**Degree Project**

Dalarna University, master's in Microdata Analysis with emphasis on Business Intelligence

**All for one one for All**

Description: Streck

Author: Bamshad Shirmohammadi

Supervisor: Professor Arend Hintze

Examiner: Professor Siril Yella

Subject/main field of study: Microdata Analysis

Course code: MI4002

Credits: 15 ECTS

Date of examination:

|  |  |
| --- | --- |
| At Dalarna University it is possible to publish the student thesis in full text in DiVA. The publishing is open access, which means the work will be freely accessible to read and download on the internet. This will significantly increase the dissemination and visibility of the student thesis.  Open access is becoming the standard route for spreading scientific and academic information on the internet. Dalarna University recommends that both researchers as well as students publish their work open access.  I give my/we give our consent for full text publishing (freely accessible on the internet, open access): | |
| Yes | No |

Dalarna University – SE-791 88 Falun – Phone +4623-77 80 00

**Abstract**

Imagine a school where everyone tries to help the weakest student, and there is no competition among them except for being more nice and supportive! Imagine this school is one of the top ten schools in the world! Yes, it is a too big goal, but every big move strat from a small step. The small step in my thesis is working on the same objective in the world of artificial intelligence. Obviously, this universe is way simpler than our world, but it covers some of the matters, such as altruism, and we can make it more realistic in the next step. Moreover, regardless of the direct application of this study on our cooperation, and considering the growth in artificial intelligence usage in the industry, we must avoid the same corruptions we have, such as selfish behaviors in the robots' collaboration. For doing such an experiment, we made a game for multiple artificial agents, and we have directed their teamwork by setting a variety of incentives. To make it more practical, we used the Darwin evolution theorem's group selection mechanism so only those artificial agents who are more supportive in the group will reproduce, and eventually, the lastest generation will include only supportive members. We used a framework that includes an evolvable neural network for implementing the agents and a genetic algorithm working based on specific rewarding schemes to address the evolution to implement the experiments. After the analysis of our results, we have concluded that it is possible to direct them, so they perform better individually and in teams at the same time. This success happens when we reward our agents based on the weakest's performance. So, we could successfully build the same atmosphere as the dreamed school for our agents' cooperation!

**Keywords:**

Artificial intelligence, Neural Networks, and Evolution

**Acknowledgment**

First and foremost, I have to thank my research supervisor, Professor Arend Hintze. Without his assistance and dedicated involvement in every step throughout the process, I would have never accomplished this study. I would like to thank you very much for your support and understanding.

**Contents**

[Introduction 1](#_Toc60971042)

[Background 1](#_Toc60971043)

[Purpose 4](#_Toc60971044)

[Literature review 4](#_Toc60971045)

[Material and Methods 5](#_Toc60971046)

[Experiments 5](#_Toc60971047)

[Game definition 5](#_Toc60971048)

[Game implementation 8](#_Toc60971049)

[Game execution 10](#_Toc60971050)

[Data Description and Format of Output Files 11](#_Toc60971051)

[LOD.csv 11](#_Toc60971052)

[Movement.csv 11](#_Toc60971053)

[Beep.csv 12](#_Toc60971054)

[Methods 12](#_Toc60971055)

[LOD analyzer 12](#_Toc60971056)

[Movement analyzer 14](#_Toc60971057)

[Beep analyzer 15](#_Toc60971058)

[Statistical Methods 16](#_Toc60971059)

[Results 16](#_Toc60971060)

[How agents adapt over the course of evolution 17](#_Toc60971061)

[Comparison of the different scenarios 19](#_Toc60971062)

[Minimum and Maximum Selection Schemes and Clonal Groups 21](#_Toc60971063)

[Other observations about clonal groups 22](#_Toc60971064)

[Agents' behaviors 24](#_Toc60971065)

[Do agents communicate for a reason? 26](#_Toc60971066)

[Discussion 27](#_Toc60971067)

[Conclusions 29](#_Toc60971068)

[References 30](#_Toc60971069)

[Appendix: the source codes 31](#_Toc60971070)

[lodAnalyzer.py 31](#_Toc60971071)

[movementAnalyzer.py 37](#_Toc60971072)

[beepAnalyzer.py 41](#_Toc60971073)

# Introduction

The state of the art machine learning technique to optimize neural network AI controllers (agents) is back-propagation and specifically deep learning. This method assumes that solutions and examples of such solutions are known. A different approach to this is evolutionary computation and genetic algorithms. Instead of rewarding proper responses to specific situations and hoping for proper generalization, a genetic algorithm uses an objective function. This objective function evaluates the total performance of a system and can select controllers that perform better. Over many generations, such a system optimizes controllers, finds more creative solutions [1], and most importantly, does not require knowledge about specific actions but only the desired outcome. This method works well when optimizing individual controllers as their objectives do not conflict with the objectives of other controllers. When agents need to interact or work in teams [2], the situation becomes much more complicated, and often the goals of the individual are contrary to the success of the group. Imagine self-driving cars. Optimizing them to reach their goal as fast as possible might induce negative side effects. How to overcome such issues? Group-level selection plays an important role in evolution. Often individuals do not act alone but in groups. As such, they can achieve together more than alone. Collaborative hunting is one of those examples where individuals are not selected individually but as a group [3]. This group-level selection scheme typically pools the resources the group collected and redistributes them back equally, which implies that groups are evaluated by their average performance. What if this scheme is altered? Imagine a group is rewarded according to the performance of its best individual (all for one) or its worst performer (one for all)? In the all for one case, you would pool all resources on one individual, and in the case of the one for all, you would distribute the resources as fairly as possible. The big question is, which of the two groups collected the most resources in total? This research will use neuro-evolution, where agents are simulated in a virtual environment and controlled by so-called Markov Brains. Group-level selection regimes such as the ones described above will be tested with respect to their effect on individual and group level performance.

## Background

Societies are depending on the collaboration of their members. This collaboration is the basis of the economy, health care system, and education, among others. Obviously, such institutions benefit from everyone contributing to them via taxes. However, since they are carried by society, individuals who for example, evade taxes can still benefit from them while also benefiting from the resources they did not contribute due to cheating.

To study this problem, social science, experimental economics, and game theory use the "Public Goods Game." [citation] In this game, the participants are given equal amounts of money. They then can decide to withhold this money (defect) or contribute (cooperate) into a public pot. The total contribution of all participants will be multiplied by a synergy factor. This synergy represents gains that can only be achieved by pooling resources. The now larger amount of money in the pot will then be divided equally among all the players - regardless of them having contributed or not in the first place. Considering the game rules, a player who is careful about the social benefits; will contribute money, and those who are selfish and only consider their personal benefits will not. The scenario that individuals only think about their gain exploiting the contribution of others is called "Tragedy of the Commons". [citation]

The question is now, how can people be incentivized to be collaborative? In other words, how we can avoid the tragedy of the commons?

If we see it from a government perspective, we would consider incentives to motivate people or punishment to avoid selfish approaches. But for a broader historical perspective, we have to ask where cooperation within groups comes from initially? We find that Darwinian Evolution Theory also struggles to explain why organisms cooperate [4], regardless of organisms, including humans, evolved to cooperate. To discuss how natural selection is the solution for the collaboration problem, let's first examine how we humans in agriculture select the best genes for reproduction. In this example, a farmer, based on a certain criterion (such as the biggest plants) selects the desired ones for reproduction.



First Generation

Second Generation

Third Generation

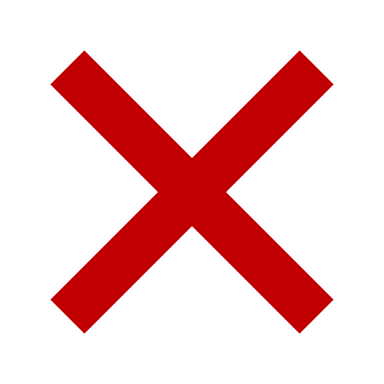
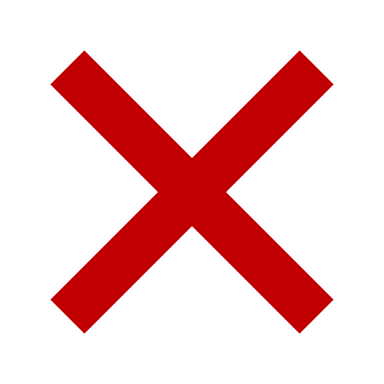
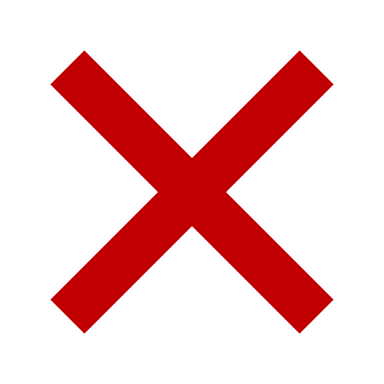
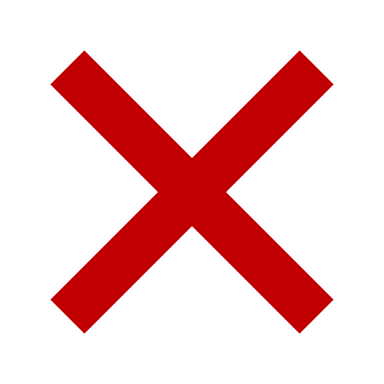
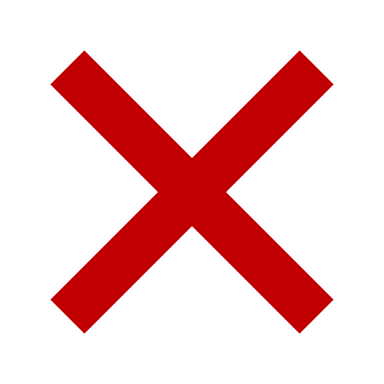
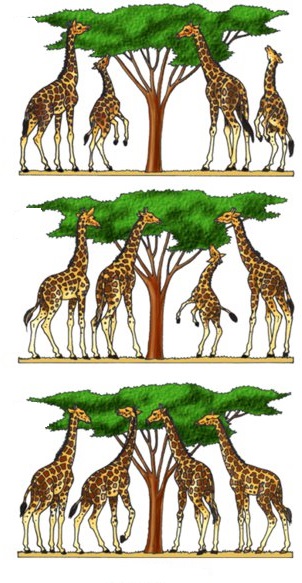


Figure 1. Human selection

In the above picture, the farmer harvests only big cabbages in the last generation by choosing only the biggest for reproducing. Nature does the same to all organisms, and only lets some of them survive through history, and the rest will become extinct.



