Universität Heidelberg
Institut für Informatik
Sommersemester 2019

Seminar Artificial Intelligence for Games
Dozent: Prof. Dr. Ullrich Köthe

Seminararbeit

Evolutionary Algorithms for Controllers in Games

Name: Oliver Mautschke
Matrikelnummer: 3538495
Studiengang: Angewandte Informatik (Master) (3. Fachsemester)
Email: olivermautschke@gmail.com
Datum der Abgabe: July 4, 2019

Hiermit versichere ich, **Oliver Mautschke**, dass ich die Seminararbeit mit dem Titel **Evolutionary Algorithms for Controllers in Games** im **Seminar Artificial Intelligence for Games** im **Sommersemester 2019** bei Prof. Dr. Ullrich Köthe selbstständig und nur mit den in der Arbeit angegebenen Hilfsmitteln verfasst habe. Zitate sowie der Gebrauch fremder Quellen, Abbildungen, Texte und Hilfsmittel habe ich nach den Regeln guter wissenschaftlicher Praxis eindeutig als solche gekennzeichnet.

Mir ist bewusst, dass ich fremde Texte und Textpassagen nicht als meine eigenen ausgeben darf und dass ein Verstoß gegen diese Grundregel des wissenschaftlichen Arbeitens als Täuschungs- und Betrugsversuch gilt, der entsprechende Konsequenzen nach sich zieht. Diese bestehen in der Bewertung der Prüfungsleistung mit “nicht ausreichend” (5,0) sowie ggf. weiteren Maßnahmen.

Außerdem bestätige ich, dass diese Arbeit in gleicher oder ähnlicher Form noch bei keiner anderen Prüfung vorgelegt wurde.

Heidelberg, den 4. Juli 2019 _____

Abstract

This paper deals with the usage of evolutionary algorithms, especially genetic algorithms in games. The concept of genetic algorithms and evolution will be described and the approaches NEAT and EBT will be presented. Then, the paper shows how this algorithm can be used in the games Super Mario. It will be discussed, what is necessary for a good setup using a genetic algorithm and how to evaluate results from a GA. Three different approaches will be shown including EBT, NEAT which is an neuroevolutionary algorithm and an interactive approach giving the user the possibility to lead the evolution.

Contents

1 Introduction

This seminar paper deals with the usage of evolutionary algorithms, especially genetic algorithms to develop and optimize controllers for games. In general it can be said that there are two different types of AIs in games. Those which do strategic decisions like for chess or Go and those which play action games where quick calculations and timing is important. For strategic AIs it can be at least estimated with an heuristic how good a decision was. With this information, the AI can be trained. For action games like Super Mario, it is a hard task to rate an action because the only real feedback from the game if the action was good or bad would be winning or dying. Genetic algorithms which are oriented on biological evolution have the possibility to train a controller only with its overall performance. Therefore it can be applied to action games like Super Mario which will be done in chapter 3. In the next chapter, evolutionary algorithms will be described and genetic algorithms will be explained in detail.

2 Evolutionary Algorithms

Within the field of Evolutionary Algorithms there are different concepts all based on the idea of biological evolutionary concepts. In this chapter, these different approaches will be shown shortly. This paper focuses on a specific type of Evolutionary Algorithms called Genetic Algorithms which will then be explained as well as the concrete approaches Neuroevolution and Evolving Behavior Trees.

But first a short introduction to evolution itself will be given.

2.1 Evolution

This will just be a very short introduction to evolutionary concepts in general.
Most people think about the cite "Survival of the fittest" when hearing about evolution. From this point they develop the understanding that evolution is somehow the process of optimizing a being. The human for example developed the thumb, because its easier to use tools with it, and so the human got fitter and survived. This is not completely wrong but leads into the misunderstanding that evolution develops new features because they help the being to survive. This idea is not correct. New features develop randomly via mutation and recombination. The new feature will then have an influence on the probability of the being to survive and being able to reproduce. So we do not see people with three arms, because they are not useful

but because the probability of reproduction did not raise for the people with this kind of mutations in the past.

