup.

With the increase in parameters we can observe that during the first half of the experiment are within the expectations and when aggressive prey start appearing the dominant type, atkT, starts to decline in population numbers, which happens

between 7800 - 9000, 6.1. During the period where there are still passive prey in the field we can see that their Mean Fitness continues to rise and when there are only aggressive prey they lose that mean fitness abruptly, 6.2. During the second half of the experiment the def and atkT types fight for numbers control but ultimately the def type families thrive.

The following are the results of the reverse swap experiment with the new setup.

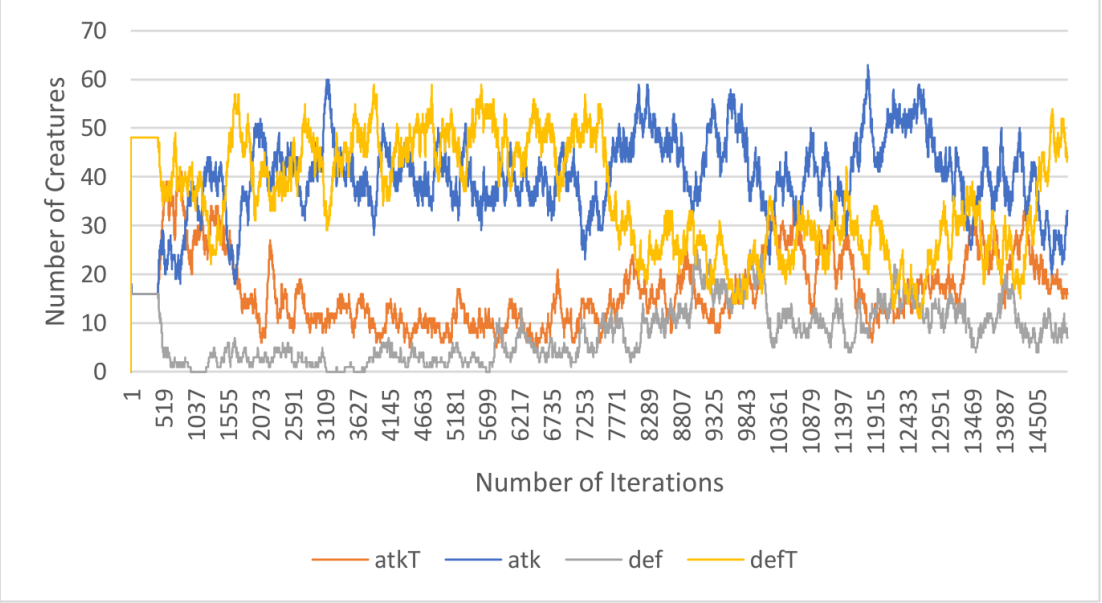

FIGURE 6.3: Number of Creatures per Type of Family per Iteration

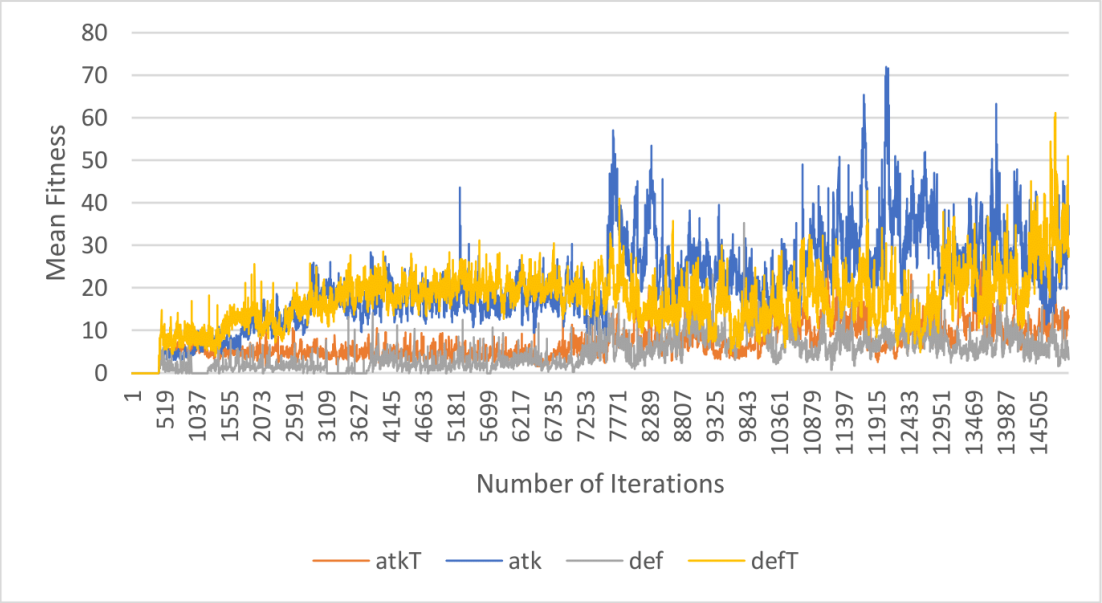

FIGURE 6.4: Mean Fitness per Type of Family per Iteration

We can see in the first half of the experiment the same type of fight that happened in the swap experiments' second half, this type between defT and atk types. With the defT having more presence in the community than the atk, 6.3. In the second half, right when the numbers of hostile prey start to dwindle, the defT type of families decays, leaving the atk types to thrive. Their mean fitness's show the struggle for dominance in the first half and the rise to prominence, of the atk types, in the second half. In the final part of the experiment there is a sudden rise in the defT types, due to a good mutation in one of the families, we can not ascertain if that leads to only a spike and the atk types continue to dominate or the defT becomes a contender for dominance again.

From these results we can verify that the byproduct competition is still present and it can be verified every time the aggressive prey is present. The grouping behaviour is also still present but its impact is diminished due to the population size. The difference in environments can be clearly seen in the charts 6.1 and 6.3.

Chapter 7

Conclusion

From this experiment we can conclude that the algorithm adapts according to inputs from the environment, in this case aggressive or passive bots, albeit in some cases not in the most optimal way. That, with increased experiments, the pool of templates to select a suitable solution will increase and therefore increase the performance and capacity to adapt. We can also observe from the results that the algorithm in the swap experiment and reverse swap experiment shows signs of being stuck in a local optima.

Although small in the swap experiment, to pinpoint probable cause of optimization problem, further testing is needed, as the number of runs in each experiment proves to be too small. A couple of reasons for this can be: the population size being too small and/or too constrained in terms of space which makes the grouping behaviour of the prey more likely to happen or the weights balancing in the fitness function.

And with the Post Review experiments we concluded that the population size and space of testing affected those results, although that does not discard the possibility that the fitness function is still not balanced enough for the presented scenario and further testing is needed.

As future work the fitness function will be increased in complexity to see how it impacts the performance and results, tests with human players will also be

conducted to see if the adaptation capabilities demonstrated in the previous tests maintain, since a human player is more unpredictable than a bot.

Also, in future work, increase the number of communities of the same species and apply concepts present in odNEAT on top of this algorithm to see how cooperation between communities affect the evolution and compare performances.

Bibliography

[Andrade et al., 2006] Andrade, G., Ramalho, G., Gomes, A. S., and Corruble, V. (2006). Dynamic game balancing: An evaluation of user satisfaction. *AIIDE*, 6:3–8.

[Apperley, 2006] Apperley, T. H. (2006). Genre and game studies: Toward a critical approach to video game genres. *Simulation & Gaming*, 37(1):6–23.

[Assembly, 2014] Assembly, C. (2014). Alien: Isolation^{TM} for mac and linux - story: Feral interactive. `http://www.feralinteractive.com/en/games/alienisolation/story/`.

[Baker, 1987] Baker, J. E. (1987). Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application*, pages 14–21, Hillsdale, NJ, USA. L. Erlbaum Associates Inc.

[Bakkes et al., 2009] Bakkes, S., Spronck, P., and Van den Herik, J. (2009). Rapid and reliable adaptation of video game ai. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(2):93–104.

[Bontrager et al., 2016] Bontrager, P., Khalifa, A., Mendes, A., and Togelius, J. (2016). Matching Games and Algorithms for General Video Game Playing. *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, pages 122–128.

[Capcom, 1975] Capcom (1975). Gun fight [model 597]. `https://www.arcade-history.com/?n=gun-fight-upright-model-no.-597&page=detail&id=1040`.

[Cowan and Kapralos, 2014] Cowan, B. and Kapralos, B. (2014). A survey of frameworks and game engines for serious game development. *Proceedings - IEEE 14th International Conference on Advanced Learning Technologies, ICALT 2014*, pages 662–664.

[Deb et al., 2002] Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197.

[Freeman, 1998] Freeman, L. M. (1998). *Industrial Applications of Genetic Algorithms*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition.

[FromSoftware, 2018] FromSoftware (2010-2018). ダークソウルシリーズサイト: DARK SOULS Series Site. `https://www.darksouls.jp/`.

[Gavin, 1996] Gavin, A. (1996). Making crash bandicoot - part 6. `https://all-things-andy-gavin.com/2011/02/07/making-crash-bandicoot-part-6/`.

[Goldberg, 1989] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition.

[Hasançebi and Erbatur, 2000] Hasançebi, O. and Erbatur, F. (2000). Evaluation of crossover techniques in genetic algorithm based optimum structural design. *Computers and Structures*, 78(1):435–448.

[Hastings et al., 2009] Hastings, E. J., Guha, R. K., and Stanley, K. O. (2009). Demonstrating automatic content generation in the galactic arms race video game. *Proceedings of the 5th Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE 2009*, pages 189–190.

[Hendrikx et al., 2013] Hendrikx, M., Meijer, S., Van Der Velden, J., and Iosup, A. (2013). Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 9(1):1.

[Hendrix et al., 2018] Hendrix, M., Bellamy-Wood, T., McKay, S., Bloom, V., and Dunwell, I. (2018). Implementing Adaptive Game Difficulty Balancing in Serious Games. *IEEE Transactions on Games*, 1502(c):1–1.

[Holland, 1970] Holland, J. H. (1970). *Hierarchical descriptions, universal spaces and adaptive systems.* University of Illinois Press, Illinois.

