

# Meistern von unterschiedlichen Computerspielen mittels Generation Based Learning

BACHELORARBEIT

zur Erlangung des akademischen Grades

**Bachelor of Science**

im Rahmen des Studiums

**Medizinische Informatik**

eingereicht von

**Manuel Esberger**

Matrikelnummer 01525631

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dr. Allan Hanbury

Mitwirkung: Pretitle Forename Surname, Posttitle

Pretitle Forename Surname, Posttitle

Pretitle Forename Surname, Posttitle

Wien, 30. September 2018

---

Manuel Esberger

---

Allan Hanbury



# Mastering Diverse Computer Games using Generation Based Learning

## BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

### Bachelor of Science

in

### Medical Informatics

by

**Manuel Esberger**

Registration Number 01525631

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dr. Allan Hanbury

Assistance: Pretitle Forename Surname, Posttitle  
Pretitle Forename Surname, Posttitle  
Pretitle Forename Surname, Posttitle

Vienna, 30<sup>th</sup> September, 2018

---

Manuel Esberger

---

Allan Hanbury



# Erklärung zur Verfassung der Arbeit

Manuel Esberger  
Preßgasse 11/2 1040 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 30. September 2018

---

Manuel Esberger



# Danksagung

Zu erst möchte ich meinen Großeltern danken. Sie haben mir angeboten bei ihnen zu wohnen als ich mein Studium angetreten habe.

Ich werde mich vermutlich mein Leben lang daran erinnern, dass ich bis spät in die Nacht in die Tastatur hämmerte um Tätigkeiten zu erfüllen, die das Studium von mir verlangten, während mein Großvater versuchte im Nebenzimmer Schlaf zu finden. Er hat es sich nie anmerken lassen, dass mein Lernen nicht nur mich wach gehalten hat. Auch erinnere ich mich an die vielen grantigen Diskussionen mit meiner Großmutter wenn es im Studium mal nicht so rund lief. Auch sie hat mir jedes zornige Wort verziehen und sie grüßt mich weiterhin Willkommen in ihrem Heim.

Hätten sie mir nicht ihre Türen offen gehalten und mir einen Platz zum Lernen angeboten, wäre das Studium vermutlich nicht möglich gewesen.

Weiteres möchte ich meiner Freundin und baldigen Mutter meines Sohnes Danken. Auch wenn es abseits vom Studium viel zu tun gab, wie zum Beispiel Möbel kaufen, dem Job oder den nicht enden wollenden Arztbesuchen, erinnerte sie mich immer wieder daran an meiner Bachelorarbeit zu schreiben. Ebenso wenn sie manchmal fragte ob wir etwas Zeit für uns haben wollen und ich sie auf Grund der Arbeit abwieß, zeigte Sie sich mit Verständnis.

Zu guter Letzt möchte ich auch allen Professorinnen, Professoren und Universitätsangestellten danken, die nicht nur an der Wissensvermittlung interessiert waren, sondern die auch aktive Schritte gesetzt haben um interessierten Studierenden bei ihren Lernprozessen zu unterstützen. Ich denke die Universität würde nicht ohne ihnen funktionieren und ich habe sie auch für mein Studium schätzen gelernt.

Vielen Dank!





# Acknowledgements

First of all, I want to thank my grandparents. They offered me to live with them when I started to study.

Probably, I will remember all my life that I pounded into the keyboard in order to solve all the tasks that studying demanded to solve. Meanwhile my grandfather tried to catch some sleep in the adjoining room. Still he never mentioned that my studies not only kept me awake. Also I remember the many times I grumpily argued with my grandma when University was rough. She forgave me every angry word and still welcomes me to her home.

If they wouldn't have opened their doors for me, most likely I wouldn't have been able to study.

Furthermore I want to thank my partner, who also happens to become the mother of our son in a few months. When there was much to do apart from studying, like buying furniture, working a job or the never ending visits at the doctor's place, she still reminded me to take time for my bachelor work. On the other hand, when she asked to spend some time together and I dismissed her proposal because of the bachelor work, she always showed herself understanding.

Last but not least, I want to thank all the professors and employees of the University, who not only tried to transfer their knowledge but also took active steps to support students who were interested. I think that the University would not work without them and I came to appreciate them while my studies.

Thank you!



# Kurzfassung

---

Ihr Text hier.



# Abstract

Enter your text here.



# Contents

<b>Kurzfassung</b>	<b>xi</b>
<b>Abstract</b>	<b>xiii</b>
<b>Contents</b>	<b>xv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation and Problem Statement . . . . .	1
1.2 Results . . . . .	2
1.3 Thesis Structure . . . . .	2
<b>2 Related Work</b>	<b>3</b>
2.1 Neuro-Evolution of Augmenting Topologies (NEAT) . . . . .	3
2.2 Tools . . . . .	3
<b>3 Generation Learning in Computer Games</b>	<b>5</b>
3.1 MarI/O . . . . .	5
3.2 NEAT_FlappyBird . . . . .	11
<b>4 Comparison and Meta-Analysis</b>	<b>17</b>
4.1 Ignored Parameters . . . . .	17
4.2 Comparison of the different game environment . . . . .	17
<b>5 Conclusion</b>	<b>19</b>
5.1 Stuff . . . . .	19
5.2 Future Work . . . . .	19
<b>List of Figures</b>	<b>21</b>
<b>List of Tables</b>	<b>23</b>
<b>List of Algorithms</b>	<b>25</b>
<b>Glossary</b>	<b>27</b>

<b>Acronyms</b>	<b>29</b>
<b>Bibliography</b>	<b>31</b>



# Introduction

*"Some people worry that artificial intelligence will make us feel inferior, but then, anybody in his right mind should have an inferiority complex every time he looks at a flower."*

— Alan Kay, (*Computer Scientist*)