2.2 Different Approaches

Transferring this concept of evolution to computer science, it could be seen as a leaded random search. With the concept of recombination and/or mutation a solution for a problem could be optimized.

There have been 4 different approaches developed so far using this concepts for optimization [Nis98]. This paper will focus on Genetic Algorithms but a short introduction to all of the 4 concepts will be given.

Genetic Algorithms are the most popular category. They will be described way more detailed later in the chapter. The idea behind GAs is to use the main operators selection, recombination and mutation on genetic encoded solutions for a given problem [Nis98]. For the selection a fitness value has to be calculated in order to rate a solution [Nis98]. If a solution can be encoded in a proper way and a fitness value can be calculated, with this concept no more domain-specific knowledge is needed to develop a solution. The operators will create a working solution by repeating over many generations.

Evolution Strategies were developed to optimize real number vectors for engineering [BS02]. The ES is completely phenotype oriented which gives some optimization opportunities in setting parameters for the numbers for the evolutionary operators [BS02]. The selection is done by choosing the best performing individuals deterministic out of the population [BS02]. The crossover can be done by point-crossover which will be described later for GAs or by interpolating between the parents [BS02]. The mutation works with random deltas in a defined distribution [BS02].

Genetic Programming uses the concepts of evolution to evolve computer programs [PBM08]. The programs are generated randomly out of a pool of primitives [PBM08]. Then a genetic algorithm is applied [PBM08]. The fitness of a program is calculated by executing it and evaluating its performance with a fitness function [PBM08]. The selection, crossover and mutation are applied to optimize the population [PBM08].

Evolutionary Programming was first an approach for artificial intelligence [XYG99]. It then was applied to numerical optimization problems [XYG99]. In difference to the other approaches, EP is only using mutation and selection [XYG99].

2.3 Genetic Algorithms

This paper will focus on GAs. In the following the different operators will be described and the basic algorithm shown on a very simple example.

As can be seen in figure 1 the optimization process of an GA is a loop. One loop is called

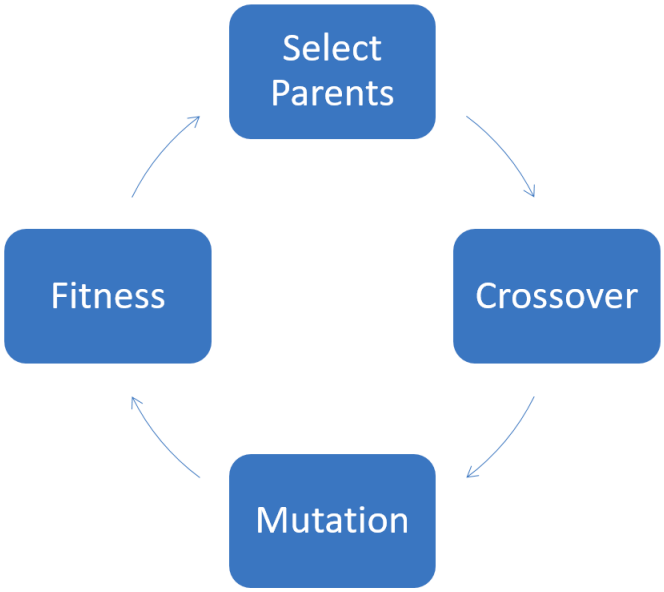

Figure 1: The concept of an genetic algorithm

a **generation** [Nis98]. The algorithm starts with a **population** consisting out of randomly generated **individuals** [Nis98]. What is important here is that an individual has two different types of encoding. First the **genotype** is the genetic encoded information [Nis98]. Second the **phenotype** is the physical expression of the genetic information [Nis98]. In the case of animals the DNA is the genotype of an individual. The phenotype is the animal itself. In nature there is sometimes the case that a specific part of the genome needs a trigger in order to express into a physical appearing.