[Hunicke and Chapman, 2004] Hunicke, R. and Chapman, V. (2004). AI for dynamic difficulty adjustment in games. *AAAI Workshop Challenges in Game Artififical Intelligence*, pages 91–96.

[Jinghui et al., 2005] Jinghui, Z., Xiaomin, H., Min, G., and Jun, Z. (2005). Comparison of performance between different selection strategies on simple genetic algorithms. *Proceedings - International Conference on Computational Intelligence for Modelling, Control and Automation, CIMCA 2005 and International Conference on Intelligent Agents, Web Technologies and Internet*, 2(January):1115–1120.

[Lach, 2018] Lach, E. (2018). New Adaptations for Evolutionary Algorithm Applied to Dynamic Difficulty Adjustment System for Serious Game.

[Llc, 2019] Llc, E. O. (2019). The demographics of video gaming: Earnest. `https://www.earnest.com/blog/the-demographics-of-video-gaming/`.

[Loshchilov et al., 2011] Loshchilov, I., Schoenauer, M., and Sebag, M. (2011). Not all parents are equal for MO-CMA-ES. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6576 LNCS:31–45.

[Magazine, 2008] Magazine, E. (2008). Gabe Newell writes for Edge - Edge Magazine. `https://archive.is/20120909153756/http://www.next-gen.biz/opinion/gabe-newell-writes-edge`.

[Magerko et al., 2006] Magerko, B., Stensrud, B. S., and Holt, L. S. (2006). Bringing the schoolhouse inside the box-a tool for engaging, individualized training. Technical report, SOAR TECHNOLOGY INC ANN ARBOR MI.

[Malla Osman et al., 2015] Malla Osman, Z., Dupire, J., Mader, S., Cubaud, P., and Natkin, S. (2015). Monitoring player attention: A non-invasive measurement method applied to serious games. *Entertainment Computing*.

[Mühlenbein and Schlierkamp-Voosen, 1993] Mühlenbein, H. and Schlierkamp-Voosen, D. (1993). Predictive Models for the Breeder Genetic Algorithm I. Continuous Parameter Optimization. *Evolutionary Computation*, 1(1):25–49.

[Newzoo, 2018] Newzoo (2018). 2018 Global Games Market Report.

[Olesen et al., 2008] Olesen, J. K., Yannakakis, G. N., and Hallam, J. (2008). Real-time challenge balance in an rts game using rtneat. In *2008 IEEE Symposium On Computational Intelligence and Games*, pages 87–94. IEEE.

[Rajakumar, 2013] Rajakumar, B. R. (2013). Static and adaptive mutation techniques for genetic algorithm: A systematic comparative analysis. *International Journal of Computational Science and Engineering*, 8(2):180–193.

[Silva et al., 2015] Silva, F., Urbano, P., Correia, L., and Christensen, A. L. (2015). odneat: An algorithm for decentralised online evolution of robotic controllers. *Evolutionary Computation*, 23(3):421–449.

[Srinivas M. & Patnaik LM, 2006] Srinivas M. & Patnaik LM (2006). Genetic Algorithms: A Survey. Computer. *Neural Networks in a Softcomputing Framework Springer*, 27(6).

[Stanley et al., 2005] Stanley, K. O., Cornelius, R., Miikkulainen, R., D'silva, T., and Gold, A. (2005). Real-time learning in the NERO video game. *Proceedings of*

the 1st Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE 2005, 2003:159–162.

[Stanley and Miikkulainen, 2002] Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies %J Evolutionary Computation. *Evolutionary Computation*, 10(2):99–127.

[Thatgamecompany, 2012] Thatgamecompany (2012). Journey™. =https://www.playstation.com/pt-br/games/journey-ps4/.

[Valve, 2008] Valve (2008). Left4dead. `https://web.archive.org/web/20090327034239/http://www.l4d.com:80/info.html`.

[Vikhar, 2016] Vikhar, P. A. (2016). Evolutionary Algorithms : A Critical Review and its Future Prospects. *2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC)*, pages 261–265.

[Yannakakis et al., 2013] Yannakakis, G. N., Spronck, P., Loiacono, D., and André, E. (2013). Player modeling. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Appendices