<https://sokogskriv.no/en/writing/structure/structuring-a-thesis/> <http://www.charleslipson.com/How-to-write-a-thesis.htm>

## 1.1 Motivation and Problem Statement

In the last decade many different solutions for neuronal networks (NN) have been implemented, whereas these implementations propose various changes like the amount and distribution of connections between neurons, the weight calculations between neuronal connections or the amount of neuronal layers of the network as well as other structural decisions. The efficiency of these algorithms depend on the problem space and the environment in which they were tested. For example ....

xor problem

concrete examples

One popular method of adapting a neuronal network is via Genetic Algorithm (GA), since GAs offer a way to find new and possibly enhanced patterns in a reasonable (but not necessarily fast) time. In the case of NNs, GAs are used to find new connections between neurons or different structures inside the network. One popular implementation of this combination is NEAT, among others.

find paper about  
GA claims

Since it is not trivial to decide how the NN architecture should look like NEAT builds up its own architecture in a minimalistic way.

write about neat

These NN implementations are used in various fields as mentioned before. One field with rather clear boundaries are games, compared to real world applications. Still there are many types of games with different complexities. Therefore this work analyses two different games which are played by autonomous NEAT implementations for these games. The first NEAT implementation is MarI/O for the game Super Mario World, made by

a popular YouTube-uploader called SethBling <sup>1</sup>. Since Super Mario World is a rather complex game, the results are later compared to a NEAT implementation for Flappy Bird developed for a coding challenge called NEAT\_FlappyBird <sup>2,3</sup>.

Super Mario World and Flappy Bird are two different games in their achievements. A level of Super Mario World has a finite game map but still offers a high level of complexity compared to the input possibilities of Flappy Bird. However, Flappy Bird has an infinite and self generating map and this game can be quite challenging to humans because of the unexpected map and fixed game speed.

Still it is expected that the game solving implementation for Super Mario World takes longer to complete a level than to find a solution for Flappy Birds that can exceed a certain threshold score because of the many possibilities of solving a level in Super Mario World.

## 1.2 Results

1. what was interesting to see
2. contrast to expectations

### 1.2.1 Some References

## 1.3 Thesis Structure

### Chapter 2

### Chapter 3

### Chapter 4

### Chapter 5

---

<sup>1</sup><https://www.youtube.com/channel/UC8aG3LDTdWNR1UQhSn9uVrw>, last accessed on 30th of October 2018

<sup>2</sup><https://github.com/11SourceCell/neuroevolution-for-flappy-birds>, last accessed 30th October 2018

<sup>3</sup>[https://github.com/rsk2327/NEAT\\_FlappyBird](https://github.com/rsk2327/NEAT_FlappyBird), last accessed 30th October 2018

# Related Work

## 2.1 NEAT

NEAT stands for NeuroEvolution of Augmenting Topologies and its concepts have been published. Over the time many implementations in many programming languages were created<sup>1</sup>. Furthermore many extensions and amendments exist<sup>2,3,4</sup>.

<https://stackoverflow.com/questions/45390481/what-is-neat-neuroevolution-of-augmenting-topologies>

## 2.2 Tools

1. MarI/O
2. Flappy Bird
  - NEAT Flappy
  - Machine Flappy
3. Python for statistics

---

<sup>1</sup>[http://eplex.cs.ucf.edu/neat\\_software/#NEAT](http://eplex.cs.ucf.edu/neat_software/#NEAT), last accessed on 31st October 2018

<sup>2</sup><https://www.cs.ucf.edu/~kstanley/neat.html>, last accessed on 31st October 2018

<sup>3</sup><http://eplex.cs.ucf.edu/hyperNEATpage/HyperNEAT.html>

<sup>4</sup><http://eplex.cs.ucf.edu/ESHyperNEAT/>, last accessed on 31st October 2018



# Generation Learning in Computer Games

1. What was measured: fitness development within neats generations
2. two different games: marI/O and flappy
3. different challenges within the game
- 4.

## 3.1 MarI/O

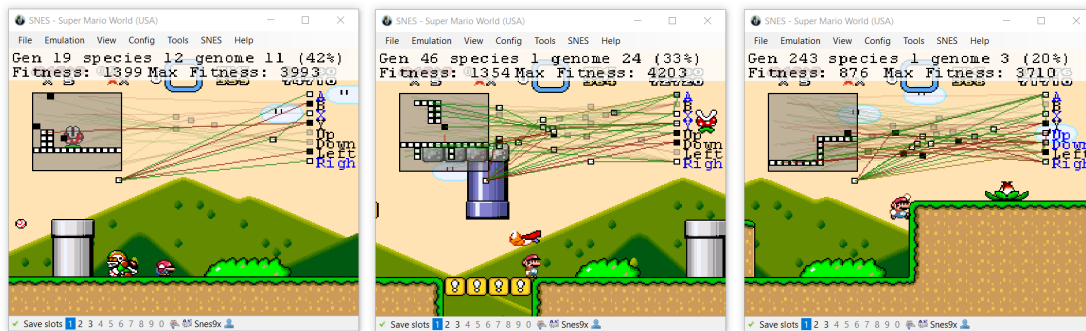


Figure 3.1: MarI/O simulation

As mentioned in section 2.2 MarI/O (see figure 3.1) is an implementation of NEAT-algorithm written in LUA. It provides a solution for automatic learning of the game Super Mario World. In Super Mario World a level is a two dimensional map with steady

as well as moving obstacles. Some of them block the path to the goal and others cause damage to Mario's health. A few of them give health upgrades to Mario or add coins to the players account, although the coins are ignored in the implementation of the Artificial Intelligence (AI). Since there are many different positions in which Mario can stay and the speed of the game depends mostly of the player and his/her/its desicions the environment of Super Mario World is rather complex when compared to the second game Flappy Bird 3.2.