One generation can be considered as finished, when the fitness of all individuals is calculated. For the first generation, after randomly generating individuals, the fitness value hast to be calculated [Nis98]. The fitness function will be optimized by the whole algorithm

[Nis98]. The fitness value describes the performance of an individual for a given task [Nis98]. It could be for example the traveled distance of an evolutionary constructed car. In this case the fitness function would consist out of the simulation for this car. The result would be the distance. As an example for this paper, binary Numbers could be seen as genetic encoded individuals. The fitness value could be the decimal number. The optimization would than be to maximise this number. The fitness function would calculate the number to every binary encoded number. The goal of course is to ensure that every single bit is set to 1. Lets consider 4 individuals in this scenario (*A=10001100, B=01101100, C=10011011, D=00011000*). The fitness values would be *fit(A)=140*, *fit(B)=108*, *fit(C)=155* and *fit(D)=24*. Because the encoding of binary numbers is well known, it can be seen, that *B* should have a higher fitness than *A*, because it has more bits set to 1. But in a scenario, where is no knowledge about the impact of a gen in a genome, the overall performance is the only way to evaluate an individual.

The next step is to select parents out of the population. If all parents will be transferred to the next population and 2 parents generating 2 children the algorithm has to choose 2 parents to have again 4 individuals in the population. There are different approaches to choose parents from a population. Two of the most popular ones are ranking and tournament selection [Nis98]. In ranking, all individuals will be ranked based on their fitness [Nis98]. The individuals with higher fitness values have a big er chance to be chosen but also the ones with lower values can be chosen by a little chance [Nis98]. This ensures, that the algorithm is not converging too fast. Also as seen in the example with the binary encoded numbers, the fitness value does not always represent the value of the genetic code. Tournament Selection tries to simulate the basic concept of parent selection in nature [Nis98]. A sub group of individuals will be chosen [Nis98]. Then the individual with the highest fitness value will be a parent for the next generation [Nis98]. This somehow simulates the competition in tiny groups in nature [Nis98]. Lets consider a selection algorithm chosed *B* and *C* to be the parents for the next generation.

The next step is the crossover. Here the genetic information of the parents will be combined. This can happen with different approaches. One of the standard methods would be the 1-point-crossover [Nis98]. This methods splits the genome of both parents in two parts and the exchanges them [Nis98]. In this example *B* would be split into *0110* and 1100, C into *1001* and *1011*. The children would then be *C1=01101001* and *C2=10011100*. With the two parents, the new population would be (*B, C, C1, C2*).

The last step is Mutation. This adds some randomness to the algorithm. In nature, copy errors in the DNA as well as radiation can cause changes in the genetic code. In a genetic algorithm, this is done by manipulating the genome after the crossover. This could e done by flipping bits or adding a randomly generated delta to a value in the genome [Nis98]. Most important is the mutation rate [Nis98]. It describes how likely a mutation is [Nis98]. Lets consider, that in the example, in *C2* the second bit was flipped and in *B* the fifth. This would lead to the following population (*01100100, 10011011, 01101001, 11011100*).

With this generation the next loop will start. As can be seen, the maximal fitness value increased to 220 for *C2*. But the idea is, that the whole population increases its fitness value on average. This causes better recombination results in the future.
One method to ensure, that the maximal fitness value will not decrease is by putting the best individual in the next population, skipping crossover and mutation [WO13]. This is called elitism [WO13].

The approach of genetic algorithms can be seen as a basis for different algorithms, which are more precise towards the encoding of the genome or the crossover method. In the following two approaches will be shown which take GAs as a basis.

2.4 Neuroevolution

The idea of Neuroevolution is to take a neural network and optimize its topology and weights trough a GA [SM02]. The simplest variant would be to take a fixed neural network and take every weight as a gen. This is possible but there are more advanced and complex variants. One of them is called NeuroEvolution of Augmented Topologies (NEAT) [SM02]. The idea is to start with a simple perceptron and evolve a structure step by step as well as optimizing the weights [SM02]. This paper will just give an quick introduction. To understand NEAT its important to understand the genetic encoding of the neural networks as well as the crossover. In figure 2 the crossover is shown but also the encoding of the two parents. A neural network is stored as an array containing the information of a connection [SM02]. Additionally there is an enable flag making it possible to switch a connection off or on by mutation [SM02]. Very important for the crossover is also the innovation number seen on top [SM02]. This number is set when the connection appears the first time [SM02]. It makes it possible, to match two connections while doing the crossover [SM02].