Appendix A

Fitness Function tests

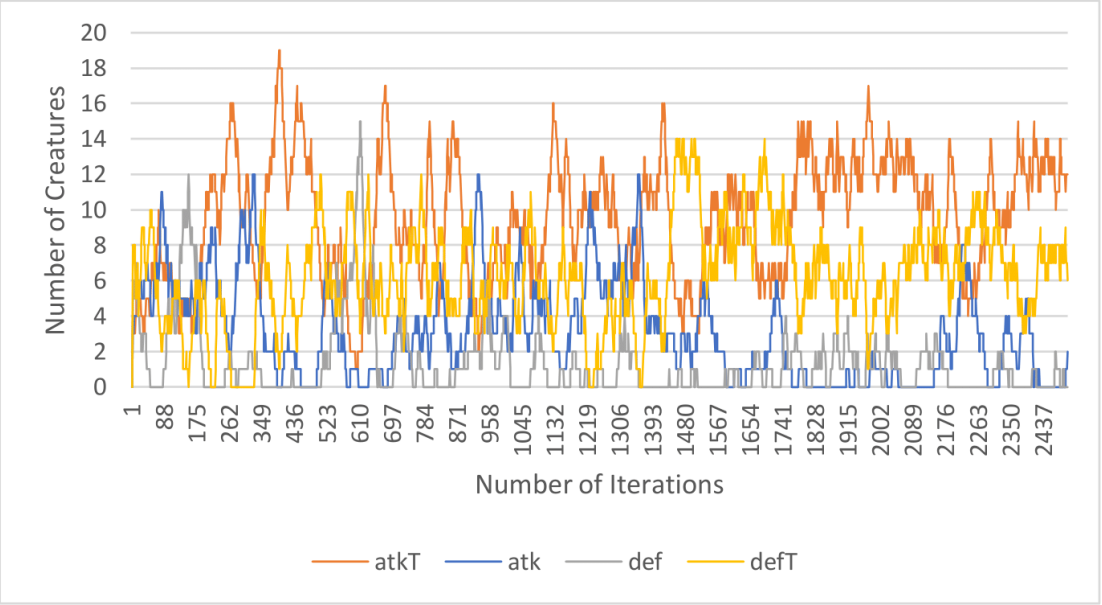

FIGURE A.1: Swing with fit: Nprk - 6, Atk - 0.75, Dist - 0.3, Ta - 0.3

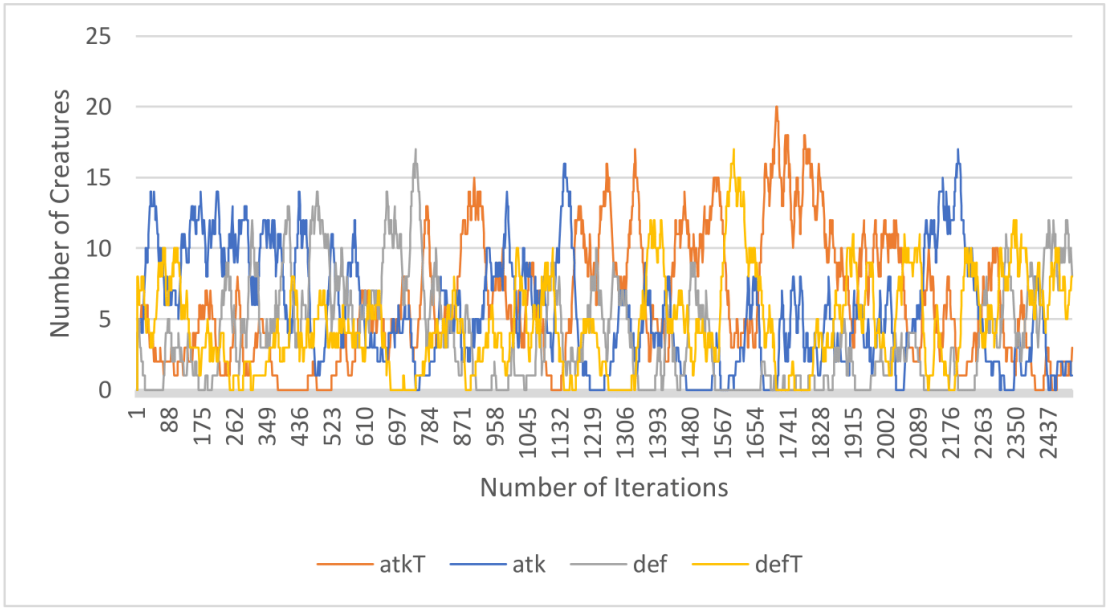

FIGURE A.2: Swing with fit: Nprk - 6, Atk - 0.6, Dist - 0.3, Ta - 0.3

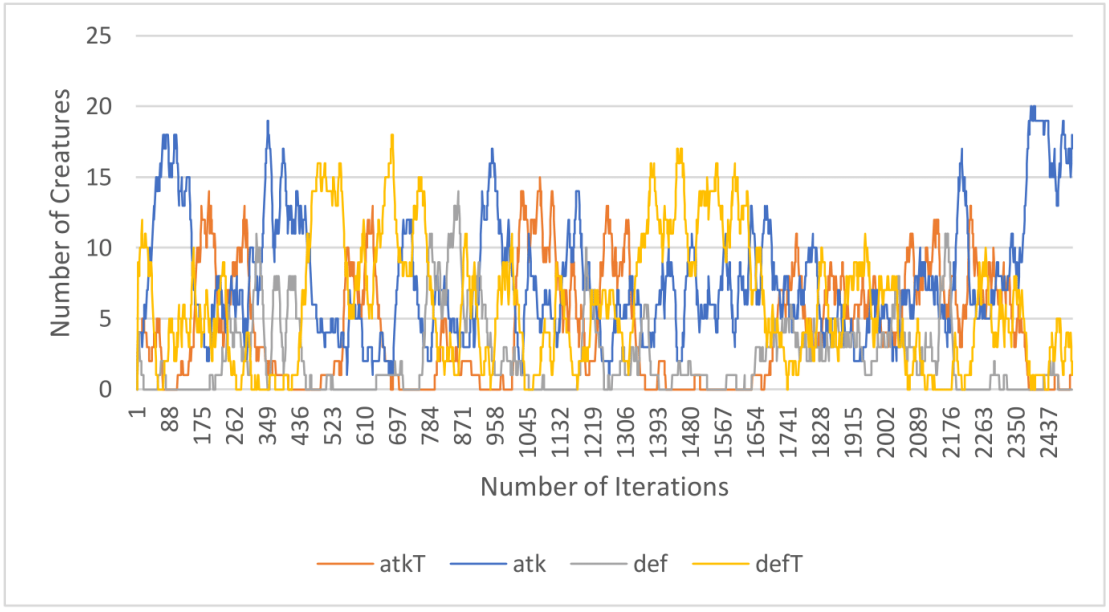

FIGURE A.3: Swing with fit: Nprk - 6, Atk - 2.5, Dist - 0.6, Ta - 0.6

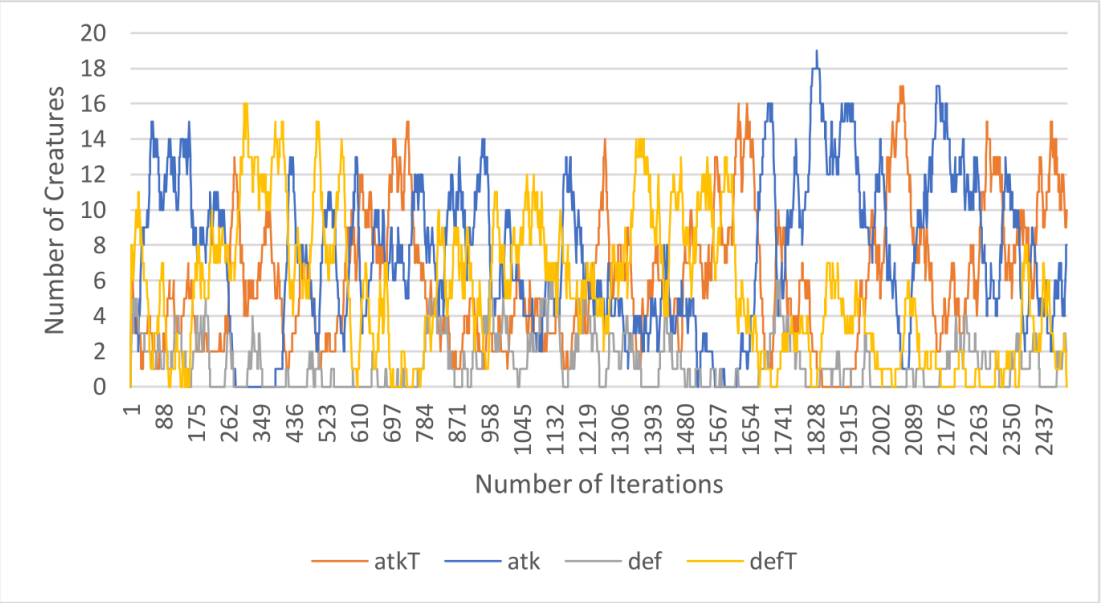

FIGURE A.4: Swing with fit: Nprk - 6, Atk - 1.5, Dist - 0.6, Ta - 0.6

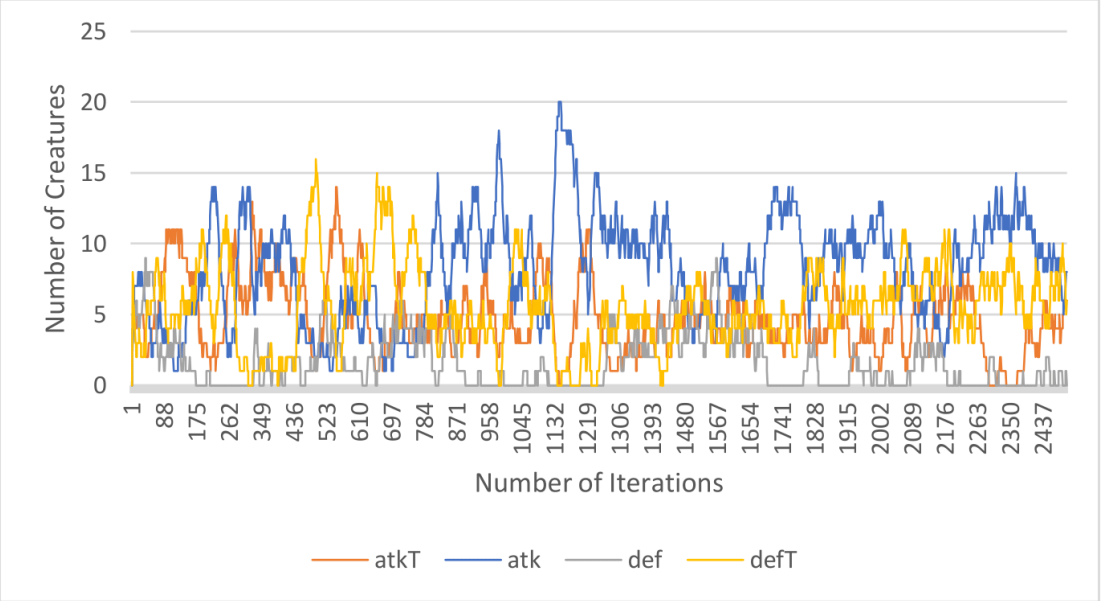