This complex world leaves the expectation that many hundred thousand runs are necessary to learn how to complete a level. However MarI/O implementation reached to goal after approximately 2664.29 runs on average in the simulations described later in this section. Still 2 of the 9 simulations didn't reach the goal once.

fitnessfunction,  
formlar? when was  
goal reached

Later in this section 3 figures with 3 homomorphic graphs will be shown. The three different figures display the success of the algorithm in different classes of initial population size. Since the NEAT-algorithm used does not produce a deterministic amount of populations after the first generation (in general: Generation 0), the initial population size define these classes. There where three classes choosen with a scaling factor of 5 between them. These initial population sizes are 10, 50 and 250. Whether or not the initial population sizes are well chosen will be discussed shortly in the conclusion section (see 4.1) of this chapter. Depending on the evolution of the NN, there are a certain amount of generations evolved. Every generation contains their own set of species. And on the other hand the species contain the genomes. In generation 0 every species contains only one genome each. The sum of all genomes in all species of a generation is called the population. In the cases where the initial population size is 10 or 50 over time to many generations where created to show a viewable graph in the end. That's why only 30 generations where picked in the display with even distances between them. Still a continuous line with the best run of a population is showed above all generations, even the skipped ones.

In the later descriptions of the population classes there are two types of runs introduced. First is the "plot-run" which indicates the simulation and the graph. Inside of this graph there were many "runs" which represent the runs of the population (genomes) of each species. In figure 3.1 there are 3 individual runs displayed. On average one plot-run consists of 4217.2 runs, whereas population 10 has 2828 runs on average, population 50 contains of 4494.6 runs on average and population 250 of 5329 runs averagely.

In order to understand the fundamental differences of these simulations, the population classes are examined in more detail:

**Population 10 / Generation 500** As it is visible in figure 3.2 the vertical axis shows the fitness score average of the genomes within a species. The horizontal axis portrays the generations containing the species. Each generation contains up to 10 populations which is divided into species and genomes within species. This species deviation was made based on the NEAT algorithm described in section 2.1. The best run of the genomes grouped by each generation is marked with a blue line. Therefore the blue line

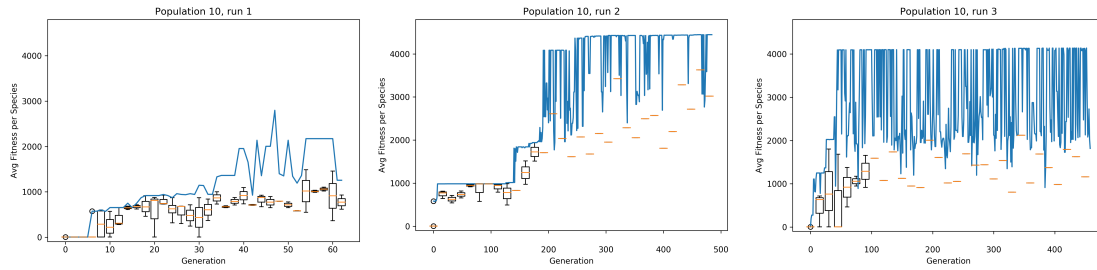


Figure 3.2: MarI/O Population 10

indicates the best overall run within a generation. Since the boxplot portrays the species's average score of each generation and the blue line shows the best run per genome (population), the boxplot and the blue line rarely meet. Still the average population score is closer to the best run than in the next two population variants (see later in this section population 50 3.1 and population 250 3.1). This can be calculated by taking the median of the species fitnesses and subtracting that number from the best run of the genomes:  $average\_distance = \frac{\sum_{g_i \in generations} \max(g_i.genomes) - \text{median}(g_i.species)}{|generations|} \approx 1107$  whereas  $g_i.genomes$  and  $g_i.species$  are lists of the respective fitnesses.

In the three plot-runs on average 334.6 generations were created, which results in a skipping of generations inside the graphics of around 11.15 generations averagely, between two displayed generations. Unfortunately the first run crashed after generation 60. Still, because of the long runtime of the simulation the plot-run was kept. However, indicated by plot-run 2 and 3, the population growth started after this generation. As it can be seen in the 3rd plot-run of the figure 3.2, sometimes runs over 3000 fitness score could be achieved even after the 30th generation. In plot-run 2 the average fitness of the single species left tend to rise, however more and longer plot-runs would be needed to test this hypothesis.

In each generations there are up to 10 populations. In the first generation (Gen 0) no mating was done. So in the first generation there were 10 species spawned with one genome each. In the 10th Generation on average only 4.3 species were left. After generation 50 maximum 3 species were left in all runs and after generation 190 in plot-run 2 and after generation 91 in plot-run 3, respectively, only 1 species was left for mating. The mating results into the crossover of species.

(up to because of neat implementation check neat implementation)

All runs except plot-run 1 reached the goal (the end of the level) multiple times which can be seen by the fitness score being over 4000. However plot run 3 reached the goal the earliest with runs over 4096 starting from generation 44. Still there was the most overall regress made in plot run 3. This can be calculated by adding the differences between the best runs of each generation if the difference was negative:  $average\_regress = \frac{\sum_{g_i \in generations} \min(\max(g_i.genomes) - \max(g_{i-1}.genomes), 0)}{|generations|} \approx -348$  again whereas  $g_i.genomes$  is a list of the fitnesses of each genome inside the generation. The regress of plot-run 1 was  $-88.27$  approximately and of plot-run 2 was around  $-109.98$ .