First Generation

Second Generation



Third Generation

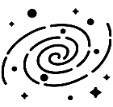
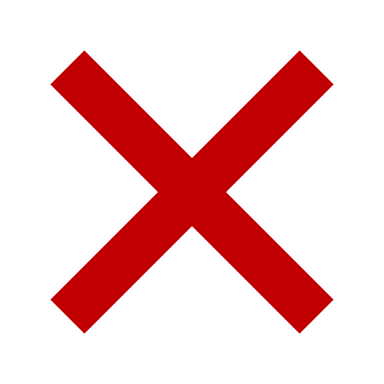
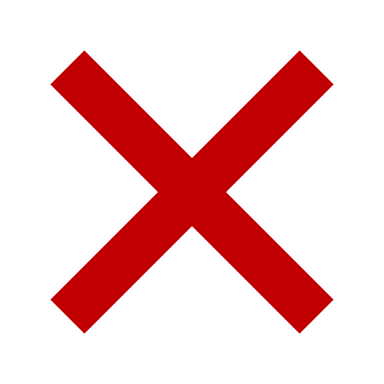
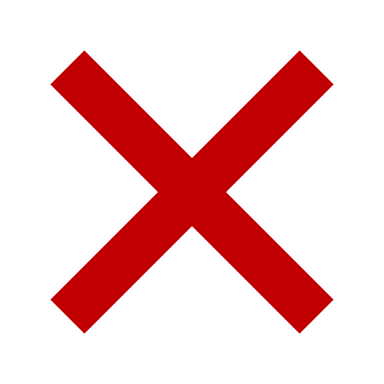


Figure 2. Natural selection

So, nature acts as the farmer and lets only the tall giraffes survive and produce offspring as they had an advantage in eating leaves on high trees. We call this *natural selection*. In the context of natural selection, the individuals' ability to survive and reproduce is called *fitness (w).* You may ask how it comes that we have a variety of different species if only the fittest survives? Then all organisms should look like each other. The answer lies within the two other concepts of mutation and selection. Mutation means sudden and significant changes in the species genetics that results in offspring being different from their parents. Suppose we take a family of giraffes and divide them into two distinct islands which one of them has tall trees, and the other has short trees. If there were no mutation in nature, regardless of the islands' differences, the giraffes after generations would not be different while both groups are from the same family. The mutation causes significantly taller and shorter offspring in the two groups. In the short tree island, the smaller ones will survive and reproduce, and the opposite happens on the other island. Eventually, after a long time by natural selection and mutation, we will have two types of giraffes (short and tall) from the same routes. Consequently, the natural environment presents organisms with different ways to be the fittest, called niches. Those niches, together with inheritance, variation, and natural selection, leads to the biodiversity we observe today. Darwin's *evolution* is, therefore, a mechanism composed of inheritance, variation (random mutation), and selection. It thus leads to an adaptation of organisms that fit their environment better.

However, evolution is inherently selfish, as its rewards happen on a short time scale. Reaping the rewards now, and making more offspring immediately, will always outcompete saving rewards and reproducing later. Cooperation needs the investment of multiple partners, but from the tragedy of the commons, we know that the defectors fare better. Still, we find different biological mechanisms that allow cooperation to evolve. Among others (kin selection, green beard effect, reciprocity [4]) the one that most likely gave rise to multicellularity is group-level selection [citation]. Here, not the individual reproduces, but the entire group benefits from the rewards the entire organism receives. Thus, the most prevalent choice for incentivizing cooperation is group-level selection.

Since evolution already solved the problem of cooperation before, in this study, I would like to optimize groups of agents (robots) using a genetic algorithm to cooperate with each other. These agents are controlled by evolvable Neural Networks, specifically Markov Brains [5]. Previous work has already performed preliminary research in this direction. The MABE (Modular Agent Based Evolution) Framework [cite MABE] to run computational experiments has been developed before, and group-level selection reward functions have been tested before. I will build in these systems and test different forms of group-level selection, as well as new payoff schemes seeking to improve cooperation. This will be compared to none group-level selection.

## Purpose

This research seeks to improve the way we train groups of AI controllers (agents) to perform better individually and in teams at the same time. While this is a basic research question in the optimization of neural networks using genetic algorithms, it has direct applications to robotics and other autonomous AI decision making systems that need to work in groups.

As explained in the introduction, the thesis is in the sequence of other works using Markov's brain. Based on the literature review on other works in this chain of researches and also the other similar studies, I decided to work on the impact of incentives or rewarding schemes on team working of the AI controllers. According to the literature review, group-level selection has been shown to improve cooperation within groups of agents. Here I introduce a new set of fitness criteria where the effort of the group is not measured by the average performance of the group, but instead by either the worst performer or the best performer. This work on the *minimum* and *maximum* reward schemes fills a particular *research gap* as it has not been tested before.

## Literature review

I must mention that I have selected the thesis proposal, and therefore, before starting the literature review, I was aware of the thesis objective that is based on the identified research gap by the supervisor. Finding similar works in this area and learning their methods is the main aim of my literature review.

The first research I want to introduce is [2]; they have simulated hierarchy among social animals using animats controlled by evolvable Markov Gate networks (MGN). The artificial animals are inside an area covered by walls and have a door to leave. These animals' performance has been evaluated and based on their observations; the best bots leave the room later than others. In other words, they have identified staying in the area as a dominant strategy. Working on the effect of different reward schemes on the formation of hierarchies is one of their future works, which is very near to my thesis's main objective, which I have explained in the previous section. Integrating disparate information sources by biological organisms is another aspect in this area which is investigated in [6]. In [6] the information integration is used in terms of navigation, and it has been observed that the artificial agents' fitness increase by its (information integration) enhancement. After the hierarchy among social animals and integrating disparate information sources by biological organisms, swarming behaviors in animals is the third concept from this world which is simulated in [7]. The authors of [7] have used Markov Network (MN) for their agents' controller. They “demonstrated that predator confusion provides a sufficient selective advantage for prey to evolve swarming behavior in a digital evolutionary model. [7]” Moreover, they have proposed a new way to test hypotheses in terms of evolution of swarming behavior in their work.

# Material and Methods

This research used the MABE (C++ Modular Agent Based Evolution Framework [8]) to implement virtual test environments. Agents were controlled using Markov Brains [5], which are a particular evolvable type of neural network. After replicating evolutionary experiments were performed, data was analyzed and visualized.

To train groups of AI controllers (agents) to perform better individually and in teams at the same time, we have defined and implemented a game for a group of agents. They do the game under different conditions (rewarding schemes and type of groups), and we have evaluated their performance according to their own payoff and their team payoff. Based on the comparison of the agents' performance, we have confirmed which reward scheme allows them to evolve good teamwork behavior or to perform well individually. I will explain the game's definition, how we have trained the agents, and how we have evaluated them in the rest of the document.

Figure 3. Overview of the research methodology

## Experiments

### Game definition

In this simple game, four agents are tasked with collecting food in a confined space. Agents are placed in the four corners of an area surrounded by a wall that prevents agents from leaving this space. Agents can then sense their environments, specifically whatever is on the tiles in front of them. This sensor can distinguish emtpy tiles, walls, agents, and food items. They also have a sensor informing them about the total number of food collected, as well as three auditorial sensors which tell them if any of the other agents beeped in the last round. Each of these sensors is mapped to a specific other agent. The auditory sensors allow for communication between the agents. Once sensor information is obtained, agents can perform computations defined by their Markov Brain. These computations lead to several possible actions that each agent can take: move forward, turn left or right, and if food has been previously collected either drop that food on an empty tile, or if an agent is in front of them, they hand the food to that agent. This computational process Markov Brains perform can be imagined like a recurrent neural network that received inputs in the first layer, and then performs a forward pass, so that outputs are computed. Agents can in addition to their movements also utter a beeping signal as a means to communicate something.

Food is distributed over the entire area of 16x16 tiles, which includes the walls. Then 50 random tiles are cleared of food, to generate a less monotonous landscape. Agents are allowed to roam for 500 updates. After that, the amount of food agents collected is used in the different reward schemes to assign fitness such that the genetic algorithm can perform its selection routine on the population.

When a new generation is formed from the last, all new agents can experience mutations, deletions, and gene duplications, which can lead to changes of the Markov Brains. Over many generations, this process of evaluating agents, selecting better ones based on their fitness, and mutating them randomly evolves their strategy to perform more optimally.

A more complex environment is easy to imagine, however, as this work is a proof of concepts for the Minimal and Maximal rewarding scheme, as well as clonal group selection, we chose this simple environment. The MABE environment as well as the necessary changes to implement this world has been done before the project started by the members of the Hintze-lab (Clifford Bohm, Jory Schossau, and Arend Hintze).

A picture containing text, scoreboard

Description automatically generated

Figure 4. Illustration of a simple example of the game

### Game implementation

The MABE computational modeling framework allows evolutionary experiments to be setup and conduced with ease. MABE consists of several modular components which have been extensively tested before, which brings high confidence in their functionality. The one component that needed to be adapted is the World-Class which defines the environment agents have to perform in, and which measures their fitness. All other components, such as Markov Brains, their Genome, the selection mechanism, the lineage and data tracking, and the parameter system remained untouched. For each conditions 100 independent replicate experiments were run, using the UPPMAX high performance compute center – "UPPMAX Grant for Neuro-Evolution Arend Hintze".

**I. Area or world:**

It is the place where the experiments will be done. This area is divided into tiles, and each tile can be set as empty, grass, and wall. The agents can only move in empty or grass tiles. The agents can collect and either consume it directly, increasing its count, or later hand it to another agent or put it in an empty tile for others. In the end, the amount of grass (energy) each agent collected will be used to determine the score of the group. If they go into a grass tile, it will be converted to an empty tile.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Figure 5. An area example. In this example, the gray tiles are walls, and the green tiles are grass.

From a technical point of view, the area is an object of the Area class created in ExampleWorld.hpp (world/ExampleWorld folder). The Area class is defined in area.cpp (core folder). We have set the area properties in the ExampleWorld.cpp (world/ExampleWorld folder).

The Area class has a constructor, which has three arguments that are x, y (as the dimensions), and startTile, which is for selecting the initial filling type (such as grass) of them. To specify the tiles type individually, we can use set() function in the Area class. X value and Y value are the first two arguments to select the tile position, and the third argument sets the tile's type.