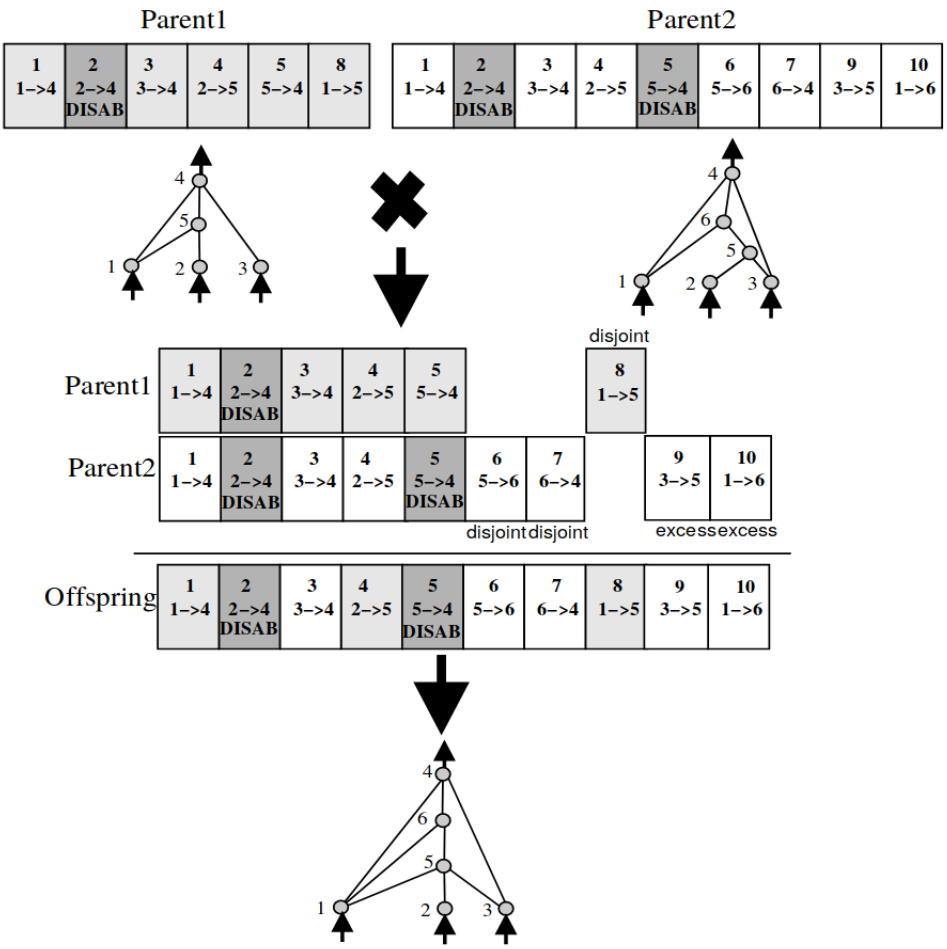

Figure 2: The concept of crossover in NEAT [SM02]

The crossover is done by mapping all connections considering the innovation number [SM02]. The connections of the fitter parent that have no matching partner will be added to the offspring [SM02]. NEAT implements a method called Niching [SM02]. This method divides the population in sub groups considering the difference of their genome [SM02]. This has two effects. First, when doing the crossover most of the connections will have a matching partner [SM02]. This ensure that by doing the crossover semantic meaningful information will be mixed. Connecting two complete different neural networks will most likely result in a bad performing offspring. Second, An individual with a complete new node, will have less competition cause it forms a new species [SM02]. This ensures, that a new feature in evolution will not be erased in the next generation because the weights are most likely not optimized yet [SM02]. In future this new node could cause a better performance.