In plot-run 1 the *average\_fitness\_increase* =  $\frac{\sum_{g_i \in \text{generations}} \max(g_i.\text{genomes}) - \max(g_{i-1}.\text{genomes})}{|\text{generations}|} \approx 19.87$  was the biggest of the three plot-runs since the first plot-run ended early and plot-run 3 had many drawbacks. The average fitness increase of plot-run 2 was around 7.97 and of plot-run 3 was only 3.95 approximately. Since it is only slightly possible to extend the maximum score above the score of 4000 and plot-run 1 has never reached this ranking, plot-run 1 pointed out to have the best score increase per round. Every successful round, whereas Mario reached the goal will only minimize the fitness increase when averaged with the generation count. In another words, for an infinitely large amount of generations the *average\_fitness\_increase* is expected to converge to 0 since the game has and end-state in contrast to the game Flappy Bird, as it can be seen in section 3.2. In mathematical terms:  $\lim_{n \rightarrow \infty} \text{average\_fitness\_increase}(n) = 0$ , whereas the *average\_fitness\_increase*( $n$ ) is defined as the average fitness increase of a set of  $n$  generations.

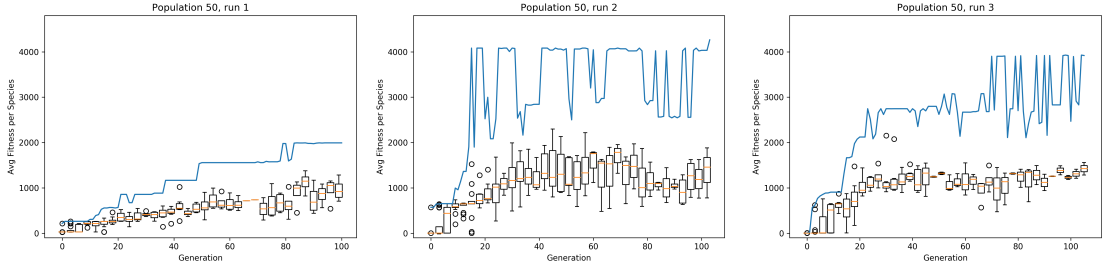


Figure 3.3: MarI/O Population 50

**Population 50 / Generation 100** In this setup the population count is up to 50, again distributed into species and genomes within species according to the MarI/O NEAT implementation. The *average\_distance* between the median of the species of each generation to the best genome run of this generation is bigger than that of the simulation with it's initial population size of 10 but it is smaller than in the last case. The *average\_distance* was calculated as described in the previous simulation (population 10 3.1) and the value is approximately 1406.

In this simulations the plot-runs where executed until there where over 100 generations (101 generations in plot-run 1, 104 in plot-run 2 and 106 in plot-run 3). This results in an average skipping of 3.456 generations between the display of two generations.

In generation 0 there where 50 species spawned, again, with one genome each. In the 10th generation there where 15 species left on average. At the end of generation 100 on average  $3.\bar{3}$  species where left from the initial 50 generations.

Interestingly the plot-run 1 couldn't learn to reach the goal. From this data it is not trivial to predict if the breakthrough would have started within the next 50 generations or if this plot-run would have stayed low in it's fitness score, since there are no clear patterns to find in the graphical representation of these runs. In order to answer on this question more profoundly, further and longer plot-runs have to be made and the big



jumps between the fitness scores of each neighbour generation would have to be analysed. Plot-run 2 and 3 had more luck in reaching the end, however plot-run 3 had more stability in it's high score results between generations. Still after generation 70 plot-run 3 also shows stronger differences between it's generation's heigh scores. Nevertheless, plot-run 2 reached the goal the earliest. The first time plot-run 2 achieved a fitness-score over 4000 was in generation 15 (it reached a score of 4082.5), whereas plot-run 3 reached a maximum score of 3928 in generation 98. Still plot-un 3 reached to goal with a score of 3902 the first time in generation 70. The 3rd plot-run has the highest *average\_fitness\_increase*  $\approx 36.92$  of the three plot-runs. Plot-run 1 has an *average\_fitness\_increase* of 17.58 approximately and plot-run 2 of 35.5 precisely.

Plot-run 2 and 3 have similar *average\_regress* values with approximately  $-158.02$  for plot-run 2 and  $-152.37$  for plot-run 3. Because of the early end of general low performance of plot-run 1 the *average\_regress* is also the lowest with  $-6.30$ . Still there where only 15 cases where the succeeding generation performed worse than the previews in plot-run 1 whereas there where 29 of these cases in plot-run 2 and 30 in plot-run 3. This indicated a certain stability in the first plot-run although the maximum score remained far lower than 3000.

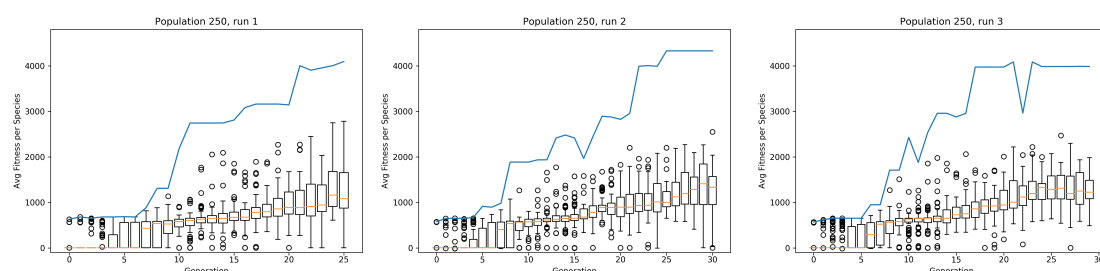


Figure 3.4: MarI/O Population 250

**Population 250 / Generation 30** In figure 3.4 the population size is up to 250 in generation 0. In the first generation (Gen 0) 250 species are born with one genome each. The *average\_distance* for this plot-runs is the biggest with approximately 1827 when compared to the plot-runs with an initial population size of 10 and 50. In this plots no generations had to be skipped in order to portray a descriptive graph since the maximum generation count is 30 in plot-run 2 (25 generations in run 1 and 29 generations in run 3). Already in the 6th generation, on average only 94.8 species where left. At the end of generation 25 there where 31 species left on average.