FIGURE A.5: Swing with fit: Nprk - 6, Atk - 0.9, Dist - 0.6, Ta - 0.6

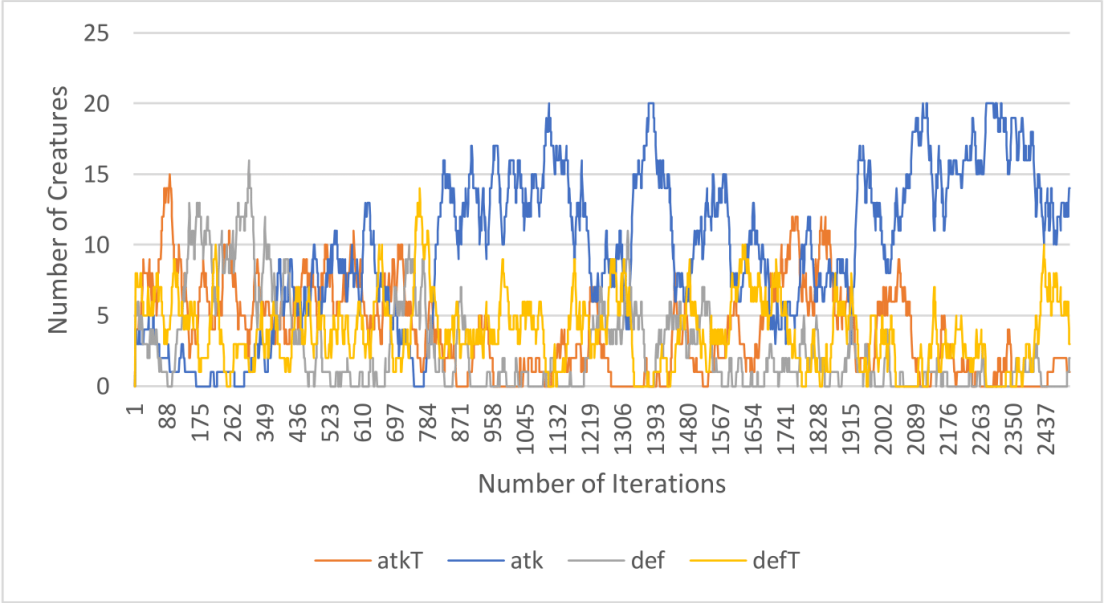

FIGURE A.6: Swing with fit: Nprk - 6, Atk - 0.75, Dist - 0.6, Ta - 0.6

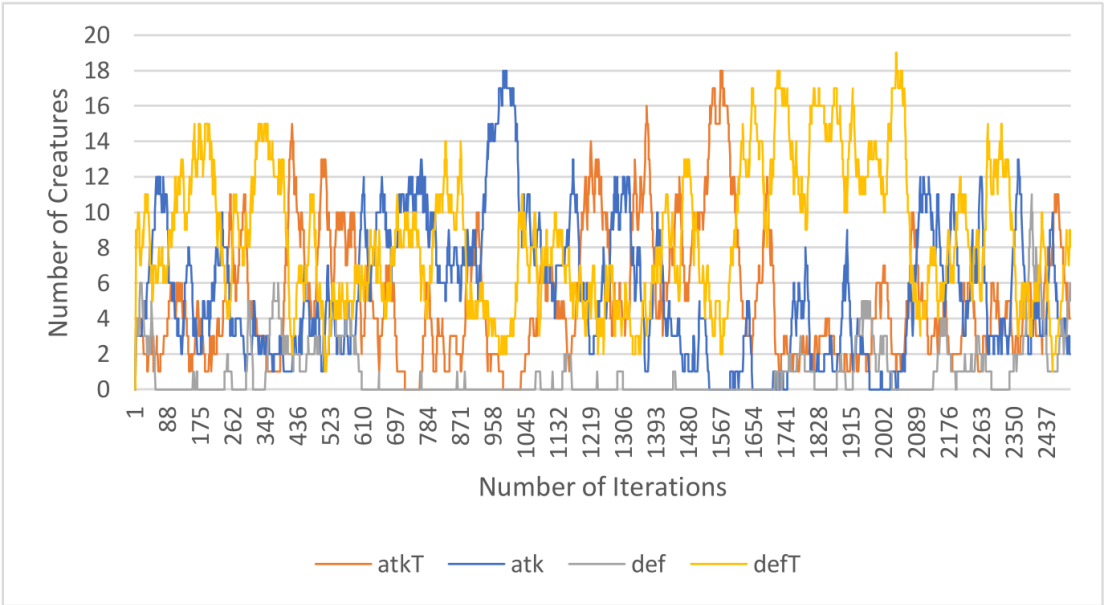

FIGURE A.7: Swing with fit: Nprk - 6, Atk - 0.7, Dist - 0.6, Ta - 0.6

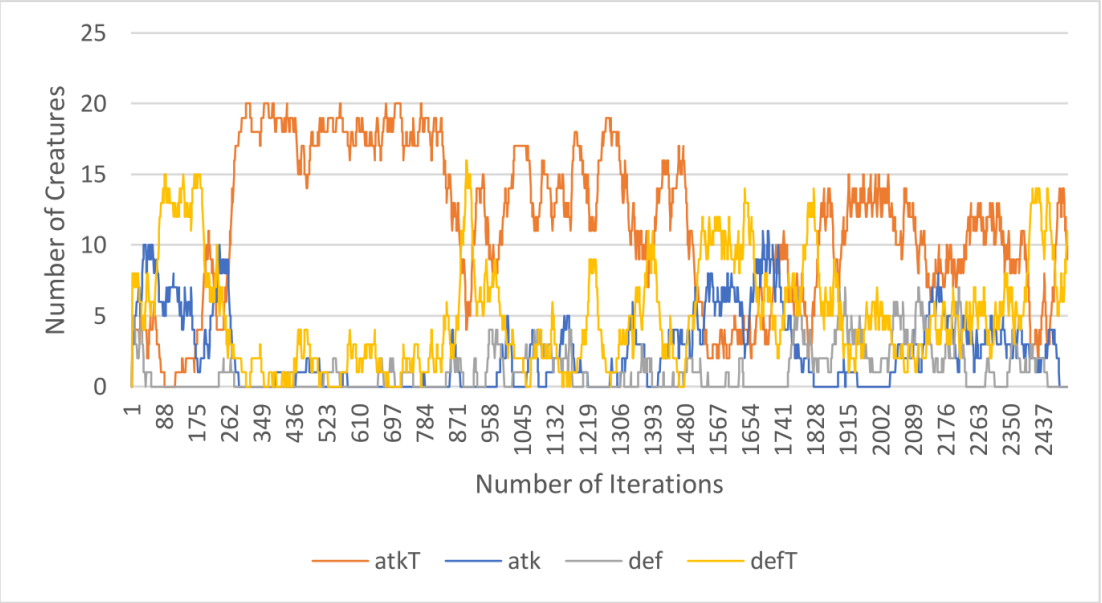

FIGURE A.8: Swing with fit: Nprk - 6, Atk - 0.5, Dist - 0.6, Ta - 0.6

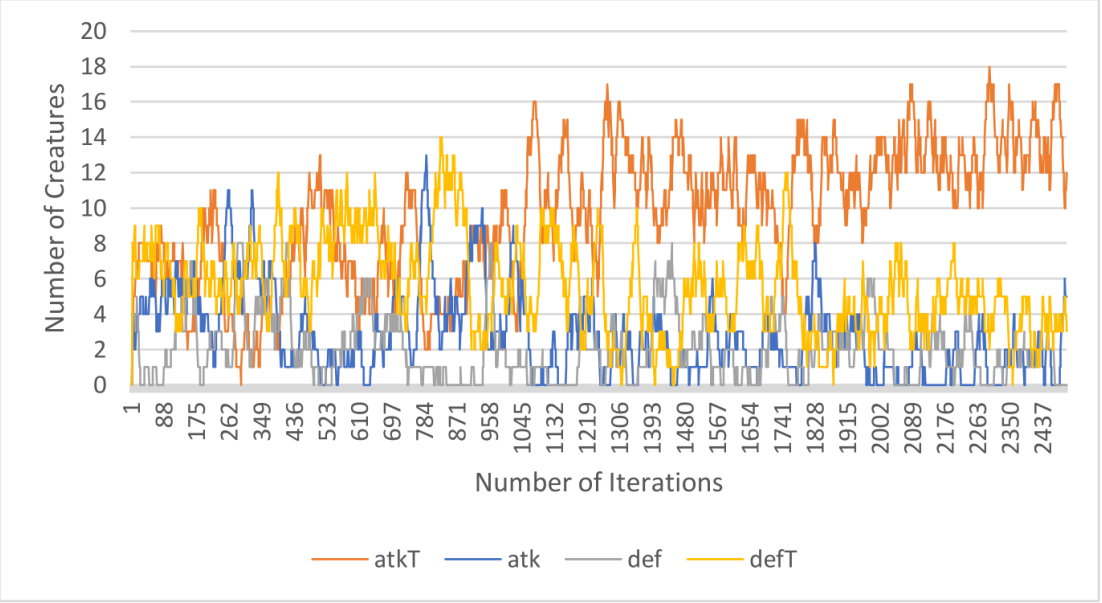

FIGURE A.9: Swing with fit: Nprk - 6, Atk - 0.5, Dist - 0.3, Ta - 0.2

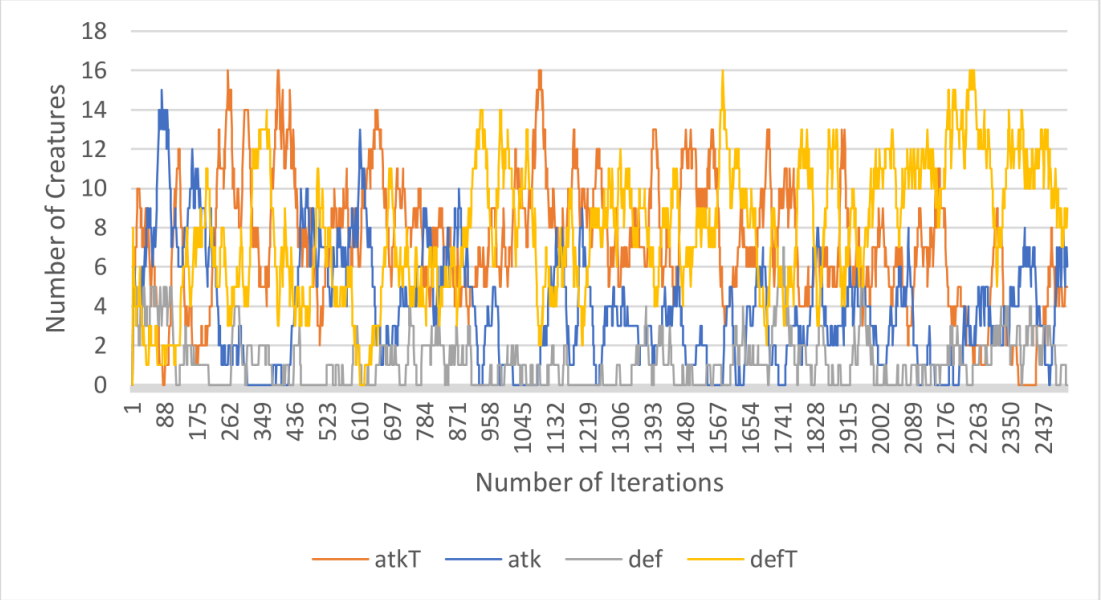