The following table explains how to set the different attributes of the area:

|  |  |
| --- | --- |
| Function name | Description |
| Area::Area(int \_xDim,int \_yDim,int startTile) | It is the constructor of the area class that set the area size and initial tiles type. \_xDim and \_yDim can be a positive number, and the startTile is defined as follow:  0 = empty  1 = grass  2 = agent  3 = wall |
| Area::set(int x,int y, int what) | Using this function, we can set each tile's type individually. x and y identify the tile position and type (grass, empty, wall, and agent) |

Table 1. The main functions to set the area attributes

**II. Agents:**

Agents are the people of the world (area), and their task is to collect grasses. They can turn left, right, move forward/backward, talk to each other, or do nothing in each turn. As soon as they move to a new tile filled with grass, they will collect it. They can pass the collected grass to other agents, put it on an empty tile, or use it for their personal score. Our agents can reproduce and have offspring, and they are also evolvable (using Darwin's theory of evolution).

Agents (organisms) are objects of the organism class, which is decelerated and defined in Organism.hpp and Organism.cpp, respectively. These four agents of the game can be either clone or different. These two types of agents are called group modes that are shown in the next table.

|  |  |
| --- | --- |
| Group mode | Meaning |
| 0 | Clone |
| 1 | Four different organisms |

Table 2. group modes

**III. Training methods and evolution**

We used different reward schemes to evaluate agents, which can lead to different evolutionary adaptations.

|  |  |
| --- | --- |
| Reward schemes | Meaning |
| 0 | Individual reward |
| 1 | Mean score (What they got on average) |
| 2 | Maximum score (What their best performer received) |
| 3 | Minimum score (What their worst performer received) |

Table 3. Reward modes

These reward schemes consider the amount of food collected, and given the conditions rewarded, translate the collected food into different fitness values for the genetic algorithm. For example, when we set the rewarding scheme to zero (individual), whatever a single agent received individually will determine its fitness. Thus, those who collect more food will get a better score and have a higher chance of producing more offspring. If the rewarding scheme is set to 2, for example, the agent who collected the most food is identified, and the amount of food of this agent now determines the fitness of each agent within the group. Thus all agents within the group receive the same rewards and thus have the same competitive advantage during evolution. In order to have a higher fitness than agents from other groups, they need to work together and ensure one agent to collect more food than any other agent from any other group.

### Game execution

So far, I have explained how we have created a world and set our agents' incentives. Each time we run the experiment, we only change two parameters (rewarding scheme and group mode), and the rest will remain constant. The rewarding scheme has four possible options, and group mode has two, and therefore, totally, we have eight types of combinations. Each experiment is repeated one hundred times to ensure our results do not happen accidentally.

For MABE to properly evolve agents, not only the task and how it is rewarded needs to be specified, but also several other parameters dealing with evolution itself are required. The parameters I used here, are default parameters which have been shown to function properly before. As they are not changed, I will not discuss them further. Only the number of generations to be 5000 has been tested such that evolution has sufficient time to optimze agents. Also, the 500 turns each agent in the game can take has been tested, and shown to be sufficient to evaluate the performance of the agent or the group. The next two tables explain the constant parameters and variables for the executions:

|  |  |
| --- | --- |
| Parameter | Possible values |
| Group mode | Clone and Not clone |
| Rewarding scheme | Individual, Maximum, Minimum and average |

Table 4. Variable parameters of the experiments (the game)

|  |  |
| --- | --- |
| Parameter | Value |
| Number of generations | 5000 |
| Number of turns | 500 |
| Population size | 100 |
| Type of brain | Markov Brain, using deterministic logic gates and genetic programing mathematical operators |
| The initial score of agents | 0 |
| Starting genome size | 5000 |
| Maximum genome size | 20000 |
| Starting number of gates seeded | 5 |
| Point mutation rate | 0.0001 |
| Deletion and Gene-duplication rate | 0.2 |
| Number of hidden nodes | 5 |
| Entire area | 16\*16 tiles including walls |

Table 5. Constant parameters of the experiments (the game)

## Data Description and Format of Output Files

When we run our compiled file, it generates three groups of CSV files: LOD, movement, and beep. As I have explained before, we have eight possible combinations (two options for group type and four for reward schemes). Each combination is repeated one hundred times. Therefore, each of the groups (groups of CSV files) includes 800 CSV files. So, our data folder has three subfolders for LOD, movement, and beep, and inside of each of these subfolders, there are 800 CSV files. I will explain the structure of these files in this section.

### LOD.csv

First, I should explain the line of descent (LOD). LOD is the first and most important part of our data. It includes the score of agents belonging to the line of descent we have used to select the best combination (set of group mode and reward scheme) for our agents (in the results section, I will explain the best combination). We have 5000 generations, and we set the resolution of our LOD files to 100 so, each of them has 50 rows. So, each row contains information of one generation that are the score of all the four members and the score and the ID of the one which belongs to LOD. I have explained the columns in the next table. (The difference between score and ownScore). We used the LOD files to generating the evolution graphs (see results section: How agents adapt over the course of evolution), finding the best combination (see results section: Comparison of the different scenarios), and the analysis on clonal groups (see the last two subsections of the results).

|  |  |
| --- | --- |
| Column name | Explanation |
| Generation | Generation number |
| ID | ID of each agent |
| Score | This field, based on rewarding scheme value, has the following meanings:   |  |  | | --- | --- | | Reward mode | Score Meaning | | 0 | Individual score | | 1 | Mean score of the four agents | | 2 | Max score of the four agents | | 3 | Minimum score of the four agents | |
| rawScores | Scores of all the group members |
| ownScore | The own score of the agent which is selected in the line of decent |

Table 6. Explanation of the LOD files structure

### Movement.csv

We used the movement files to analyze our users' actions, such as using a beep or giving their grass (energy) to other agents (see Agents' behaviors in the results section). These files contain only information about the last generation. They explain the agents' actions in each turn, and therefore they have 500 rows (because our game has 500 turns). These columns (actions) are explained in the following table. I must mention I worked on all of them but, only A and B representing usage of beep and passing the grass (either to an agent or an empty tile) and do nothing are used in our results and analysis.

|  |  |  |
| --- | --- | --- |
| Column name | Explanation | |
| T | Turn number | |
| X | Position in X axis | |
| Y | Position in Y axis | |
| D | Direction | Meaning |
| 0 | Up |
| 1 | Right |
| 2 | Bottom |
| 3 | Left |
| E | The number of grasses that the agent gathered. It can be named energy as well. | |
| A | Action type | Meaning |
| 0 | Do nothing |
| 1 | Turn left |
| 2 | Turn right |
| 3 | Move forward |
| 4,5,6,7 | giving grasses to an agent or putting it on a tile. |
| B | Beep (yes or no) | Meaning |
| 0 | No |
| Any other number | Beep |

Table 7. Explanation of the movement files structure

### Beep.csv

The last group of files is beep. These files are used to identify if our agents use beep purposefully. So, we have tested their usage of beep in two conditions: mute (when they can not beep) and talking (when they can beep). Each test is replicated 100 times, which is equal to the number of rows for these files.

|  |  |
| --- | --- |
| Column name | Explanation |
| Replicate | The replicate number |
| m0 to m3 | These four columns show the score of the four agents when we have muted them. |
| b0 to b3 | These four columns show the score of the four agents when we have allowed them to communicate (using the beep) |

Table 8. Explanation of the beep files structure

## Methods

### LOD analyzer

I have created lodAnalyzer.py to convert LOD files to our desired graphs and a table. Its input is all the 800 LOD files. It generates eight graphs (one for each combination) which explain how agents adapt throughout evolution (see the first subsection in the results) and a table called lod statistics, which I have used for comparison of the different scenarios (see results' second subsection). Also, I have used the information on this table (lod statistics) to compare clone and non-clone groups (see the third and fourth subsections in the results).

Figure 6. The LOD analyzer code flow

To ensure this module's results (graphs) are accurate and correct, I have compared a small percentage (0.25% of LOD\_0\_0\_0 and LOD\_0\_0\_1) of the python code results with the manual outputs from excel. The next figure shows that the python code graphs match the graphs created by excel tools.

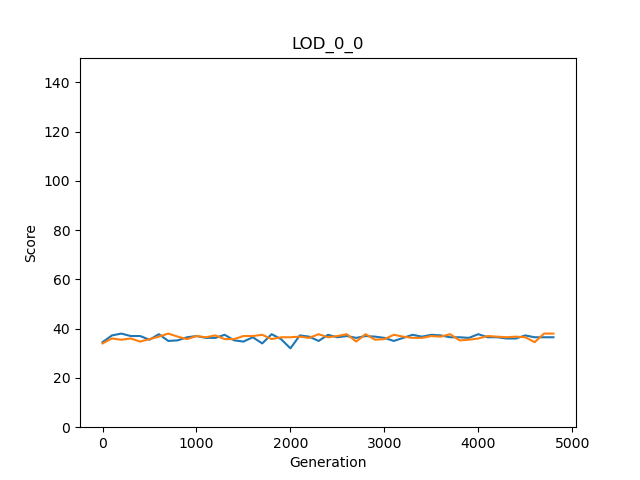


Figure 7. comparison of the python cod's exported graph and graph created using excel

As I mentioned before, this python code (lodAnalyzer.py) also generates a table containing all the combinations' statistics. For this test, I have checked if I calculated the minimum, maximum, sum, and average of each row correctly. For this case, I calculated two of the rows from LOD\_0\_0\_0 and LOD\_0\_0\_1 manually and compared them with the python results.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| rewardGroupModes | meanOwnScores | meanMinimums | meanMaximums | meanAverages | meanSums |
| LOD\_0\_0 Replicate 0 & 1  (Excel) | 36.41326531 | 25.83673469 | 48.3775510 | 36.413265 | 145.653061 |
| LOD\_0\_0  Replicate 0 & 1  (Python) | 36.41327 | 25.83673 | 48.37755 | 36.41327 | 145.6531 |

Table 9. comparison of the python cod's results and manual calculations using excel

As you can see in the above table, all the results from python and manual calculations using excel for this small sample are matched.

### Movement analyzer

This module (movementAnalyzer.py) gets the movement files (800 files) as the input and exports two excel files. The first file (movStatisticsLog) contains all the files' statics, and therefore, it has 800 rows. The statistics are about the three main agents' actions (do nothing, passing the grass, and beeping). The next excel file (movStatistics), which is the main one, summarizes the previous file for the eight combinations. I have used this information to analyze the agents' behaviors (fifth subsection in the results).

Figure 8. The movement (actions) analyzer code flow

First, to test this module, we should make sure that each movement file's percentages are calculated correctly. For this proposal, I selected two of the files and compared their results using python and excel.