2.5 Evolving Behavior Trees

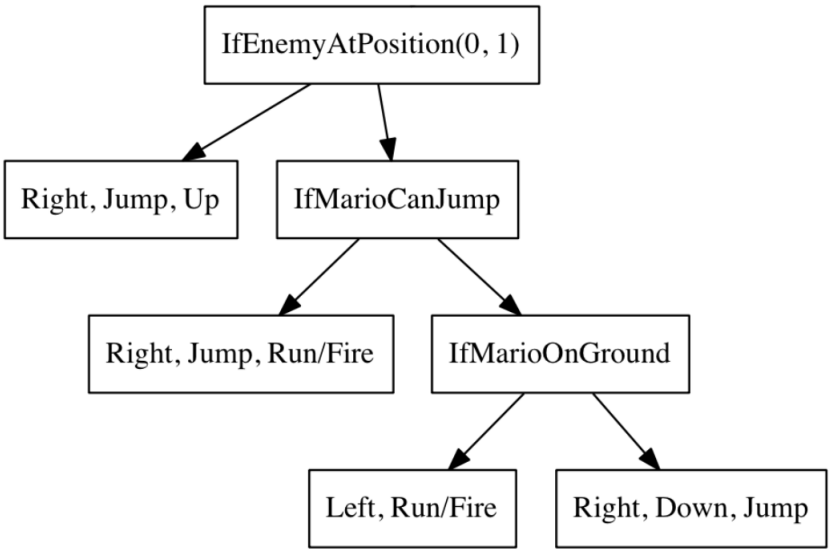

Figure 3: Example for a Behavior Tree [FK15]

Evolving Behavior Trees (EBTs) are the last GA based approach for this paper. An Behavior Tree can be seen in figure 3 for Super Mario. The leafs are actions and the branching nodes are statements which can be true or false [FK15]. For Evolving Behavior Trees a GA is used to optimize this trees [FK15]. This approach works very simple. Because the Behavior Tree is somehow a large logic statement it can be stored via a context free gramatic in an array [FK15]. This is the genotype of an EBT. The crossover is done by swapping sub trees between the parents [FK15]. Mutation is done by randomly changing nodes or leaves or adding them to the tree [FK15]. The application of an EBT to Super Mario will be one of the topics in the next chapter.

3 Super Mario

Super Mario is a jump and run game developed by Nintendo. The player controls a guy called Mario and helps him to survive a 2D Level with enemies [SOR16]. Because this game is an real time game and has limited input options, it fits perfectly to GAs.

3.1 Infinity Super Mario

All following algorithms are based on the Infinity Super Mario Game. This version of Super Mario was originally created by Markus Person and modified by Julian Togelius and Sergey Karakovskiy for the Mario AI competition [FK15]. The terrain is generated randomly and there are power ups and coins to collect like in the original Super Mario [SOR16]. Also enemies are placed randomly in the levels [SOR16]. Mario can be in three diferent states: small, big and big with the ability to shoot fireballs [SOR16]. With fireballs, all enemies can be killed [SOR16]. Mario is able to run in both directions, sprint, jump, crounch and shoot fireballs if he is in the right state [SOR16].