FIGURE A.10: Swing with fit: Nprk - 6, Atk - 0.5, Dist - 0.25, Ta - 0.25

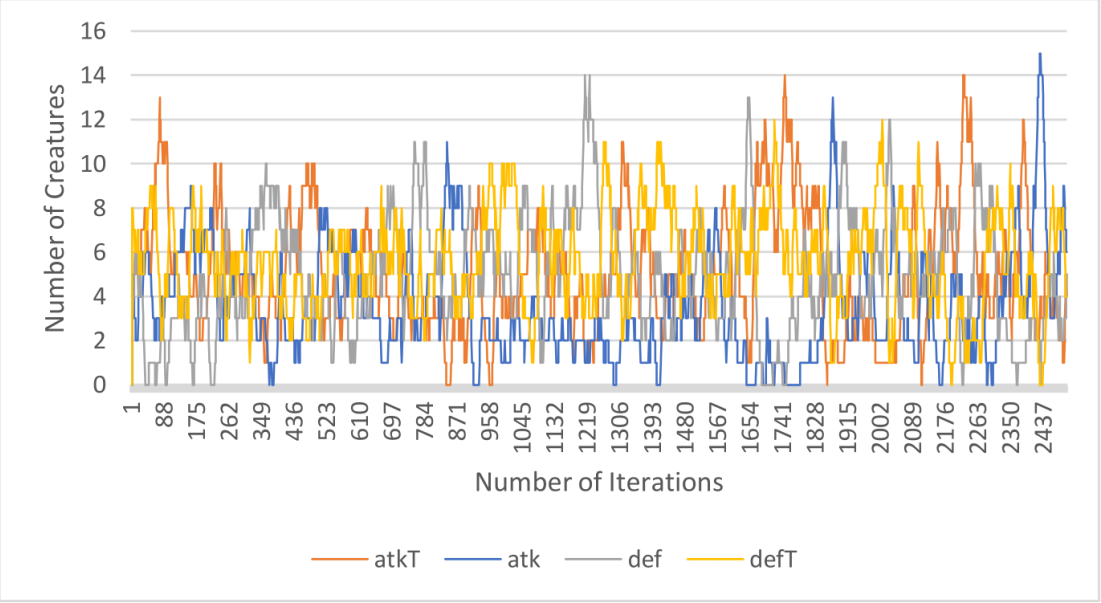

FIGURE A.11: Swing with fit: Nprk - 6, Atk - 0.4, Dist - 0.6, Ta - 0.0

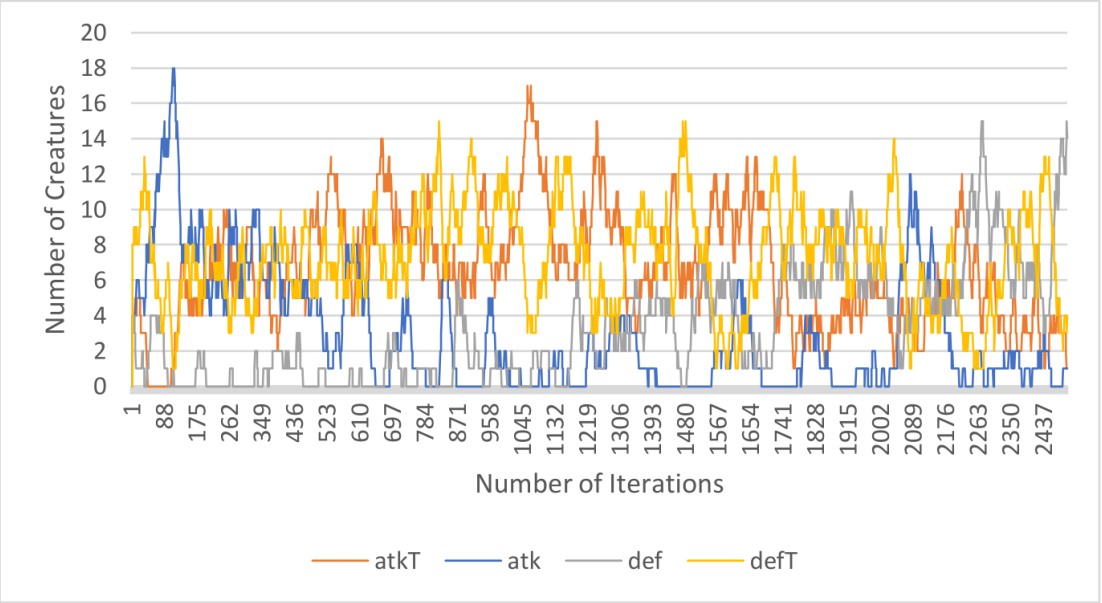

FIGURE A.12: Swing with fit: Nprk - 6, Atk - 0.4, Dist - 0.3, Ta - 0.3

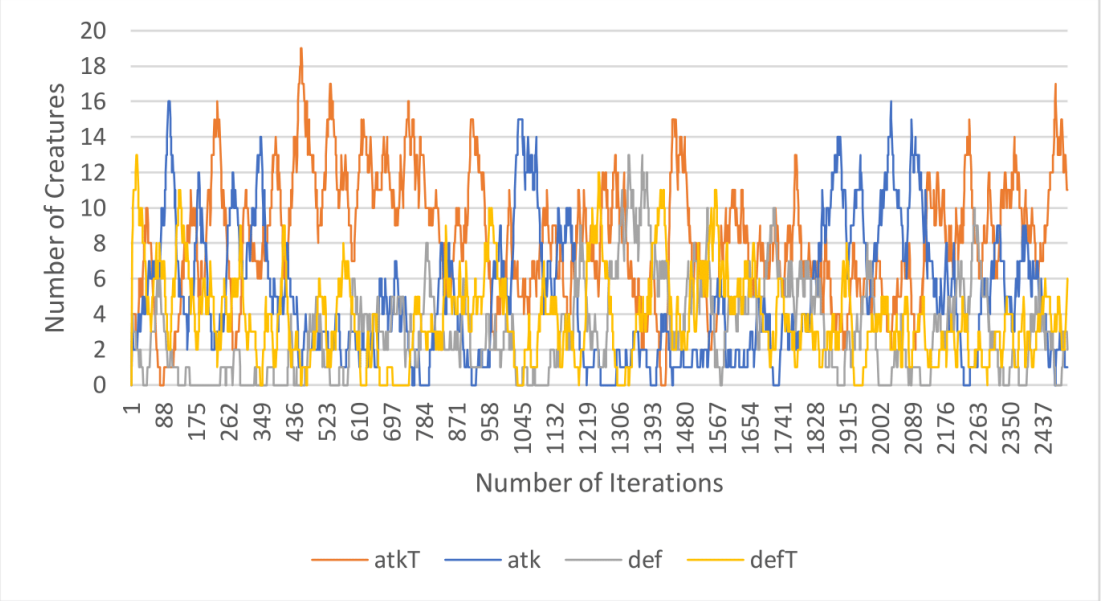

FIGURE A.13: Swing with fit: Nprk - 6, Atk - 0.4, Dist - 0.3, Ta - 0.2

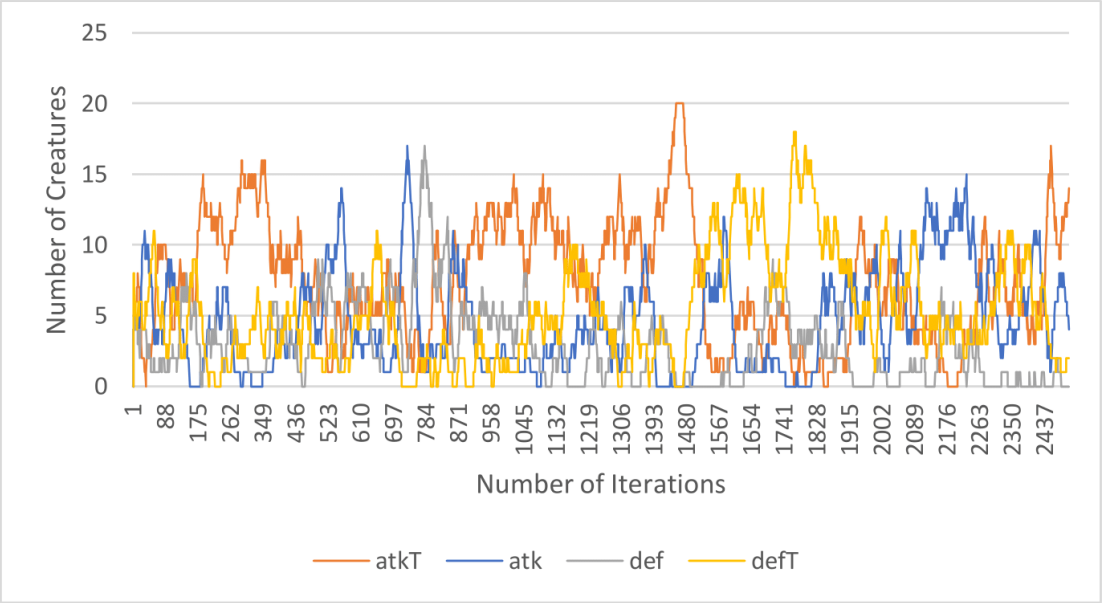

FIGURE A.14: Swing with fit: Nprk - 6, Atk - 0.4, Dist - 0.3, Ta - 0.1

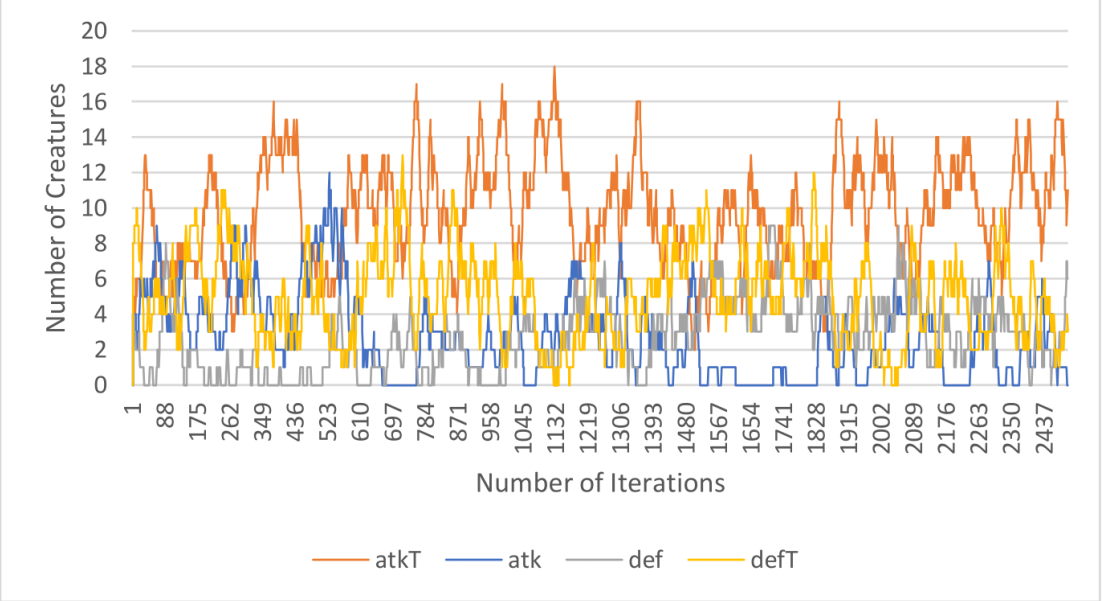