Secondly, the next testing part is about the averages, which again we compared python and the Excel results (for two of the files).

|  |  |  |  |
| --- | --- | --- | --- |
| Reward scheme and  group mode | Mean of *giving or putting grasses* percentage | Mean of *do nothing*  percentage | Mean of *beep* percentage |
| Movement\_1\_1 Excel | 0.002 | 2.0775 | 21.978 |
| Movement\_1\_1  Python | 0.002 | 2.0775 | 21.978 |
| Movement\_2\_1  Excel | 2.006 | 0.8965 | 21.8615 |
| Movement\_2\_1  Python | 2.006 | 0.8965 | 21.8615 |

Table 10. comparison of the python cod's results and manual calculations using excel

As the previous table shows, the manual calculation using excel and the python outcomes are identical.

### Beep analyzer

I have made this code to ensure that our agents use beeping purposefully (not randomly). Its input is the 800 beep files, and it makes an excel file called beepStatistics. I used this output file information is in the results section for the usage of the beep (table number 15 and 16). Like the other modules, I tested it by comparing its results for a small percentage of data with excel results.

Figure 9. The beep analyzer code flow

### Statistical Methods

Two different statistical methods were used. The standard error was computed to estimate if measurements are different from each other, used for Table 14.

The two-sided Kolmogorov-Smirnov Test is used to determine if the unknown distributions of values are different from each other, and to estimate a p-value of the statement that both distributions are different by chance. This test was used to assess if values in Table 15 and 16 are significantly different from each other. We do not know if the samples compared in those cases are normal distributed or not, thus this more complex test, and not a simple t-test or other method was necessary.

# Results

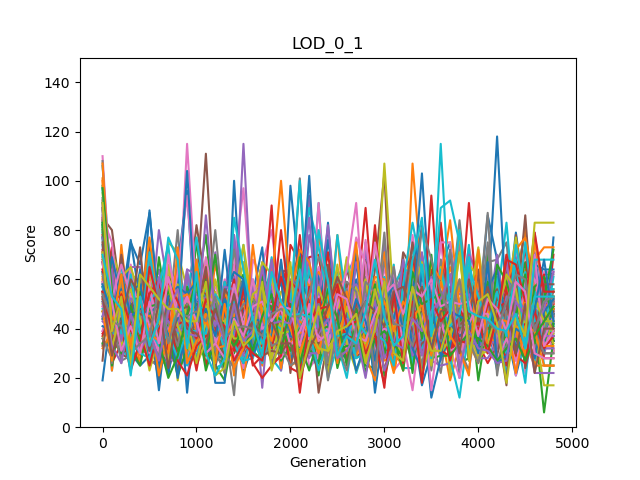
The question to be addressed here is about payoff distribution within groups. Specifically, groups of cloned agents were compared to groups composed of randomly selected agents from the population. Secondly, the payoff scheme was varied: Distributing the mean payoff the group obtained to everyone (mean), identifying the payoff the best performer in the group had and rewarding all agents of the group with that performance (max), and lastly, awarding each group member according to the least performer in the group (min). While group-level selection, as present in clonal groups, is believed to improve cooperation, here additionally, payoff according to the worst (min) or best (max) performer is tested, with the hope that it might reveal an even better incentive for cooperation.

For this purpose, all conditions were tested using an evolutionary computational model, testing each condition in 100 replicate evolutionary runs. The results of these experiments are organized into five sections: a) How agents adapt over the course of evolution, b) Comparison of the different scenarios, c) Minimum and Maximum Selection Schemes and Clonal Groups, d) Agents' behaviors e) do agents communicate for a reason?

## How agents adapt over the course of evolution

Agents were created randomly at the beginning and had 5000 generations to evolve, given the different experimental conditions. Over the course of evolution, their performance is expected to improve according to the selection criteria.

Individual | Clone

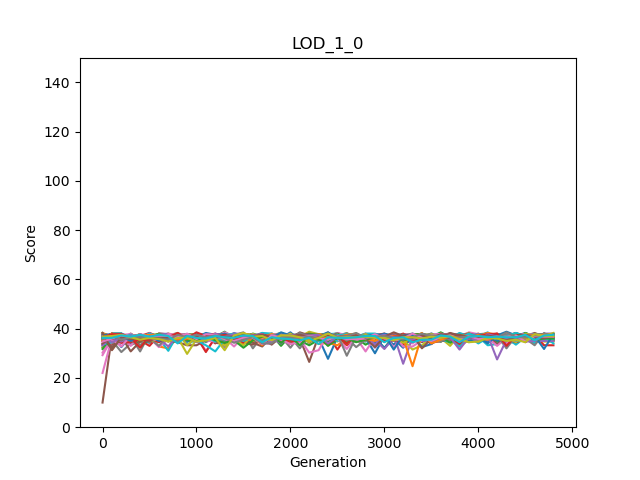
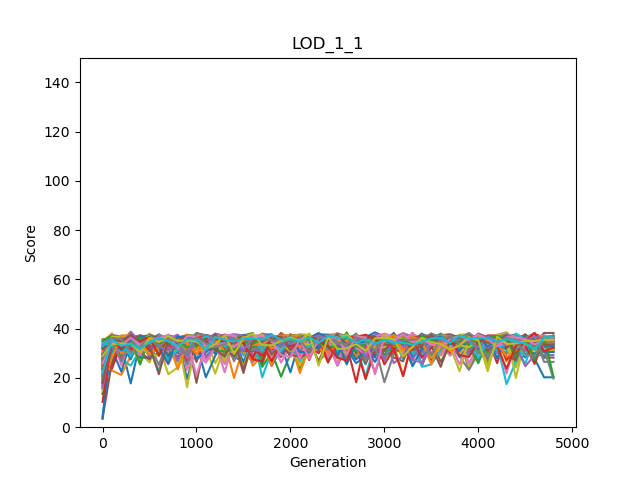


Individual | Not Clone

Figure 10. Score variations of the individual rewarding scheme

When rewarding individuals according to the amount of grass collected by themselves, and agents are clones, I find little improvement of performance, and the performance remains roughly constant over the course of evolution (see Figure 10 left). When the group is not composed of clones, little performance improvements are observed, but performance varies highly during evolution (see Figure 10 right). Comparing these two different scenarios suggests that the not-clonal group evolves better performance than rewarding clones. Additionally, higher variance during evolution seems to have a beneficial effect on performance, but this needs to be explored further, comparing more conditions.

Mean | Clone

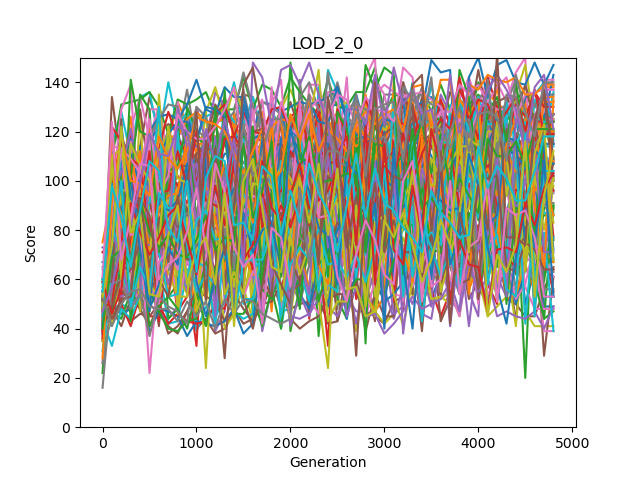
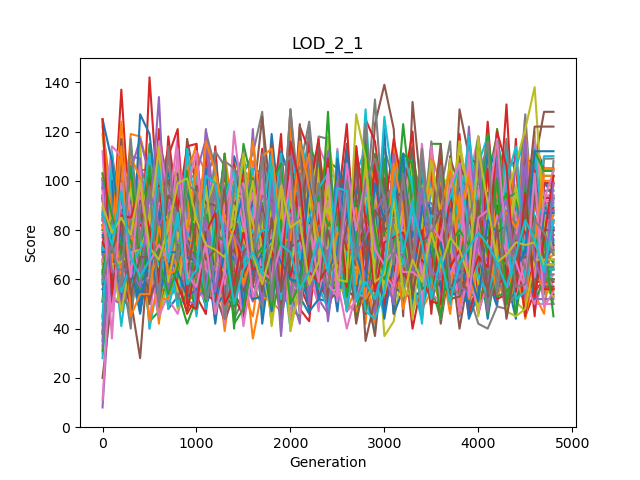


Mean | Not Clone

Figure 11. Score variations of the mean rewarding scheme

When rewarding the group members not according to their own performance but according to the average performance of the group, the results differ. While clonal groups again show little improvement over 5000 generations, the not clonal groups show some improvement, but both conditions have little variation during adaptation (see Figure 11 left and right).

Maximum | Clone

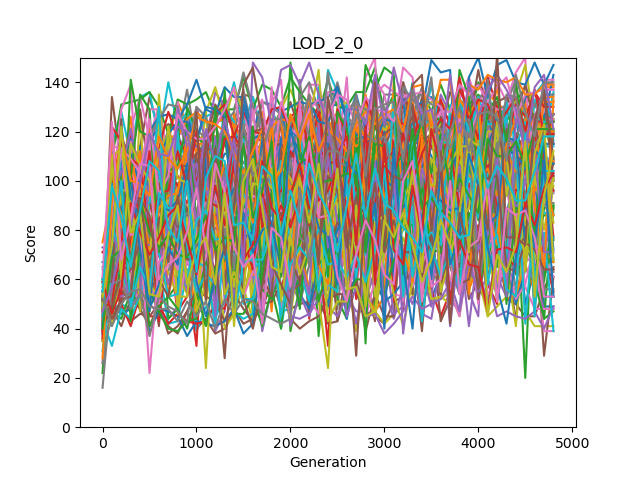
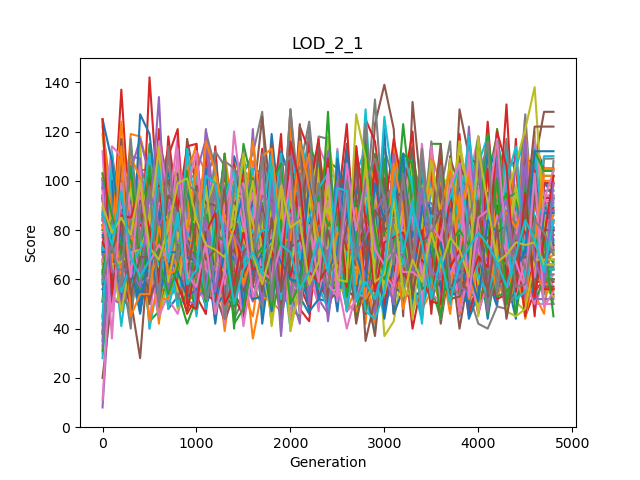


Maximum | Not Clone

Figure 12. Score variations of the maximum rewarding scheme

The max reward scheme identifies the individual of the group who collected the most grass and now rewards all agents in the group according to that maximum performance. Thus, the group should evolve a strategy that pools resources on one individual or at least avoid that individuals compete about resources. Both clonal and not clonal groups show that their performance improves over generations while also showing a high degree of variance throughout (see Figure 12 left and right). The clonal group seems to have a higher variance as well as a higher performance in the end.