3.2 NEAT & EBT

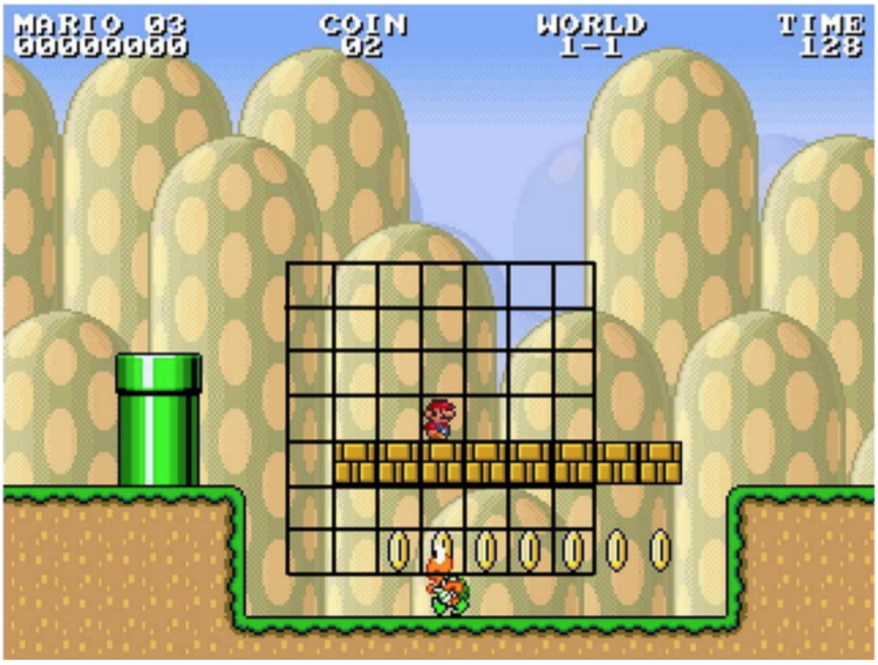

Figure 4: The input grid for EBT & NEAT [FK15]

The first application compares EBTs and NEAT to train a controller for Infinite Super Mario in the Mario AI Championship configuration [FK15]. The population consists out of 100 individuals trained over 1000 generations [FK15]. Each controller was tested on 25 levels which had 4096 steps each [FK15]. The maximal fitness value during training was 25 * 4096 [FK15]. For both EBT and NEAT they tested 3 different input setups using a 3x3, 5x5 and 7x7 input grid [FK15]. Additionally, the algorithm got the *onGround* and *canJump* state as

an input [FK15]. In figure 4 the setup with a 7x7 grid is shown. The grid stores information about the environment at its place. It can be block, enemy or empty [FK15].

The approaches were compared on the different setups. The measurements for the comparison were the maximum fitness reached in the 1000th generation, the rise time, which means the generation, where the mark of 75% of the maximum fitness was reached, as well as the time needed for every generation to evolve. [FK15].
The overall result was that the EBT approach was much faster in training and was able to handle with smaller generation numbers [FK15]. The NEAT approach needed more time but produced a slightly better performing best individual [FK15].

3.3 Interactive Evolution

The second application this paper will cover used an exotic approach. NEAT was used to train a controller for Super Mario but they replaced the fitness function with an interactive rating by the user [SOR16]. The user is watching different behaviors of Super Mario and decides how good they are [SOR16]. The GA uses this algorithm for its selection [SOR16]. As

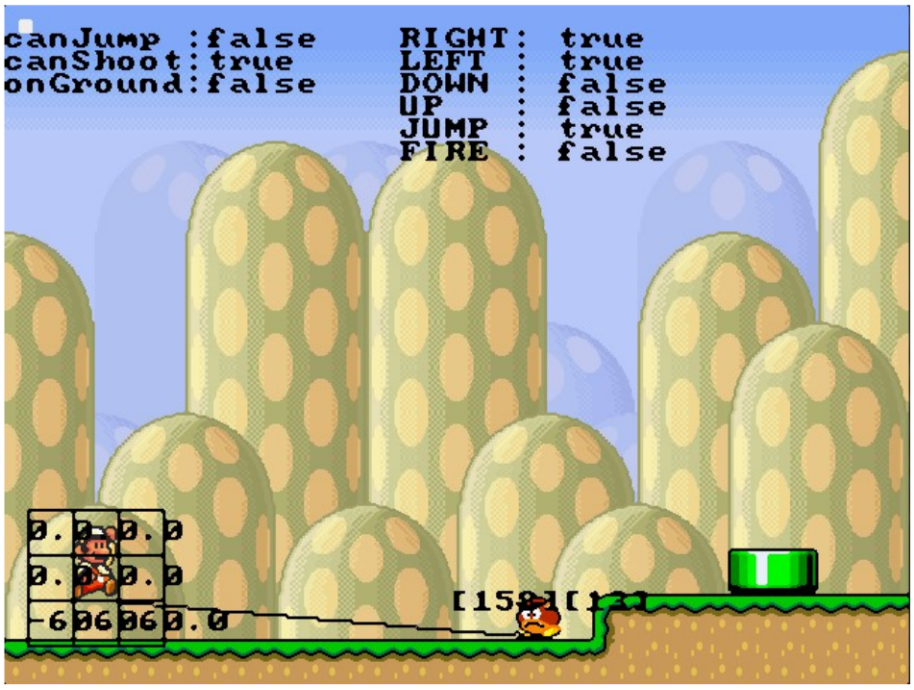

Figure 5: The input for the neural network [SOR16]