FIGURE A.15: Swing with fit: Nprk - 6, Atk - 0.4, Dist - 0.3, Ta - 0.05

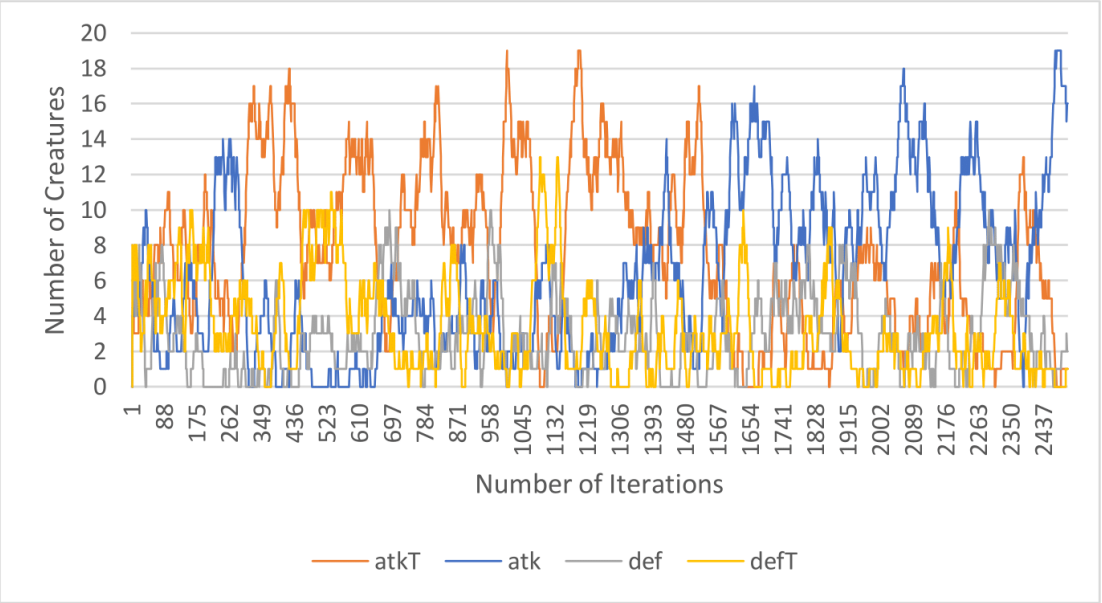

FIGURE A.16: Swing with fit: Nprk - 6, Atk - 0.4, Dist - 0.3, Ta - 0.0

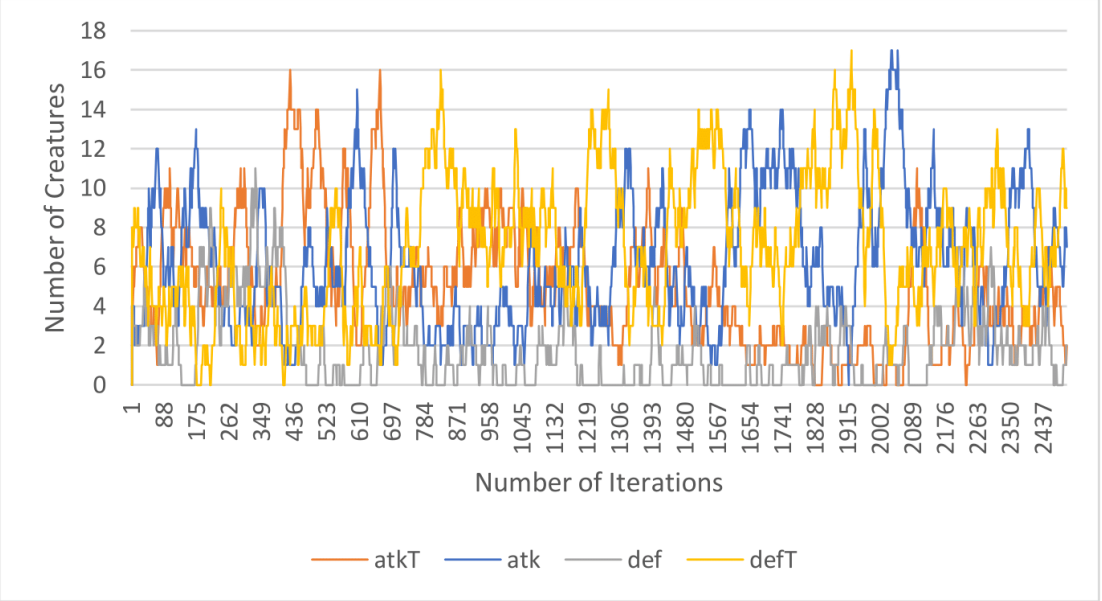

FIGURE A.17: Swing with fit: Nprk - 6, Atk - 0.4, Dist - 0.25, Ta - 0.25

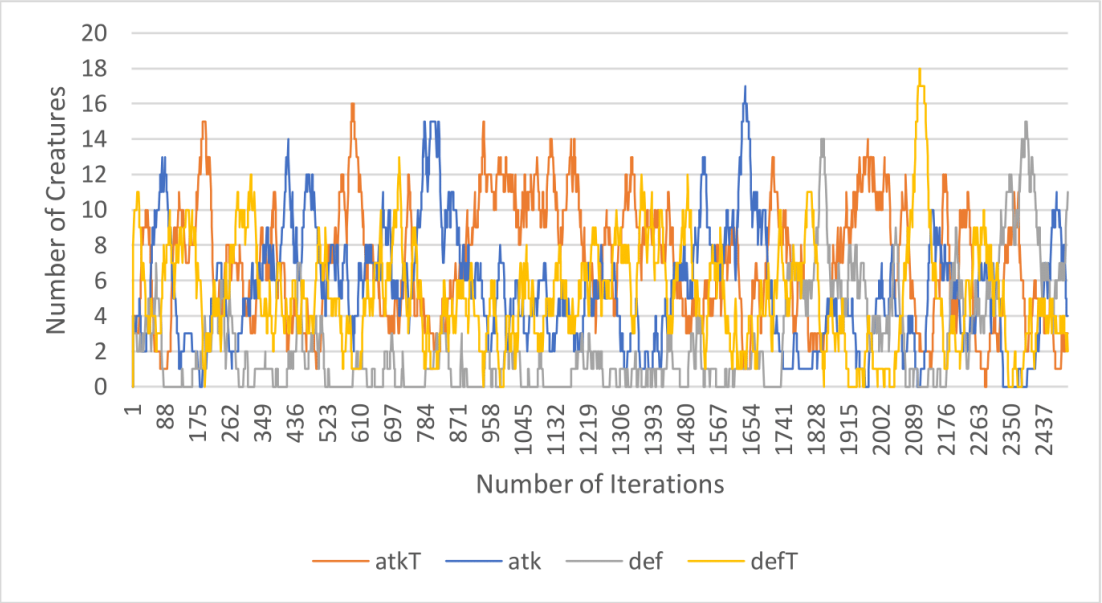

FIGURE A.18: Swing with fit: Nprk - 6, Atk - 0.4, Dist - 0.2, Ta - 0.2

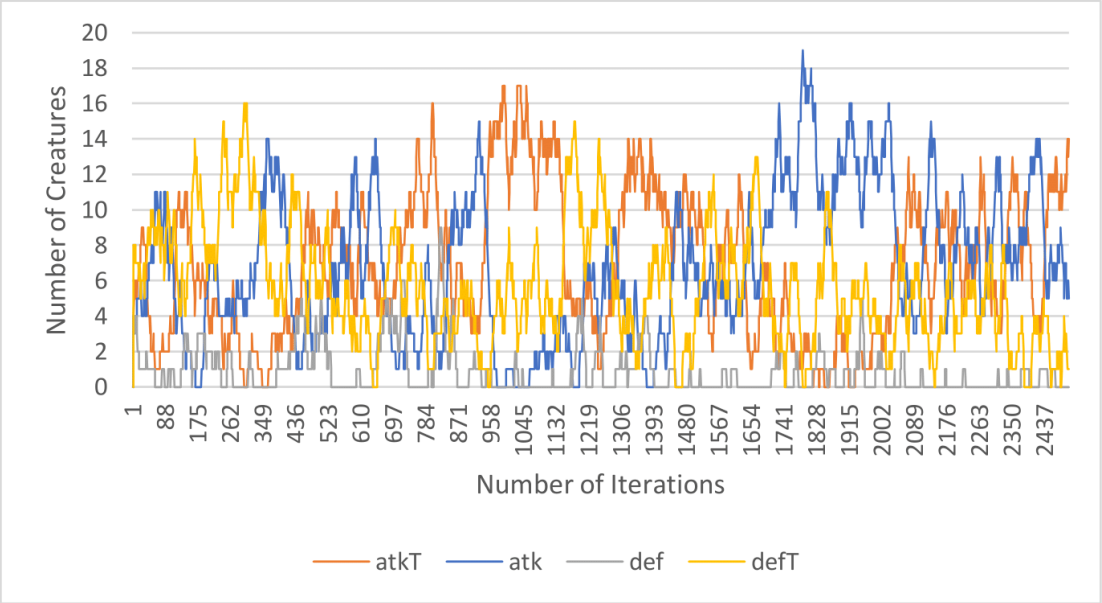

FIGURE A.19: Swing with fit: Nprk - 6, Atk - 0.4, Dist - 0.1, Ta - 0.3

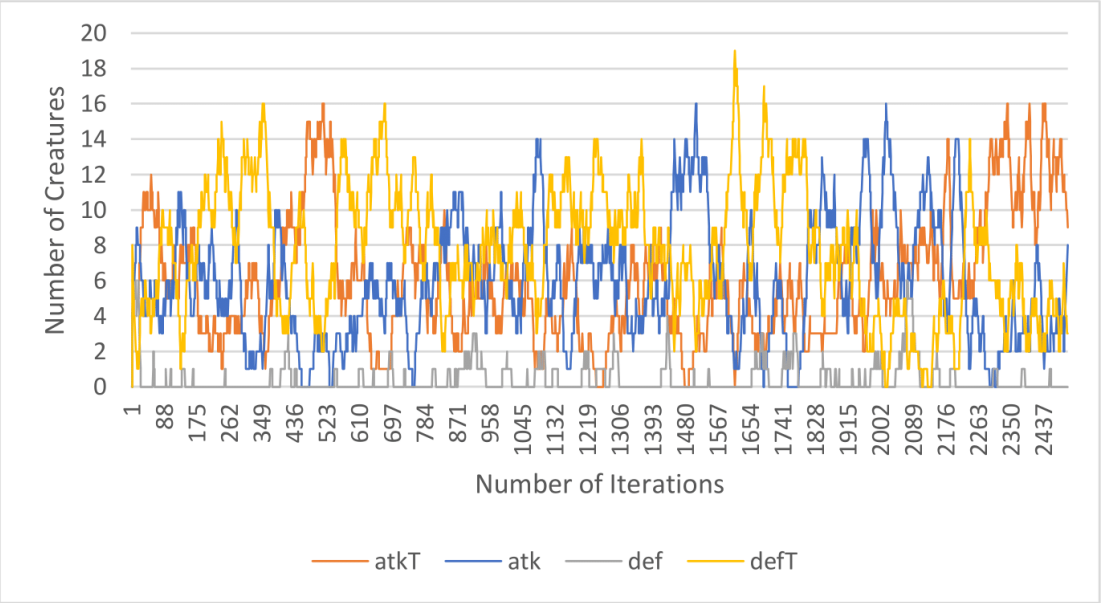