Minimum | Clone



Minimum | Not Clone

Figure 13. Score variations of the minimum rewarding scheme

The last condition identified the individual who collected the least grass and then rewards all group members according to that performance. This reward scheme should prevent agents to not participate in collecting grass at all, as that would result in a payoff of zero. At the same time, not one individual should collect overly much grass; instead, agents should devise a strategy to keep the amount of grass collected by everyone equal. Like in the maximum rewards scheme, we find an improvement in the performance of 5000 generations, as well as a high variance within the adaptive periods (see Figure 13 left and right).

In the next section, the performance of agents after evolution will be compared. Further, the observation that some conditions lead to high variances in performance, while others do not, seem interesting as well and should be investigated further. Is it possible that those evolutionary runs that show a high variance explore more options and thus have a higher chance to find better solutions?

## Comparison of the different scenarios

As we have discussed in the experiment section, we kept all the parameters constant except the rewarding scheme and how groups are composed (clone or not clonal), resulting in eight different experimental conditions. We expect different experimental conditions to affect individual behavior and how the groups as a whole perform. Thus the criteria compared are the average payoff of the individual across all 100 replicate experiments with the same condition. Further, the average performance of the group member who collected the least (mean minimum) and the most (mean Maximum), the average amount the groups collected (mean Averages), and finally the average of what groups collected in total (mean sums).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| rewardGroupModes | Mean  OwnScores | Mean  Minimums | Mean  Maximums | Mean  Averages | Mean  Sums |
| LOD\_0\_0  Group mode: Clone  Reward mode: Individual | 36.4275510 | A picture containing shape  Description automatically generated26.0175510 | 47.4516326 | 36.4275510 | 145.710204 |
| LOD\_0\_1 3  Group mode: Not Clone  Reward mode: Individual | 44.7497959 | 20.1351020 | 52.0906122 | Logo  Description automatically generated36.5244898 | Logo  Description automatically generated146.097959 |
| LOD\_1\_0 2  Group mode: Clone  Reward mode: Average | 36.4372959 | Logo  Description automatically generated25.8336734 | 47.5702040 | A picture containing shape  Description automatically generated36.4372959 | A picture containing shape  Description automatically generated145.749183 |
| LOD\_1\_1  Group mode: Not clone  Reward mode: Average | 47.0285714  Logo  Description automatically generated | 11.6457142 | Logo  Description automatically generated56.7748979 | 33.9260204 | 135.704081 |
| LOD\_2\_0  Group mode: Clone  Reward mode: Maximum | 91.9987755  Logo  Description automatically generated | 2.70897959 | Logo  Description automatically generated91.9987755 | 30.8365306 | 123.346122 |
| LOD\_2\_1  Group mode: Not clone  Reward mode: Maximum | 67.5675510  **A picture containing shape  Description automatically generated** | 1.56244898 | **A picture containing shape  Description automatically generated**75.5930612 | 31.0185204 | 124.074081 |
| LOD\_3\_0 1  Group mode: Clone  Reward mode: Minimum | 30.7763265 | Logo  Description automatically generated30.7763265 | 42.3973469 | Logo  Description automatically generated36.4311734 | Logo  Description automatically generated145.724693 |
| LOD\_3\_1  Group mode: Not clone  Reward mode: Minimum | 34.1008163 | 23.0051020 | 44.5406122 | 33.0433673 | 132.173469 |

Table 11. Comparison of the different scenarios. Observe that "own score" does not imply the amount of grass collected, but what that agent was awarded due to the reward scheme. For example, an agent might have collected more grass than it received as a reward in the minimum scheme, since another agent collecting less defined that reward.

There are five measurements for each scenario in the above table, and they are ranked using gold, silver, and bronze medals. The performance of individuals is different from who well each member in a group fair. Thus, the average minimal performance could be understood as an indicator for the entire group. Obviously, the reward that every group member receives will always be equal or better than the amount of grass the worst performer collected. Similarly, the total performance (Mean Averages, and Mean Sums) of the group reflects how well groups work together.

With respect to the total number of grass collected (Mean Averages, and Mean sum), we find that the not clonal group with individual payoff fairs best, followed by clonal groups with the average payoff. Interestingly, the third-best performance is achieved by using clonal groups awarded by the minimum. This confirms that this reward scheme is indeed capable of driving groups to perform optimally.

However, when considering how well every member of the group performs, we have to consider the mean Minimum payoff (which is an individual payoff). The clonal minimal rewarding scheme optimizes this parameter the best (place 1 Mean minimum) followed by Clonal groups with the individual payoff, and Clonal groups with the average payoff. While maybe not surprising that the reward scheme that selects for the highest minimal payoff optimizes this parameter the best, it still confirms that this reward scheme is very capable of not only optimizing the least amount each agent collects but more importantly it also compares very well on the total number of grass collected within the entire group (third place in Mean Averages and Mean sum).

When ignoring how well the group performs but focusing on a single individual to collect the most grass, we find the maximum clonal reward scheme to perform best, followed by not clonal groups maximum, and not clonal groups with the average payoff. Again, this might not be surprising as the reward function selects for a single group member to perform optimally, even incentivizing others to give their collected grass to said the best performer.

To sum it up, when we select the clone agent and set the reward scheme as minimum, we get a high average score while ensuring that all group members also fair well. In other words, this combination *trains* our agents to *perform better both individually and in teams at the same time*!

## Minimum and Maximum Selection Schemes and Clonal Groups

As it has been argued before, group-level selection drives cooperation. Here we further find that the two new selection regimes (minimum and maximum) result in the best performances, if the groups are clonal (see Figures 14 and 15).

Figure 14. comparison of clone VS Not clone modes in different rewarding schemes based on the mean of minimum scores. The blue rectangle shows the selection regime matching the y-axis, here the selection scheme rewards the group according to minimal payoff, and shows the highest minimal payoff across all other conditions.

Figure 15. comparison of clone VS Not clone modes in different rewarding schemes based on the mean of maximum scores. The orange rectangle again shows the matching selection regime for the y-axis. Here, the selection regime rewards group members for the maximal performance within the group, which drives the highest individual payoff to be maximal.

While obviously, the selection regime that optimizes for minimal and maximal payoff should drive those outcomes, performance is better in either case, when groups were clonal. In none clonal groups, individuals have to compete against each other, while in clonal groups all members benefit equally from the success of the group, as they are the same genotype. Thus, both selection regimes should be applied to clonal groups as that promises the best results.

## Other observations about clonal groups

So far, we have discussed the maximum and minimum rewarding scheme which was the main aims of this research; now, I will explain the rest of the rewarding schemes, and check if clonal groups evolve better than none clonal groups.

Figure 16. comparison of clone VS Not clone modes in different rewarding schemes based on the mean of own scores. The yellow rectangle again shows the matching selection regime for the y-axis. Here, the selection regime rewards group members for the individual performance within the group.

As you can see in the case of the own-score, when we using the individual reward scheme none clonal groups fair better. Probably because individual rewards benefit selfish agents more. In clonal groups individual benefits would be shared, hampering selfish behavior.

Figure 17. comparison of clone VS Not clone modes in different rewarding schemes based on the mean of average scores. The green rectangle shows the matching selection regime for the y-axis. Here, the selection regime rewards group members for the average performance of the group.

When considering the average score, like in minimum and maximum, groups are dealing with an objective requiring team work. Here clonal groups perform better, as those goals are not selfish goal but succeeding in those benefits the entire group. In general, we can conclude that whenever team work in this context is required, clones are better.

Figure 18. comparison of clone VS Not clone modes in different rewarding schemes based on the mean of the sum of scores. Unlike the other figures in this section here, we do not have a matching rewarding scheme.

We have not a tested a rewarding scheme that selected for the total payoff of the group directly, and consequently, we do not find a single rewarding scheme to optimize this parameter better in clonal or none clonal groups. In the case of average and minimum rewarding schemes, clones have better performance probably because, in general, they are more evolve more optimally when working in teams. The exact opposite ratio happens in the case of maximum and individual rewarding schemes that causes not clones to perform better. As before, we find that using clonal groups allows to evolve behavior that is benefitted by team work easier. At the same time, when the task requires individuals to optimize their own payoff independently of others, not clonal selection is better suited to evolve agents to perform.

|  |  |
| --- | --- |
| Scores | Clonal vs. None clonal groups |
| Individual food collected | None clonal groups evolve better |
| Total food collected as a group | Sometimes clonal groups perform better, sometimes none clonal groups perform better |
| Average food collected with in a group | Mostly clonal groups perform better |
| Maximum and minimum food collected within a group | Clonal groups manage to work together better, and thus fulfil these objectives better |

Table 12. Clonal vs, none clonal groups and their effect on the different scores, such as individual, average, group total, and minimum/maximum score within in group.