Compared to the other two population classes there are at least 7 times more species left at the end of the simulations which results in longer whiskers of the boxplot. The whiskers even contains bad starts with fitness-scores lower than 100 in plot-run 1 and 2. Interestingly the best runs are always exceptions after generation 6 (in plot-run 1 and 2 even earlier).

Further it is to mention that the plots are rather uniform compared to the plots of population 10 and 50. Therefore the *average\_fitness\_increase* has similar values with

a low variance which are 133.25 for generation 1, around 121.08 for generation 2 and 113.95 for generation 3. The *average\_regress* is the lowest in plot-run 1 with  $-5.87$  approximately. This is because the maximum value of the succeeding generation is smaller than the previous generation in only 4 cases. The other two plot-runs have an *average\_regress* of approximately  $-19.97$  in plot-run 2 and  $-61.92$  in plot-run 3. All of the plot-runs reached the end of the level even though plot-run 3 reached the end at generation 17, whereas plot-run 1 reached the end at generation 23 and plot-run 2 at generation 22.

MarI/O	avg. runs $/\sigma$	avg. fitness score $/\sigma$	avg distance $/\sigma$	avg. regress $/\sigma$	avg. fitness increase $/\sigma$
Population 10	2828 /2055.44	1231.42 /531.37	1107.09 /534.5	-182.03 /144.01	10.6 /8.28
Population 50	4494.6 /176.09	960.96 /321.34	1405.96 /664.75	-105.56 /86.01	30 /10.78
Population 250	5329 /656.74	776.31 /57.88	1826.32 /81.79	-29.25 /29.16	122.76 /9.76

Table 3.1: MarI/O Population Comparison Overview

Differences and similarities between runs in graph

### Comparison of the results

In order to compare the results, 5 distinct values (see table 3.1) of the plot-runs were calculated, as three of them were introduced in more detail earlier in this section 3.1. The first observations indicate that the average fitness score of each generation drops when establishing a bigger initial population. However the standard deviation tends to drop as well.

Also the distance of the median of the species to the best run of the generation seems to become greater with a greater population count in generation 0. However, the average regress (if present) becomes lower with bigger population sizes and fewer generations, as well as its deviation. Since there are fewer generations in the simulations with an initial population of 250 and these simulations having similar achievements, the average fitness increase is higher than in the other two simulation classes. The standard deviation of the average fitness increase is relatively similar.

It is interesting to see how the fitness increase compares to the average distance value. Even though the fitness increase of population class 250 is much higher than the fitness increase of population class 10, the distance remains large which indicates that the majority of runs stayed low and the average score of population class 10 is higher than in the other two population classes. Still the other two classes remained more stable when taking the average regress into account.

Another interesting point of view is the reaching of the end of the level (the goal). In all population classes the goal was reached even though population class 1 and 2 didn't reach the goal in one plot-run each. Population 10 reached the goal in the first 24.48% on average, only including the cases where the goal was reached. Population 50 reached the goal in the first 40, 24% on average and population class 250 in the first 69, 47%.

To summarize the results roughly it can be said that an initial population size of 10 promises faster and better results in a complex environment like Super Mario World, however more stability can be reached when increasing the initial population size.

## 3.2 NEAT\_FlappyBird

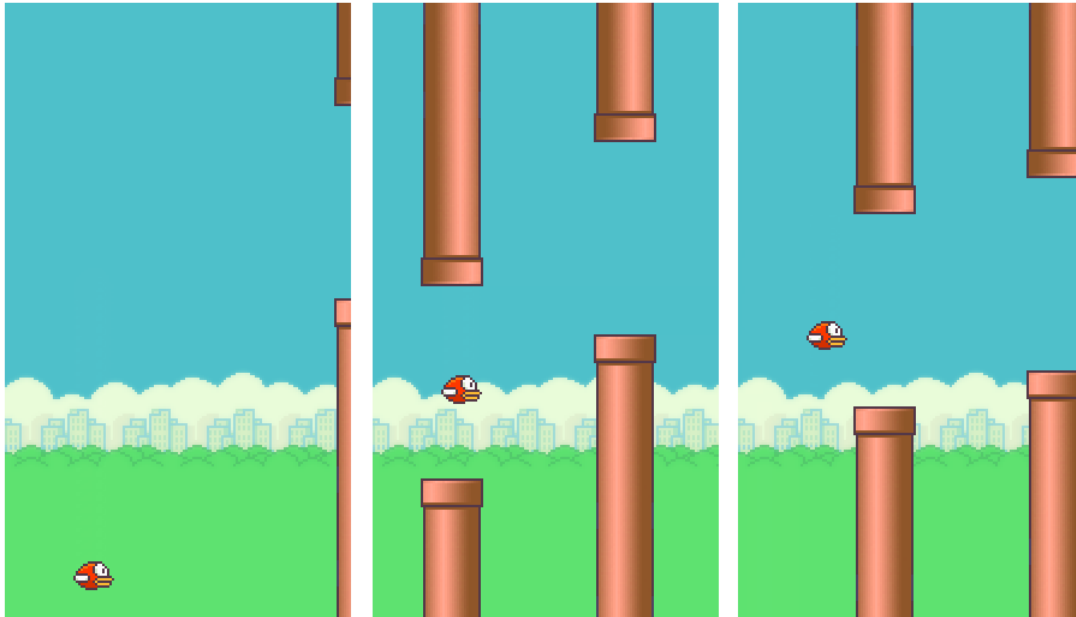


Figure 3.5: Flappy Birds simulation

As mentioned in section 2.2, NEAT\_FlappyBird (see figure 3.5) is an implementation of NEAT-algorithm written in Python using the NEAT framework NEAT-Python<sup>1</sup>. It provides a solution for mastering the game Flappy Bird which was also rewritten in Python. In Flappy Bird a level is a two dimensional, infinite map with obstacles along the way. The player, in the representation of a bird, moves in a constant speed to the right, where new obstacles are generated. These obstacles are pipes which create gaps, through which the bird has to fly (see figure 3.5).