FIGURE A.20: Swing with fit: Nprk - 6, Atk - 0.4, Dist - 0.05, Ta - 0.3

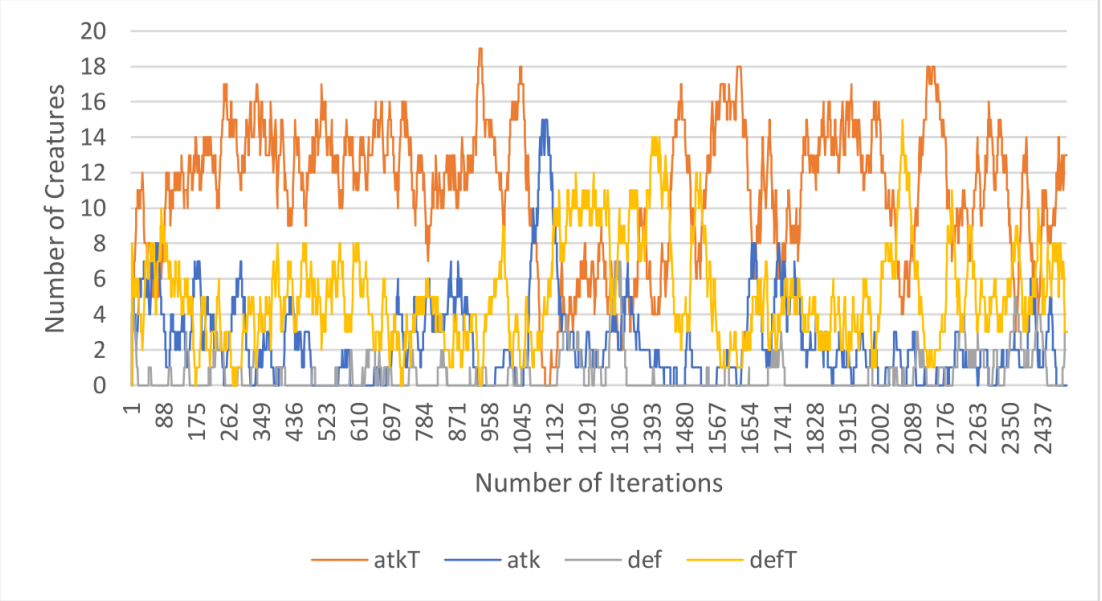

FIGURE A.21: Swing with fit: Nprk - 6, Atk - 0.4, Dist - 0.0, Ta - 0.3

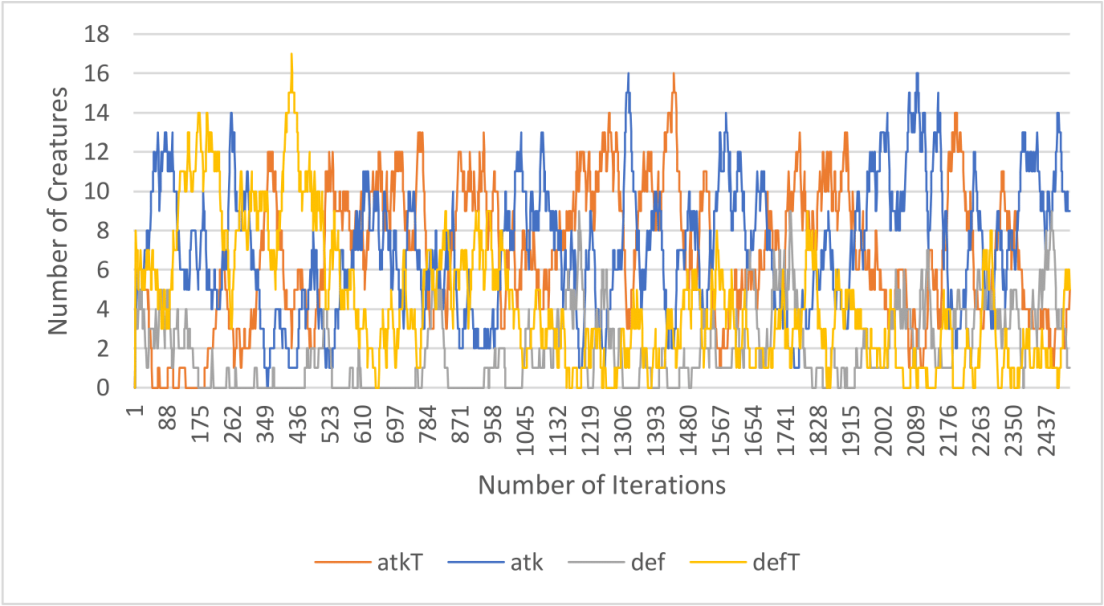

FIGURE A.22: Swing with fit: Nprk - 6, Atk - 0.2, Dist - 0.4, Ta - 0.4

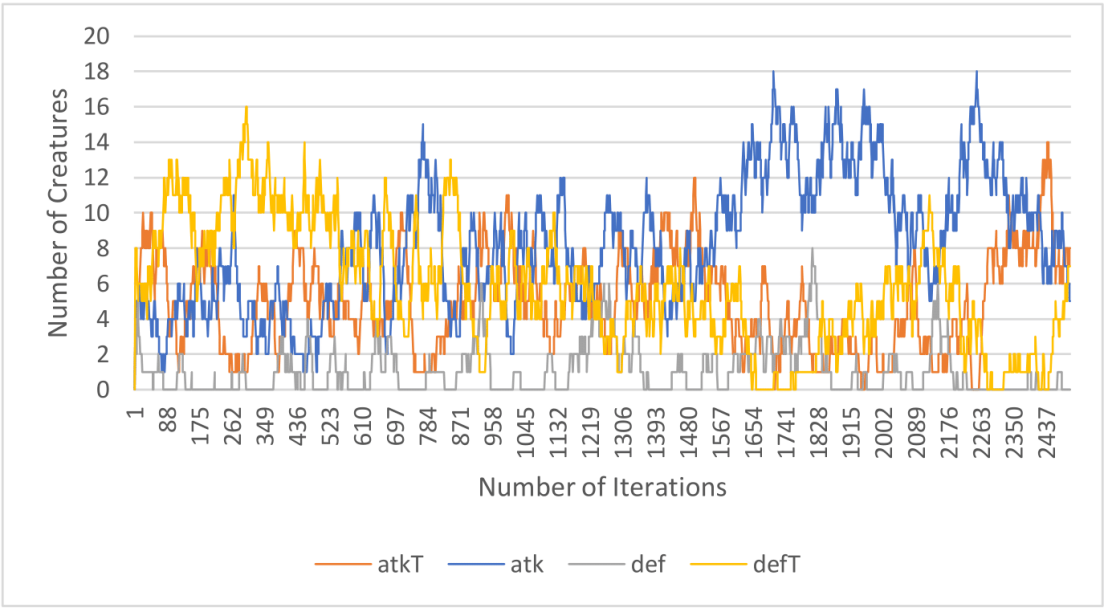

FIGURE A.23: Swing with fit: Nprk - 3, Atk - 0.1, Dist - 0.6, Ta - 0.6

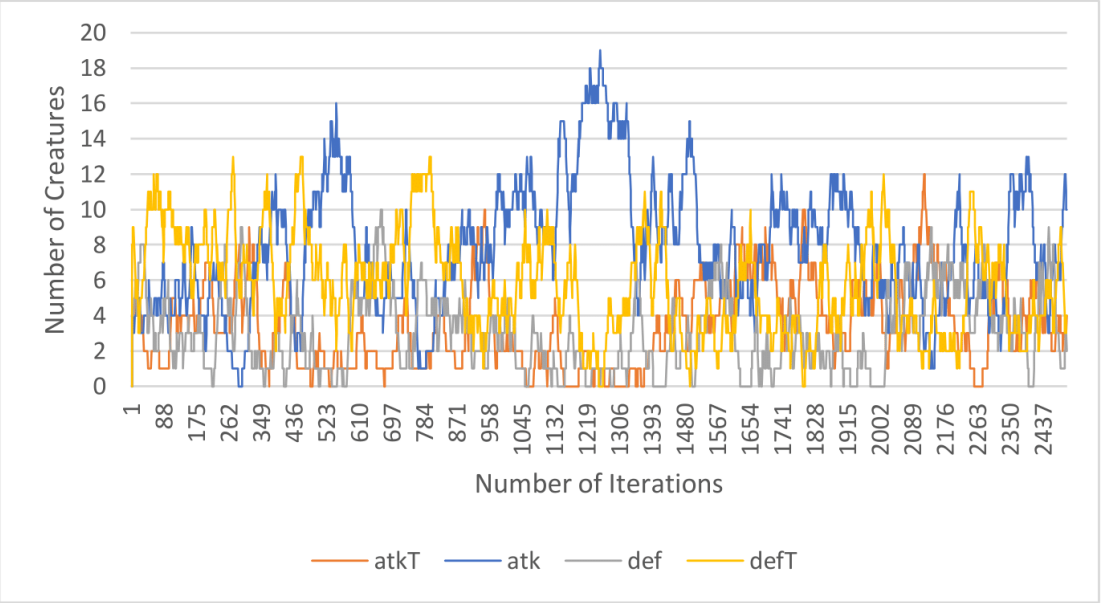

FIGURE A.24: Swing with fit: Nprk - 6, Atk - 0.1, Dist - 0.6, Ta - 0.0