## Agents' behaviors

In the comparison of the different scenarios, we have found the best three combinations that were 1. Clone and minimum, 2. Clone and average, and 3. Not clone and individual. We have to ask how those different results are obtained? In order to achieve different payoffs and distributions of payoffs, agents should pursue different strategies and thus should behave differently. For example, in order to maximize the payoff of a single individual (selecting for maximal payoff) one agent should accumulate all rewards, which needs to be organized somehow. Similarly, if selection for minimal payoff is applied, agents should share the workload more equally, again resulting in different behavior. In this part of the results, we will check which actions are the reason behind success or failure in our agents' performance. We have worked on three actions that are explained in the next table.

|  |  |
| --- | --- |
| Action name | Explanation |
| Beep | Beeping is a way for the agents' to communicate. The beeping channel can have different values. When it is zero, they do not use a beep; any other value is considered a beep, but what those values "mean" depends on the individual brain's evolution. Here, I distinguish only beeping from none beeping; what those other values mean is neglected for now. |
| giving grass to other agents or on empty tiles | We have allowed the agents to give their energy (grass) to other agents or simply put it on an empty tile. In this study, we have considered both cases as one action. |
| do nothing | As its name suggests, it means that the agent does nothing in its turn. While not increasing the amount of food of the individual in most cases, it might be used to avoid taking food from someone else – for example, when selecting for maximum payoff. |

Table 13. Agents' actions definitions

|  |  |  |  |
| --- | --- | --- | --- |
| Reward scheme and  group mode | Mean of *giving or putting grasses* percentage | Mean of *do nothing*  percentage | Mean of *beep* percentage |
| Individual  Clone | 0 ste +/- 0 | 0.7005 ste +/- 0.199 | 13.4225 ste +/- 2.801 L |
| Individual  Not clone | 0 ste +/- 0 | 0.136 ste +/- 0.047 | 24.147 ste +/- 3.453 |
| Average  Clone | 0 ste +/- 0 | 0.296 ste +/- 0.112 | 17.156 ste +/- 3.207 |
| Average  Not clone | 0.002 ste +/- 0.001 | 2.077 ste +/- 0.357 \* | 21.978 ste +/- 3.485 |
| Maximum  Clone | 24.948 ste +/- 3.009 \* | 3.304 ste +/- 0.834 \* | 53.814 ste +/- 3.637 H |
| Maximum  Not clone | 2.006 ste +/- 1.210 \* | 0.896 ste +/- 0.248 | 21.8615 ste +/- 3.541 |
| Minimum  Clone | 0 ste +/- 0 | 0.963 ste +/- 0.379 | 51.2315 ste +/- 3.164 H |
| Minimum  Not clone | 0 ste +/- 0 | 8.108 ste +/- 0.671 \* | 9.5385 ste +/- 2.424 L |

Table 14. The Agents' actions are shown as the percentage of actions taken per evaluation. The standard error was calculated and shown. Different conditions that have an effect are highlighted by \*, L, and H. \* indicate conditions different from the rest. The letter L indicates conditions to be low; the letter H indicates conditions to be high with respect to remaining values.

When rewarding groups by their individual or average amount of food collected, we do not expect any cooperative behavior. Payoff can be maximized when every agent individually performs best, and collects the most food. When rewarding the group by the maximum or minimum food collected by an individual on the other hand, coordinating behavior seems to be better, than just trying to collect as much food as possible individually. In the case of maximum, in fact three agents should either do nothing, and leave all food for one agent, or should collect food and hand that collected food over to the one who collected the most food. Either way, also one agent to collect the most food needs to be identified. We find, that agents evolved under this condition, indeed hand over food often. In the maximum selection regime with clonal groups we find agents to give food on average 25% of the time, with a standard error of ~3, suggesting that this behavior evolved across all experimental replicates (see Table 14, left most colum \*). Further, under these selection conditions, we find that these agents also on average do nothing about 3.3% of their time (+/- 0.8 standard error). Which might indicate situations in which one of the agents waits for the leading agent to collect food. We observe the same waiting behavior when selecting for minimum (~8) and average (~2) rewards in none clonal groups (see Table 14, middle column \*), while all other conditions have agents wait less than 1 action per evaluation in total, which we think has an neglectable effect. For the minimal selection criterion it can be again imagined that waiting for another agent, probably that one who collected the least food so far, again provides a benefit. However, how such behavior benefits agents selected for the average food collected is unclear.

Beeping allows agents to communicate with each other, and it can be expected that this helps them to coordinate their actions. For the minimal and maximal selection regime, this kind of coordination could provide a fitness benefit. Thus, it is not surprising to find this notion confirmed, as agents selected in clonal groups indeed beep on average more than 50% of their time (with a standard error of 3) (see Table 14, right most column marked with H). This is about twice as much as other agents beep, for cases where beeping would not convey a fitness benefit. However, we also find that when selecting for individual payoff in clonal groups, as well as in groups selected for minimal payoff none clonal, beeping is reduced to about 10% of their time (with a standard error of ~2.5) (see Table 14, right most column L). It seems that under these conditions beeping is suppressed. However, it is unclear to us how this avoidance of beeping could provide a fitness benefit.

This leads us to conclude that the two new selection regimes (minimum and maximum) not only impact the total or individual payoff, but also change agent behavior to be in general more cooperative or to be better organized.

## Do agents communicate for a reason?

In the previous section, we have identified that the usage of beep, in general, has a positive impact on our agents' performance for clonal groups. But this result was only a correlation, now we wanted confirm that beeping improved performance directly in an experiment. Therefore, we allowed evolved agents to act without any restriction, and compared those results to agents that were prevented from beeping. The beeping channel was muted (forced to 0, regardless of the agent behavior). The agents tested here, were the product of the previous evolutionary experiment, just tested again under different conditions.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Reward  And Group | Beep  Mean Minimums | Mute  Mean Minimums | Beep  Mean Maximums | Mute  Mean Maximums |
| Individual  Clone | 26.09 pvalue=0.000151 | 25.24 | 47.19  pvalue=0.000246 | 46.11 |
| Individual  Not clone | 26.958  pvalue=0.699400 | 26.954 | 47.53  pvalue=0.545742 | 47.43 |
| Mean  Clone | 26.14  pvalue=4.69e-06 | 25.27 | 47.45  pvalue=2.85e-05 | 46.20 |
| Mean  Not clone | 25.41 pvalue=0.699400 | 25.45 | 46.97 pvalue=0.746165 | 46.99 |
| Maximum  Clone | 1.56  pvalue=3.4e-176 | 3.74\* | 92.88 \*  pvalue=0.0 | 34.00 |
| Maximum  Not clone | 25.13  pvalue=0.011563 | 25.57 | 46.89  pvalue=0.192315 | 47.41 |
| Minimum  Clone | 31.51\*  Pvalue = 0 | 8.72 | 41.49\*  pvalue=0.0 | 15.19 |
| Minimum  Not clone | 23.72 pvalue=0.048228 | 23.46 | 43.85  pvalue=0.192315 | 43.47 |

Table 15. Beep vs. mute for the mean of the minimum and maximum scores. The p values of the pairs are presented in the beep columns.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Reward  And Group | Beep  Mean Averages | Mute  Mean Averages | Beep  Mean Sums | Mute  Mean Sums |
| Individual  Clone | 36.40  pvalue=0.000193 | 35.40 | 145.61  pvalue=0.000193 | 141.63 |
| Individual  Not clone | 36.97  pvalue=0.967081 | 36.96 | 147.88  pvalue=0.967081 | 147.84 |
| Mean  Clone | 36.52  pvalue=2.8484e-08 | 35.46 | 146.10  pvalue=2.8484e-08 | 141.86 |
| Mean  Not clone | 35.88  pvalue=0.305041 | 35.92 | 143.55  pvalue=0.305041 | 143.69 |
| Maximum  Clone | 30.00\*  pvalue=0.0 | 16.24 | 120.03\*  pvalue=0.0 | 64.97 |
| Maximum  Not clone | 35.75  pvalue=0.021410 | 36.21 | 143.03, pvalue=0.021410 | 144.87 |
| Minimum  Clone | 36.46\*  pvalue=0.0 | 11.89 | 145.84\*  Pvalue =0.0 | 47.56 |
| Minimum  Not clone | 33.44  pvalue=0.026427 | 33.15 | 133.77  pvalue=0.026427 | 132.61 |

Table 16. Beep vs. mute for the mean of the average and sum scores. The p values of the pairs are presented in the beep columns.

In the case of rewarding according for minimal and maximal in clonal groups we find their performance with respect to the food collected to be highly dependent on beeping. When beeping is muted, clonal groups perform significantly worse. Interestingly, muting communication does not effect the none clonal groups, suggesting that they do not communicate using this channel. Thus, communication evolved only for clonal groups. In itself, this is an interesting find. For communication between genetically unrelated individuals to evolve, it must convey some fitness benefit. Here, such coordination and communication behavior does convey such a benefit, otherwise the clonal groups shouldn't fair better under minimal and maximal reward schemes. However, this potential benefit seems to not be enough for communication to evolve. Or in other words, the benefit from communication and coordination can only be obtained when using clonal groups in the genetic algorithm.

# Discussion

Optimizing how groups of agents or robots perform will become an important challenge the more autonomous machines will populate our environment. Two prominent optimization techniques have been developed: deep learning and neuro evolution. Here, I focus on neuro evolution, with the aim to develop an objective function that trains groups of AI controllers (agents) to perform better *individually* and in *teams*. Even though it is not tested here, objective functions such the ones developed here to be used in a genetic algorithm can also be used in deep-Q learning as those methods also require to properly assess the performance of agents or groups of agents.

Groups of agents are tested in a simple virtual game, where they need to collect food within a confined space. The details of the game and its parameters are explained in the material and methods; however, the environment allows two parameters to be varied. Groups can be made from clones or random individuals from the population. Further for rewarding schemes have been proposed and tested: a) *minimum* rewarding the group according to the performance of the weakest member, *b) maximum* rewarding the group according to the performance of the strongest member, *c) average* groups are rewarded based on the average performance across group members, and *d) Individual* where each member received a reward identical to its own performance. This results in eight experimental conditions, of which the minimum and maximum conditions are new, while other conditions have been used before. It is also known that selection within clonal groups (similar to group-level selection in evolution) drives cooperative behavior. As such, the primary question asked here is about the efficiency of the minimum and maximum rewarding objective function as compared to the other ones.

We found that the minimum reward scheme applied to clonal groups not only works equivalent to average and individually competing groups of none clones. Further, this reward scheme ensures that each agent has on average collected more food than in all other groups compared to. Secondly, when selecting for maximum performance, a single agent of the group collects way more food than in any other condition, but at the expense of the total food collected by that group. As such, we get at least one agent to perform excellently, but others sacrifice their performance to accomplish that goal. Therefore, it has been shown that under these experimental conditions, the minimal reward scheme for clonal groups is a viable alternative to the individual or average reward schemes given to clonal or none clonal groups, with the extra benefit of potentially having an equal payoff distribution among group members.

We also found that those reward schemes that resulted in a high variance of performances during the optimization of the genetic algorithm tend to result in higher performance in the end. It is possible that this variation indicates that more and different strategies get explored, resulting in the possibility to choose among more solutions. However, at this point, this is just an observation and needs to be explored further.