can be seen in figure 5 the input to the neural network is a 3x3 grid around Mario. Its values

are between 1.0 and -1.0 were the different values describe different states of the grid entry [SOR16]. Additionally, the angle as well as the distance to the next 2 enemies are provided as input [SOR16]. There are also two state variables *canJum* and *onGround* like in the firsxt approach [SOR16]. They can be true or false [SOR16]. The output of the network are the keys one can press in the original Super Mario: *up, down, left right jump, fire* [SOR16]. The button will be activated with a threshold of 0.5 [SOR16]. For concurring actions like right and left, if both are above the threshold the one with the high activation will be chosen [SOR16].

After each generation has run, the user will be shown 9 Gifs recorded from the playing of the controllers [SOR16]. The user will then be able to compare them and choose its favourite [SOR16]. Based on this decision, the GA will continue.

The interactive method (IEC) was compared to an fitness function based approach [SOR16]. They both optimized the controller significantly, but the IEC approach performed slightly better [SOR16]. Also the users tried to give Mario a specific behavior like killing all enemies or collect all coins [SOR16].

4 Discussion

Evolutionary algorithms are an interesting topic which allows approaches that would not be possible with classical optimization. Their different variants are using the same principles that made life on earth and eventually other planet make possible. Genetic algorithms can always be used if a well behaving solution can be rated as such. There is a saying, that an genetic algorithm is always the second best option to take. Genetic algorithms need a lot of computing power, because they lead a random search by rewarding good paths and punish bad ones. If there is no training data for an machine learning algorithm to calculate a loss and optimize the calculations, a genetic algorithm shows its strength. If it is possible to optimize a solution mathematically genetic algorithms should not be used.

References

[BS02] Hans-Georg Beyer and Hans-Paul Schwefel. "Evolution strategies - A comprehensive introduction." In: *Natural Computing* 1.1 (2002), pp. 3–52. ISSN: 15677818. DOI: 10.1023/A:1015059928466.

[FK15] Ross Foley and Karl Kuhn. *A Comparison of Genetic Algorithms using Super Mario Bros.* Ed. by Sonia Chernova. 2015.

[Nis98] Volker Nissen. "Einige Grundlagen Evolutionärer Algorithmen." In: 1998, pp. 55–78. DOI: 10.1007/978-3-322-86843-5{\textunderscore}3.

[PBM08] Riccardo Poli, William B. Langdon, and Nicholas Mcphee. *A Field Guide to Genetic Programming*. 2008. ISBN: 978-1-4092-0073-4.

[SM02] Kenneth O. Stanley and Risto Miikkulainen. "Evolving Neural Networks through Augmenting Topologies." In: *Evolutionary Computation* 10.2 (2002), pp. 99–127. DOI: 10.1162/106365602320169811.

[SOR16] P. D. SØrensen, J. M. Olsen, and S. Risi. "Breeding a diversity of Super Mario behaviors through interactive evolution." In: *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. 2016, pp. 1–7. DOI: 10.1109/CIG.2016.7860436.

[WO13] Tanapuch Wanwarang and Machigar Ongtang. "Elitism Enhancements for Genetic Algorithm based Network Intrusion Detection System." In: *Journal of Convergence Information Technology* (2013).

[XYG99] Xin Yao, Yong Liu, and Guangming Lin. "Evolutionary programming made faster." In: *IEEE Transactions on Evolutionary Computation* 3.2 (1999), pp. 82–102. ISSN: 1089-778X. DOI: 10.1109/4235.771163.