Therefore the player has only one input, namely if he/she/it wants to fly up, or if not then the player automatically falls down. Since there is only one decision to make the game is rather simple compared to Super Mario World (see section 3.1).

This simple set-up creates the expectation that the NEAT algorithm should be able to find a solution with a score above a certain threshold (which is defined later in this section). In fact all simulations exceeded this threshold and the generations used were far lower the maximum generation defined for this simulations. NEAT\_FlappyBird reached the fitness threshold after  $1047.4$  runs on average.

fitnessfunction,  
formlar?

In this section, 3 figures similar to the ones used in section 3.1 are shown. However, this time these figures are split into two ranges since the details of the graph are too far apart. Therefore a simulation of a population class contains 3 plot-runs whereas one plot-run is displayed in 2 graphs of different ranges. As the size of the population classes the same 3

<sup>1</sup><https://neat-python.readthedocs.io>, last accessed 30th October 2018

### 3. GENERATION LEARNING IN COMPUTER GAMES

sizes where chosen as used in the MarI/O simulation, which are 10, 50 and 250.

Again a maximum amount of 30 generations are displayed in the graphs when the generation size was bigger than 30. Luckily, this was only the case in the population 10 simulations. Also, again, the generations contain the species, which by them self contain the genomes (the population). As in MarI/O a "plot-run" indicates the simulation and also the graph which was plotted based on the simulation. Inside of this graph there were many "runs" executed which represent the runs of the population (genomes) of each species.

Now the simulations are examined in more detailed categorized by their population size:

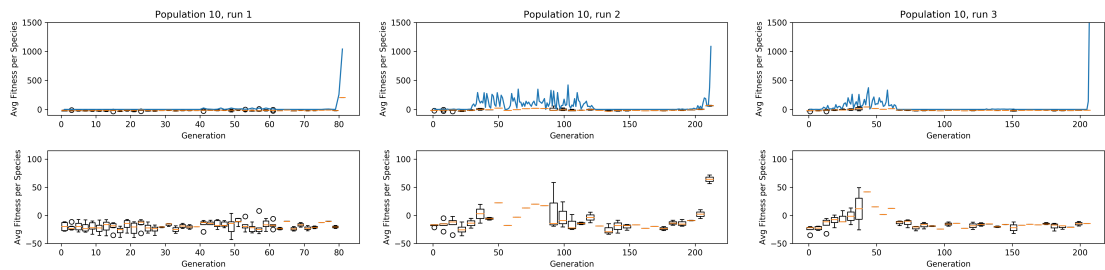


Figure 3.6: Flappy Bird Population 10

**Population 10 / Generation 500** As it is with the graphical representations of MarI/O (see figure 3.2 for example) the vertical axis shows the fitness score average of the genomes within a species. Again, the horizontal axis portrays the generations containing the species. Each generation contains up to 10 populations which is divided into species and genomes within species. In this NEAT-implementation a fixed size of generations was specified.

As in MarI/O the best runs of genomes per generation is marked with a blue line. In the Flappy Bird simulations the best runs and the median of each species (or the *average\_distance* described in section 3.1) are to far apart to be able to display them in one graph of linear scale. That's why two graphs are shown with different fitness ranges. The upper part displays a range from -100 to 1500 and the upper plot shows values in the range of -50 to 115 fitness score.

Although the *average\_distance* is big in general, this plot-run has the smallest *average\_distance* of the three simulation classes with an average value of 51.3. In this three plot-runs the upper boundary was 500 generations however there was an fitness-threshold implemented as well which ended the simulation when a fitness score over 600 was reached. Plot-run 1 launched 81 generations, plot-run 2 launched 307 generations and plot-run 3 had 367 generations before this threshold was exceeded. This results in an average skipping of  $8.3\overline{8}$  generations inside the graphs, between two displayed generations.

Interestingly, plot-run 2 and 3 managed to enhance their score by generation 30 but dropped again latest at generation 130, however all three plot-runs managed to reach a score beyond 600. Plot-run 1 reached this goal the earliest in generation 81 and therefore needed less than a third of the generations plot-run 2 and 3 spawned.

In comparison to MarI/O (see section 3.1) where in generation 0 there were as many species spawned as configured with the population size, whereas every species contained only 1 genome, this implementation of the NEAT algorithm spawns the configured amount of genomes first and after the first simulation run assigns them into species. Moreover, the generation number starts from 1 and not from 0 as in MarI/O. So after the first run on average  $5.\bar{3}$  species were classified. In generation 80 on average  $1.\bar{6}$  species were left. Since there was a setting configured which reseted the species if a total extinction (no species left) has occurred. This values have to be considered carefully.

In plot-run 2 and 3 there was a significant *average\_regress* made of approximately  $-24.54$  in plot-run 2 and  $-11.38$  in plot-run 3. Plot-run 2 regressed 67 times and plot-run 3 41 times. Plot-run 1 kept the *average\_regress* far lower with an approximate value of  $-1.42$ .

Since plot-run 1 has the fewest generations the *average\_fitness\_increase* is the highest with a value of around 16.04, whereas plot-run 2 has a value of  $\approx 6.86$  and plot-run 3 of  $\approx 10.54$ . However, the average score of each generation is the lowest in plot-run 1 ( $\approx -15.87$ ). Plot-run 2 has a value of approximately  $-4.22$  and plot-run 3 of  $-10.62$ .

Since the game environment is open ended the expectations from MarI/O (see 3.1) that  $\lim_{n \rightarrow \infty} \text{average\_fitness\_increase}(n) = 0$  does not hold here. However the opposite is true that  $\lim_{n \rightarrow \infty} \text{average\_fitness\_increase}(n) = \infty$  is to be expected. Therefore there was the fitness-threshold introduced to end the simulation before an infinite flight is expected.