Another interesting observation is that when using clonal groups, the minimum and maximum reward scheme result in overall higher performance. While it has been shown that group-level selection improves cooperation, it is new that it also improves the kind of cooperation required to solve this task when using the minimal and maximal reward scheme. One can speculate that when using none clonal groups, competition between group members is maintained, as selection during the evolutionary process remains selfish. When using clonal groups, each member of the group has the same genome, and thus differences between the group members are irrelevant during the selection process. Again, further analysis of this phenomenon is in order, as at this stage, the result is observational.

Since the minimum and maximum reward schemes require agents to coordinate behavior, we allowed agents to communicate using a far-ranging beeping signal. Agents evolved under those conditions beep more often and also lose their performance if those signals are muted. This proves that beeping is used to coordinate behavior among clonal group members. When not using clonal groups, the use of beeping to coordinate behavior does not evolve under these conditions. When thinking about possible future tasks of robots who have to work on complex tasks that also require them to communicate, this result suggests that using clonal groups will allow them to evolve communication easier. However, here we only correlated beeping to function and did not further study the actual nature of the signal. In theory, a binary communication channel like the one provided here might allow for a more sophisticated kind of communication. For example, under the maximum reward scheme, an individual can be identified to collect all the food. How this works has not been studied here, again suggesting a further investigation into this matter.

However, we must mention that these conclusions are based on the specific experiments conducted here. While they might generalize, we should do more experiments. The environment, for example, is a simple food collection game, and a more complex environment are easy to imagine. Similarly, only Markov Brains were used, which present a subset of all possible computational neural controllers. Lastly, the genetic algorithm used simple roulette wheel selection, but much more sophisticated methods such as map-elites (citation) and novelty search (citation) as well as methods for genetic recombination or sexual selection (cite) are possible. Further, it might be tempting to generalize these results to human behavior, suggesting that rewarding groups according to the performance of the weakest member leads to better performance or an equal distribution of resources. An interesting perspective that again warrants further investigation.

This model, specifically the minimum and maximum reward schemes, might allow the study of altruistic and selfish behavior. If collected food is handed over to other agents, without a direct fitness benefit to the one who gives the food, we would truly observe altruism. At the same time, such true altruism is believed to not be able to evolve, as only things that are neutral or beneficial will prevail during evolution. As such, different conditions that reward seemingly altruistic behavior have been suggested, such as reciprocity [4] or the green beard effect (cite). Here, the benefits come from the selection regime that rewards effective groups over selfish individuals. Studying these effects in more detail is another application of the approach presented here.

# Conclusions

The goal of identifying a new reward scheme that improves the individual, as well as the group's performance, has succeeded. Rewarding clonal groups by the performance of the individual that collected the least amount of food (minimum reward scheme) drive the total number of food collected up and also ensures that each member on average performs equally or better than under control conditions

We could do go beyond the initial research aim and studied three more aspects. Firstly, the high variance in the agents' scores through the generations causes better performance at the end. The high variance in the scores is probably because of testing a wider variety of methods that enhance the agents' evolution. Secondly, when we set rewarding schemes to minimum, maximum, and average, the clones have better performance, whereas non-clones are better for the individual reward scheme. So, clones are better for teamwork. Finally, we have identified that there is a relationship between agents' scores and their actions (beep, do nothing, and giving food to others or empty tiles). In general, communication or usage of beep causes better performance for the agents. This work is a proof of concept and sets the stage for a series of more detailed investigations.

As we have discussed, the greater variance in agents' performance in their evolutionary journey might result from testing different approaches by different generations. We could check the agents' behaviors, but these actions are not linked to generations to identify the changes in the agents' actions from different generations. So, this study's first future work is identifying the variations in the agents' behavior through their evolution. Our expected outcome is first, in successful combinations such as minimum rewarding scheme for clonal agents, we should have very different actions in different generations. Secondly, there should be a growth in the amount of communication (usage of beeps) their evolution in case of best combinations. The other observation of our study was the better performance of clones relative to different agents in teamwork. Identification of all the reasons behind this fact is the second priority of our future work. Indeed, there are some expectations in this aspect, such as more altruistic behaviors among clones because they do not have a competition in reproducing offsprings. We have found that communication enhances the agents' performance, which is more promising in the case of clones. Why can clones communicate better? Finding an answer to this question is another future work. In this regard, it is also interesting to decode their language to identify how the agents talk to each other. Apart from those three future works putting the evolved agents in our environment in different worlds and testing their performance is fascinating. For example, it is very important to know if we use those who are evolved using minimum rewarding scheme reminds supportive if we reward them for their individual payoff.

As the final point of this thesis, maybe we could not establish a school in which all students support the weakest one, and at the same time the school is among the top ten schools, but we could find the method to motivate all the AI controls to help their weakest member, and it happens that they are also one of the most successful groups as well!

# References

|  |  |
| --- | --- |
| [1] | L. Joel, C. Jeff and M. Dusan, "The surprising creativity of digital evolution," in *Artificial Life Conference Proceedings 14, pages 76-83. MIT Press*, 2017. |
| [2] | A. Hintze and M. Mirmomeni, "Evolution of autonomous hierarchy formation and maintenance," in *Artficial Life Conference Proceedings 14, pages 366-367. MIT Press*, 2014. |
| [3] | B. Gergely and S. Szabolcs, "Beneficial laggards: multilevel selection, cooperative polymorphism and division of labour in threshold public good games," *BMC evolutionary biology,* 2010. |
| [4] | M. Nowak, ”Five Rules for the Evolution of Cooperation,” *American Association for the Advancement of Science,* vol. 314, nr 5805, 2015. |
| [5] | A. Hintze, J. A. Edlund, S. O. Randal, B. K. David, S. Jory, A. Larissa, T. S. Ali, K. Peter, S. e. Leigh och G. Heather, ”Markov brains: A technical introduction,” 2017. |
| [6] | J. A Edlund, N. Chaumont, A. Hintze, C. Koch, G. Tononi och C. Adami, ”Integrated information increases with fitness in the evolution of animats,” *PLoS Computational Biology,* vol. 7, nr 10, 2011. |
| [7] | R. S. Olson, A. Hintze, F. C. Dyer, D. B. Knoester och C. Adami, ”Predator confusion is sufficient to evolve swarming behavior,” *J. Royal Society Interface,* 2013. |
| [8] | B. Clifford and H. Arend, "Mabe (modular agent based evolver): A framework for digital evolution research," in *Artificial Life Conference Proceedings*, 2017. |

# Appendix: the source codes

## lodAnalyzer.py

import matplotlib.pyplot as plt

import pandas as pd

# Settings:

# Folder's path of the LOD files.

filesFolder="C:/Users/blue/Documents/study/BI/Thesis/allOne/data/lod/second/"

# Folder's path of the graphs

graphFolder="C:/Users/blue/Documents/study/BI/Thesis/allOne/results/graph/second/"

# Folder's path of the lod statistics

lodStatisticsFolder="C:/Users/blue/Documents/study/BI/Thesis/allOne/results/lodStatistics/second/"

# number of replicate

numberOfReplicate = 99

replicateNumbers = range(numberOfReplicate+1)

# reward modes:

rewardModes = [0,1,2,3]

# group modes:

groupModes = [0,1]

# End of settings

# 1. Finding the maximum length of the lod files

lodLengths = []

maxLodLengths = 0

for rewardMode in rewardModes:

for groupMode in groupModes:

for replicateNumber in replicateNumbers:

fileName = "LOD"+"\_"+str(rewardMode)+"\_"+str(groupMode)+"\_"+str(replicateNumber)

df = pd.read\_csv(filesFolder+fileName+".csv")

lodLengths.append(len(df.index))

maxLodLengths = max(lodLengths)

# 2. Making the lod files same number of rows

# We do this step to avoid dimension error in plotting

for rewardMode in rewardModes:

for groupMode in groupModes:

for replicateNumber in replicateNumbers:

fileName = "LOD"+"\_"+str(rewardMode)+"\_"+str(groupMode)+"\_"+str(replicateNumber)

df = pd.read\_csv(filesFolder+fileName+".csv")

newRow = df.iloc[-1]

if len(df.index) < maxLodLengths:

for i in range(maxLodLengths-len(df.index)):

newRow[:1] = newRow[:1] + 100

df = df.append(newRow)

df.to\_csv(filesFolder+fileName+".csv", index=False)

# 3. Plotting the graphs with generations as x axis and score for y axis

x = range(0, maxLodLengths\*100, 100)

for rewardMode in rewardModes:

for groupMode in groupModes:

plotName = "LOD"+"\_"+str(rewardMode)+"\_"+str(groupMode)

for replicateNumber in replicateNumbers:

fileName = "LOD"+"\_"+str(rewardMode)+"\_"+str(groupMode)+"\_"+str(replicateNumber)

df = pd.read\_csv(filesFolder+fileName+".csv")

plt.plot(x, df["score"])

plt.title(plotName)

plt.ylim(0, 150)

plt.xlabel('Generation')

plt.ylabel('Score')

plt.savefig(graphFolder+plotName+'.png')

plt.clf()

# 4. Calculation of mean of minimum, maximum, sum and average of each conditoin (like 0\_0, 0\_1, 1\_0, 1\_1, ....).

# 4.1 Adding the minimum, maximum, sum and average of rowScores to each row (each generation)

rawScores = []

minimums = []

maximums = []

averages = []

sums = []

for rewardMode in rewardModes:

for groupMode in groupModes:

plotName = "LOD"+"\_"+str(rewardMode)+"\_"+str(groupMode)

for replicateNumber in replicateNumbers:

fileName = "LOD"+"\_"+str(rewardMode)+"\_"+str(groupMode)+"\_"+str(replicateNumber)

df = pd.read\_csv(filesFolder+fileName+".csv")

for i in range(maxLodLengths):

df["rawScores"][i] = df["rawScores"][i].replace('[','')

df["rawScores"][i] = df["rawScores"][i].replace(']','')

rawIntegers = df["rawScores"][i].split(',')

rawScores =[float(rawIntegers[0]),float(rawIntegers[1]),float(rawIntegers[2]),float(rawIntegers[3])]

minimums.append(min(rawScores))

maximums.append(max(rawScores))

averages.append(sum(rawScores)/4)

sums.append(sum(rawScores))

df['minimumScore'] = minimums

df['maximumScore'] = maximums

df['averageOfScores'] = averages

df['sumOfScores'] = sums

df.to\_csv(filesFolder+fileName+".csv", index=False)

minimums.clear()

maximums.clear()

averages.clear()

sums.clear()