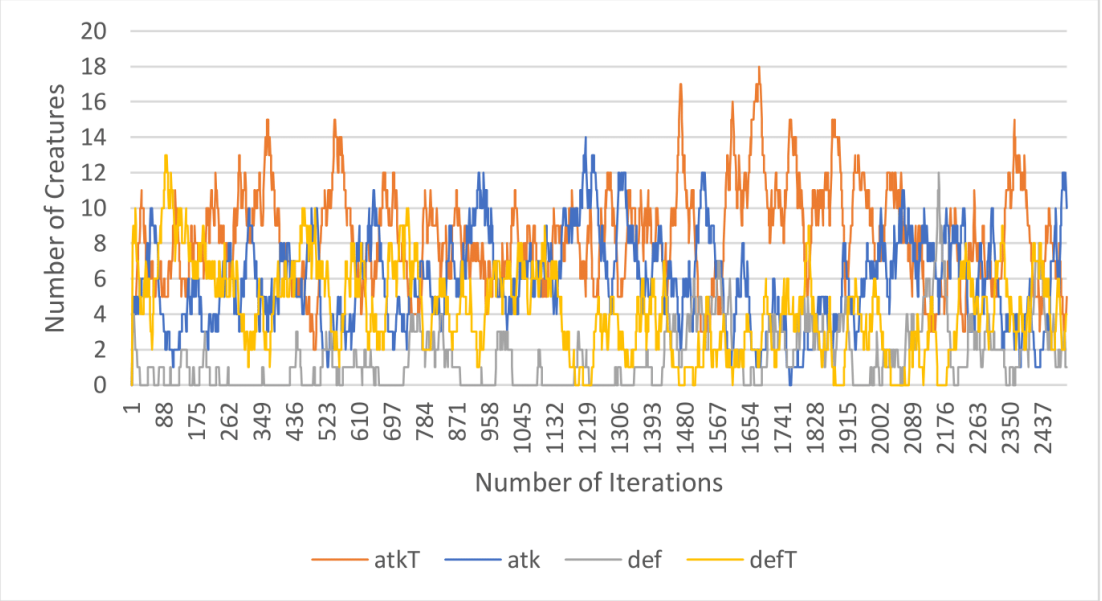

FIGURE A.25: Swing with fit: Nprk - 6, Atk - 0.1, Dist - 0.4, Ta - 0.6

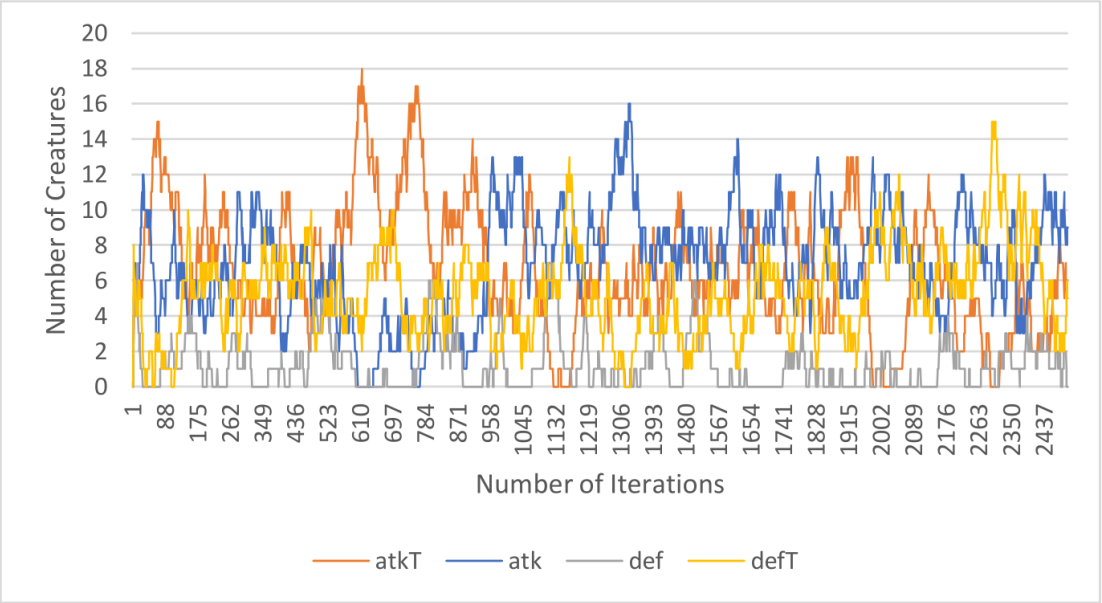

FIGURE A.26: Swing with fit: Nprk - 6, Atk - 0.1, Dist - 0.3, Ta - 0.6

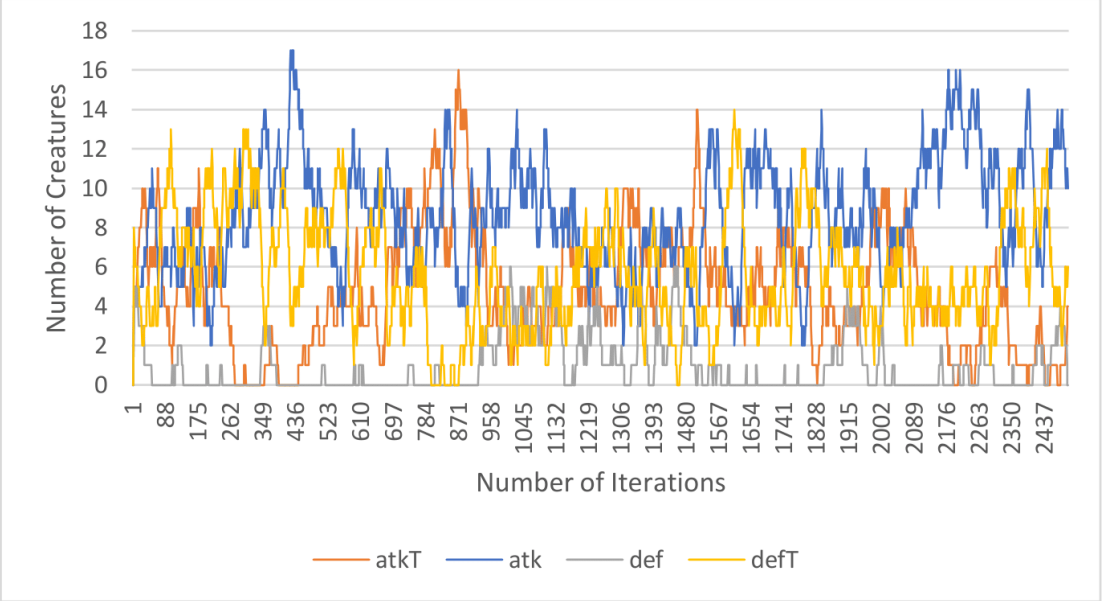

FIGURE A.27: Swing with fit: Nprk - 6, Atk - 0.1, Dist - 0.25, Ta - 0.6

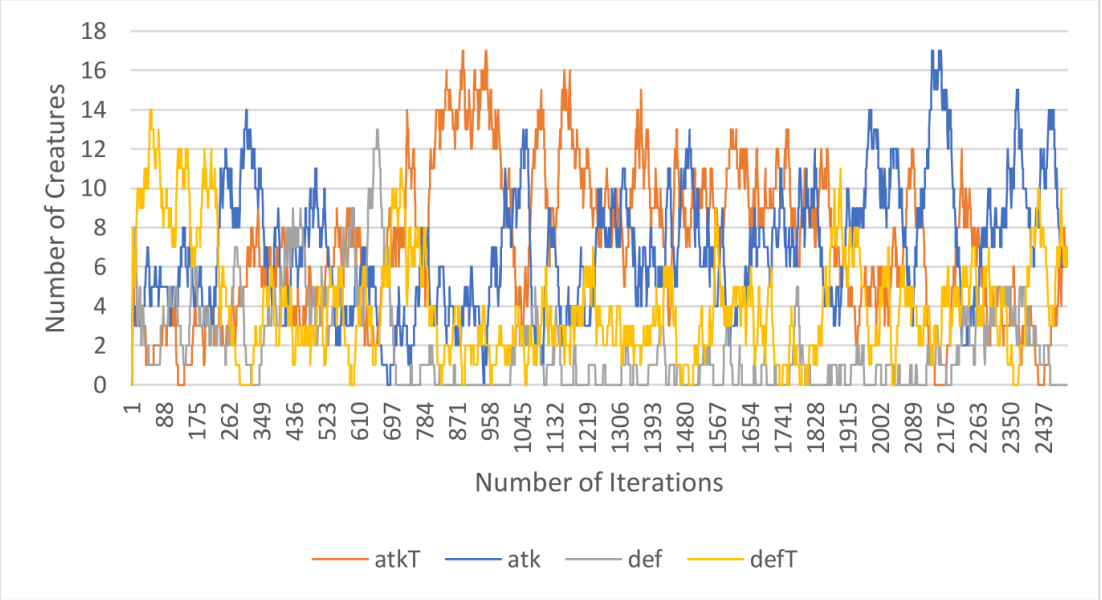

FIGURE A.28: Swing with fit: Nprk - 6, Atk - 0.1, Dist - 0.15, Ta - 0.6

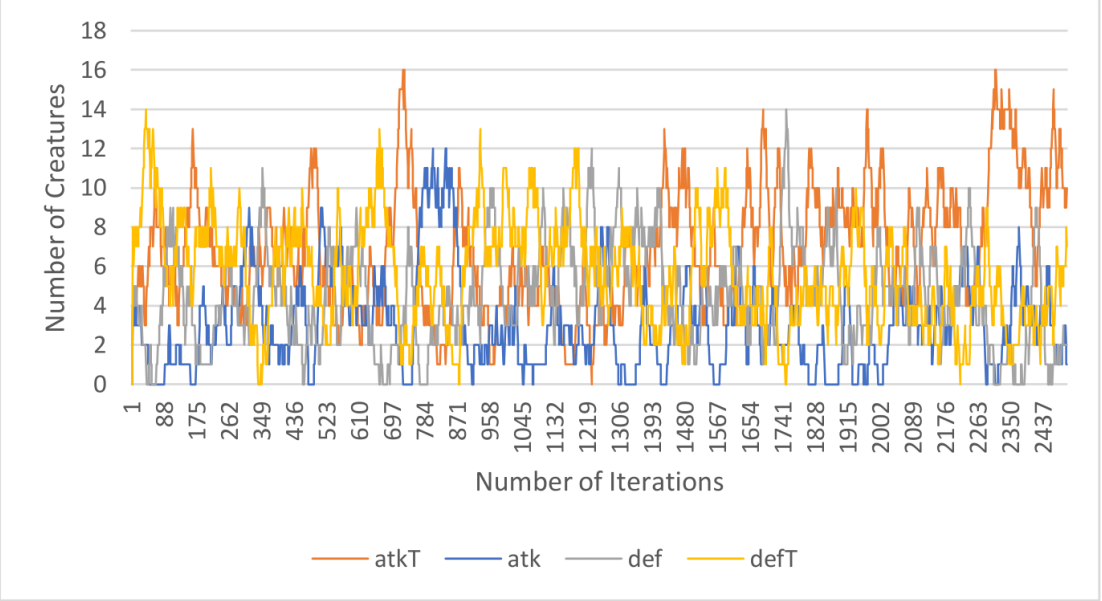

FIGURE A.29: Swing with fit: Nprk - 6, Atk - 0.1, Dist - 0.06, Ta - 0.6