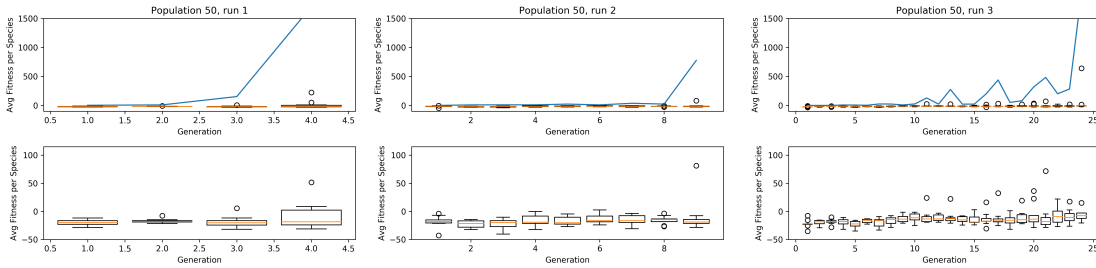


Figure 3.7: Flappy Bird Population 50

**Population 50 / Generation 100** This simulations have an average *average\_distance* of 270.25. In this instance the standard deviation should be taken into account, which is also quite high with a value of 192.24. In the plot-runs of this population class the upper boundary was 100 generations however, again the fitness-threshold of 600 was exceeded in every plot-run. Plot-run 1 launched 4 generations, plot-run 2 launched 9 generations and plot-run 3 had 24 generations when the threshold was exceeded. Therefore no generation had to be skipped in the graphical display.

After the first run, in generation 1, the genomes were divided into  $9.\bar{3}$  species on average. At the end of generation 4, plot-run 2 and 3 kept their species and plot-run 1 lost one species. The reset of species did not have to be used at this point. In plot-run 3, 7 species

### 3. GENERATION LEARNING IN COMPUTER GAMES

where left at the end (generation 24). In plot-run 1 no regress occurred and in plot-run 2 only one time a regress was made which results in an average-regress of  $-0.07$ . Plot-run 3 had many potential outbreaks which resulted to 9 regresses with an *average\_regress* of  $\approx -45.84$ .

The *average\_fitness\_increase* is the highest in plot-run 1 since the generation count is the smallest. It's value is 461.52. The *average\_fitness\_increase* values are similar in plot-run 2 and 3 with approximately 87.93 in plot-run 2 and 92.79 in plot-run 3. The average score of each generation remains negative in this simulations for all plot-runs.

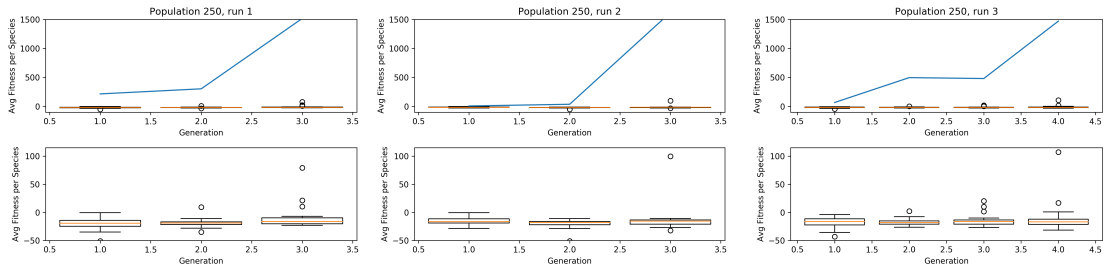


Figure 3.8: Flappy Bird Population 250

**Population 250 / Generation 30** In three plot-runs have an average *average\_distance* of 636.5. Compared to the population 50 instances the standard deviation is lower with a value of  $\approx 63.14$ . In the plot-runs of this population class the upper boundary was 30 generations however, again the fitness-threshold of 600 was exceeded in every plot-run. Plot-run 1 and 2 launched 3 generations, and plot-run 3 had 4 generations when the threshold was exceeded. Again, no generation had to be skipped in the graphical display. After the first run, in generation 1, the genomes where divided into  $22.3$  species on average. At the end of generation 3, all plot-runs contained more than 10 species each. Interestingly all plot-runs could avoid regress within the next generations.

The *average\_fitness\_increase* was high in general with a value of 433.86 in plot-run 1, 537.57 in plot-run 2 and 347.07 in plot-run 3 since there were few generations spawned. Since the NEAT algorithm had no problems with this game it is more interesting to see how this population classes compare to one another.

NEAT_FlappyBird	avg. runs / $\sigma$	avg. fitness score / $\sigma$	avg distance / $\sigma$	avg. regress / $\sigma$	avg. fitness increase / $\sigma$
Population 10	1684 / 714.01	-10.23 / 5.84	51.3 / 18.93	-12.45 / 11.59	11.15 / 4.62
Population 50	626.6 / 531.71	-9.77 / 2.95	270.24 / 192.24	-15.30 / 26.45	214.08 / 214.30
Population 250	831.6 / 139.72	-13.81 / 1.38	638.5 / 63.14	0 / 0	439.5 / 95.38

Table 3.2: Flappy Bird Population Comparison Overview

#### Comparison of the results

In order to compare the results, the same 5 distinct values (see table 3.2) of the plot-runs where calculated, that were taken into consideration in section 3.1.

Differences and similarities between runs in graph

The most obvious observation is that the average fitness increase rises with the amount of populations used, since the generations remain low in count when the threshold is reached. Secondly, the average distance (from the species median to the best genome run) tend to rise as well with a bigger population size.

The other measurement values leave little to no conclusions since the values don't rise or fall with growing/shrinking population sizes. The average regress, for example tend to rise between population class 10 and 50 but is 0 in population class 250. The question is, if the population class 250 would have a greater regress than the other two population instances if there would be any (future) regress. the average fitness score is non rising/shrinking as well, however the standard deviation seems to shrink with greater population size. Probably, the negative values of the fitness score depend on the environment of Flappy Birds, where only partially well learned birds can make it through the first obstacle (namely the pipes).

Since there is no defined goal, the fitness-threshold can be taken into consideration when deciding how good simulations have been. Since the threshold was exceeded in every plot-run, the last generation holds the best run of a plot-run. The fewest average runs were done by population class 50 with averagely  $626.\bar{6}$  runs. However population class 250 had the smallest standard deviation of their running length, which indicates that the goal can be more consistently reached around the  $831.\bar{6}$ th run compared to population count 10 and 50.

### 3.2.1 Plain Machine learning flappy bird

1. better results
2. multi simulation made it easier
3. easy algorithm for easy environment might be explanation for better results
4. a) Differences between runs (lucky runs with 4th champion generation)

### Comparison to NEAT results





# Comparison and Meta-Analysis

## 4.1 Ignored Parameters

1. abstract parameters
  - population size (maybe not choosen well => futer work)
  - Population sizes smart? (10, 50, 250)
  - many wheels that can be turend
  - further work, checking influence of nn parameters
2. differen setup => similar goal
3. differences / similarities in neat implementation (fixed size in machine learning flappy bird whereas dynamic species with marI/O)

## 4.2 Comparison of the different game environment

1. differences / similarities in outcome table with all the data collected (non concrete values because they depend on games (arrows))

Data Trend Comparison	avg. runs $/\sigma$	avg. fitness score $/\sigma$	avg distance $/\sigma$	avg. regress $/\sigma$	avg. fitness increase $/\sigma$
MarI/O	$\uparrow / \times$	$\downarrow / \downarrow$	$\uparrow / \times$	$\downarrow / \downarrow$	$\uparrow / \times$
NEAT_FlappyBird	$\times / \downarrow$	$\times / \downarrow$	$\uparrow / \times$	$\times / \times$	$\uparrow / \times$

Table 4.1: Date Trend Comparison of different games and their NEAT implementation

diff:

- a) The first observations indicate that the average fitness score of each generation drops when establishing a bigger initial population. However the standard deviation tend to drop as well.
- b) However, the average regress (if present) becomes lower with bigger population sizes and fewer generations, as well as it's deviation.
- c) The standard deviation of the average fitness increase is relatively similar.

sim:

- a) Also the distance of the median of the species to the best run of the generation seem to become greater with a greater population count in generation 0.
- b) population 250, the average fitness increase is higher than in the other two simulation classes.
- c) It is interesting to see how the fitness increase compares to the average distance value. Even though the fitness increase of population class 250 is much higher than the fitness increase of population class 10, the distance remains large which indicates that the majority of runs stayed low and the average score of population class 10 is higher than in the other two population classes. Still the other two classes remained more stable when taking the average regress into account.

#### 2. future studies

- genome/generation plot & differences to other plot
- compare calculations with all data not only with maxfitness in case of *average\_fitness\_increase* and *average\_regress*
- check in text for (future or further)

#### 3. test MarI/O previous evolutions on other levels (short 5)

# Conclusion

*"By far, the greatest danger of Artificial Intelligence is that people conclude too early that they understand it."*

— Eliezer S. Yudkowsky, (*Artificial Intelligence Researcher*)

<https://sokogskriv.no/en/writing/structure/structuring-a-thesis/> <http://www.charleslipson.com/How-to-write-a-thesis.htm>

- 
- test MarI/O previous evolutions on other levels (long: what would be expected, how long to adapt to differences, would be better than start from anew)

## 5.1 Stuff

## 5.2 Future Work



# List of Figures

3.1	MarI/O simulation . . . . .	5
3.2	MarI/O Population 10 . . . . .	7
3.3	MarI/O Population 50 . . . . .	8
3.4	MarI/O Population 250 . . . . .	9
3.5	Flappy Birds simulation . . . . .	11
3.6	Flappy Bird Population 10 . . . . .	12
3.7	Flappy Bird Population 50 . . . . .	13
3.8	Flappy Bird Population 250 . . . . .	14



# List of Tables

3.1	MarI/O Population Comparison Overview . . . . .	10
3.2	Flappy Bird Population Comparison Overview . . . . .	14
4.1	Date Trend Comparison of different games and their NEAT implementation	17





# List of Algorithms



# Glossary

**LUA** "Lua is a powerful, efficient, lightweight, embeddable scripting language. It supports procedural programming, object-oriented programming, functional programming, data-driven programming, and data description."<sup>1</sup>. 5

---

<sup>1</sup><https://www.lua.org/about.html>



# Acronyms

**NEAT** Neuro-Evolution of Augmenting Topologies. 5

**NN** neuronal networks. 1



# Bibliography

- [App10a] Apple Inc. *Keynote '09 User Guide*. Apple Inc., 2010.
- [App10b] Apple Inc. *Numbers '09 User Guide*. Apple Inc., 2010.
- [App10c] Apple Inc. *Pages '09 User Guide*. Apple Inc., 2010.
- [Fre10] Free Software Foundation, Inc. Gnu general public license, 2010.
- [Jür95] Manuela Jürgens. *LaTeX: Fortgeschrittene Anwendungen*. FernUniversität Gesamthochschule in Hagen, 1995.
- [Jür00] Manuela Jürgens. *LaTeX: eine Einführung und ein bisschen mehr*. FernUniversität Gesamthochschule in Hagen, 2000.
- [KJUM11] Markus Kohm and Jens-Uwe-Morawski. *KOMA-Script: Die Anleitung*. 2011.
- [Mie11a] André Miede. *A Classic Thesis Style: An Homage to The Elements of Typographic Style*, 2011.
- [Mie11b] André Miede. A classic thesis style by andré miede, 2011.