# 4.2 Calculation of the mean of minimum, maximum, sum and average of all the replicates of each condition.

allReplicates = pd.DataFrame (columns = ['generation','ID','score','rawScores','ownScore','minimumScore','maximumScore','averageOfScores','sumOfScores'])

rewardGroupModes = []

meanOwnScores = []

meanMinimums = []

meanMaximums = []

meanAverages = []

meanSums = []

for rewardMode in rewardModes:

for groupMode in groupModes:

for replicateNumber in replicateNumbers:

fileName = "LOD"+"\_"+str(rewardMode)+"\_"+str(groupMode)+"\_"+str(replicateNumber)

df = pd.read\_csv(filesFolder+fileName+".csv")

allReplicates = allReplicates.append(df)

rewardGroupModes.append("LOD"+"\_"+str(rewardMode)+"\_"+str(groupMode))

meanOwnScores.append(sum(allReplicates["ownScore"])/((numberOfReplicate+1)\*maxLodLengths))

meanMinimums.append(sum(allReplicates["minimumScore"])/((numberOfReplicate+1)\*maxLodLengths))

meanMaximums.append(sum(allReplicates["maximumScore"])/((numberOfReplicate+1)\*maxLodLengths))

meanAverages.append(sum(allReplicates["averageOfScores"])/((numberOfReplicate+1)\*maxLodLengths))

meanSums.append(sum(allReplicates["sumOfScores"])/((numberOfReplicate+1)\*maxLodLengths))

allReplicates.drop(allReplicates.index, inplace=True)

dict = {'rewardGroupModes': rewardGroupModes, 'meanOwnScores': meanOwnScores, 'meanMinimums': meanMinimums,'meanMaximums': meanMaximums,'meanAverages': meanAverages,'meanSums': meanSums}

lodsStatistics = pd.DataFrame(dict)

lodsStatistics.to\_csv(lodStatisticsFolder+"lodStatistics"+".csv", index=False)

## movementAnalyzer.py

import matplotlib.pyplot as plt

import pandas as pd

import numpy

import math

# Settings:

# Folder's path of the movement files.

filesFolder="C:/Users/blue/Documents/study/BI/Thesis/allOne/data/mov/second/"

# Folder's path of the movement statistics

movStatisticsFolder="C:/Users/blue/Documents/study/BI/Thesis/allOne/results/movStatistics/second/"

# number of replicate

numberOfReplicate = 99

replicateNumbers = range(numberOfReplicate+1)

# reward modes:

rewardModes = [0,1,2,3]

# group modes:

groupModes = [0,1]

# End of settings

# 1. Finding the percentage of the following values:

# I. Number of times that action (a columns) has higher value than 3

# II. Number of times that action (a columns) iz zero

# II. Number of times that b is not zero

movStatisticsLog = pd.DataFrame (columns = ['fileName','movLength\*4','aBiggerThanThree','aBiggerThanThreePercentage','aZero','aZeroPercentage','bNotZero','bNotZeroPercentage'])

movStatistics = pd.DataFrame (columns = ['rewardGroup','meanABiggerThanThreePercentage','stdErrMeanABiggerThanThreePercentage',

'meanAZeroPercentage','stdErrMeanAZeroPercentage',

'meanBNotZeroPercentage','stdErrMeanBNotZeroPercentage'])

movLength = 0

aBiggerThanThree = 0

aBiggerThanThreePercentage = 0

aZero = 0

aZeroPercentage = 0

bNotZero = 0

bNotZeroPercentage = 0

aBiggerThanThreePercentageS = []

aZeroPercentageS = []

bNotZeroPercentageS = []

for rewardMode in rewardModes:

for groupMode in groupModes:

for replicateNumber in replicateNumbers:

fileName = "movement"+"\_"+str(rewardMode)+"\_"+str(groupMode)+"\_"+str(replicateNumber)+".csv"

rewardGroup = "movement"+"\_"+str(rewardMode)+"\_"+str(groupMode)

df = pd.read\_csv(filesFolder+fileName)

movLength = len(df.index)

for i in range(len(df)) :

# check if a iz bigger than 3

if(df.loc[i, "a0"]>3):

aBiggerThanThree = aBiggerThanThree + 1

if(df.loc[i, "a1"]>3):

aBiggerThanThree = aBiggerThanThree + 1

if(df.loc[i, "a2"]>3):

aBiggerThanThree = aBiggerThanThree + 1

if(df.loc[i, "a3"]>3):

aBiggerThanThree = aBiggerThanThree + 1

# Check if a is zero

if(df.loc[i, "a0"]==0):

aZero = aZero + 1

if(df.loc[i, "a1"]==0):

aZero = aZero + 1

if(df.loc[i, "a2"]==0):

aZero = aZero + 1

if(df.loc[i, "a3"]==0):

aZero = aZero + 1

# Check if b is NOT zero

if(df.loc[i, "b0"]!=0):

bNotZero = bNotZero + 1

if(df.loc[i, "b1"]!=0):

bNotZero = bNotZero + 1

if(df.loc[i, "b2"]!=0):

bNotZero = bNotZero + 1

if(df.loc[i, "b3"]!=0):

bNotZero = bNotZero + 1

aBiggerThanThreePercentage = (aBiggerThanThree/(movLength\*4))\*100

aZeroPercentage = (aZero/(movLength\*4))\*100

bNotZeroPercentage = (bNotZero/(movLength\*4))\*100

aBiggerThanThreePercentageS.append(aBiggerThanThreePercentage)

aZeroPercentageS.append(aZeroPercentage)

bNotZeroPercentageS.append(bNotZeroPercentage)

#print(fileName," ",aBiggerThanThree," ",movLength\*4," ",aBiggerThanThreePercentage, " ",aZero)

movStatisticsLog = movStatisticsLog.append({'fileName': fileName, 'movLength\*4' : movLength\*4,'aBiggerThanThree' : aBiggerThanThree, 'aBiggerThanThreePercentage' : aBiggerThanThreePercentage,'aZero':aZero,'aZeroPercentage':aZeroPercentage,'bNotZero':bNotZero,'bNotZeroPercentage':bNotZeroPercentage}, ignore\_index=True)

movLength = 0

aBiggerThanThree = 0

aBiggerThanThreePercentage = 0

aZero = 0

aZeroPercentage = 0

bNotZero = 0

bNotZeroPercentage = 0

#print(sum(aBiggerThanThreePercentageS)/(numberOfReplicate+1))

movStatistics = movStatistics.append({'rewardGroup': rewardGroup,

'meanABiggerThanThreePercentage': sum(aBiggerThanThreePercentageS)/(numberOfReplicate+1),

'stdErrMeanABiggerThanThreePercentage': numpy.std(aBiggerThanThreePercentageS)/math.sqrt(numberOfReplicate+1),

'meanAZeroPercentage': sum(aZeroPercentageS)/(numberOfReplicate+1),

'stdErrMeanAZeroPercentage': numpy.std(aZeroPercentageS)/math.sqrt(numberOfReplicate+1),

'meanBNotZeroPercentage': sum(bNotZeroPercentageS)/(numberOfReplicate+1),

'stdErrMeanBNotZeroPercentage': numpy.std(bNotZeroPercentageS)/math.sqrt(numberOfReplicate+1),

},

ignore\_index=True)

aBiggerThanThreePercentageS.clear()

aZeroPercentageS.clear()

bNotZeroPercentageS.clear()

#print(movStatisticsLog)

#movStatisticsLog.to\_csv(movStatisticsFolder+"movStatisticsLog"+".csv", index=False)

#print(movStatistics)

movStatistics.to\_csv(movStatisticsFolder+"movStatistics"+".csv", index=False)

## beepAnalyzer.py

import matplotlib.pyplot as plt

import pandas as pd

from scipy import stats

# Settings:

# Folder's path of the LOD files.

filesFolder="C:/Users/blue/Documents/study/BI/Thesis/allOne/data/beep/second/"

# Folder's path of the lod statistics

beepStatisticsFolder="C:/Users/blue/Documents/study/BI/Thesis/allOne/results/beepStatistics/second/"

# number of replicate

numberOfReplicate = 99

replicateNumbers = range(numberOfReplicate+1)

# reward modes:

rewardModes = [0,1,2,3]

# group modes:

groupModes = [0,1]

# End of settings

bValues = []

bMinimums = []

bMaximums = []

bAverages = []

bSums = []

mValues = []

mMinimums = []

mMaximums = []

mAverages = []

mSums = []

# 1. Adding the minimum, maximum, sum and average of each row for beep and muted cases.

for rewardMode in rewardModes:

for groupMode in groupModes:

for replicateNumber in replicateNumbers:

fileName = "beep"+"\_"+str(rewardMode)+"\_"+str(groupMode)+"\_"+str(replicateNumber)

df = pd.read\_csv(filesFolder+fileName+".csv")

for i in range(numberOfReplicate+1):

bValues.append(df["b0"][i])

bValues.append(df["b1"][i])

bValues.append(df["b2"][i])

bValues.append(df["b3"][i])

bMinimums.append(min(bValues))

bMaximums.append(max(bValues))

bAverages.append(sum(bValues)/4)

bSums.append(sum(bValues))

bValues.clear()

mValues.append(df["m0"][i])

mValues.append(df["m1"][i])

mValues.append(df["m2"][i])

mValues.append(df["m3"][i])

mMinimums.append(min(mValues))

mMaximums.append(max(mValues))

mAverages.append(sum(mValues)/4)

mSums.append(sum(mValues))

mValues.clear()

df['mMinimumScore'] = mMinimums

df['bMinimumScore'] = bMinimums

df['mMaximumScore'] = mMaximums

df['bMaximumScore'] = bMaximums

df['mAverageOfScores'] = mAverages

df['bAverageOfScores'] = bAverages

df['mSumOfScores'] = mSums

df['bSumOfScores'] = bSums

df.to\_csv(filesFolder+fileName+".csv", index=False)

bMinimums.clear()

bMaximums.clear()

bAverages.clear()

bSums.clear()

mMinimums.clear()

mMaximums.clear()

mAverages.clear()

mSums.clear()

# 2. Calculation of the mean of minimum, maximum, sum and average of all the replicates of each condition.

allReplicates = pd.DataFrame (columns = ['replicate','m0','m1','m2','m3','b0','b1','b2','b3',

'mMinimumScore','bMinimumScore','mMaximumScore','bMaximumScore',

'mSumOfScores','bSumOfScores','mAverageOfScores','bAverageOfScores',

'MinimumsKs','MaximumsKs','AveragesKs','SumsKs'])

rewardGroupModes = []

bMeanMinimums = []

bMeanMaximums = []

bMeanAverages = []

bMeanSums = []

mMeanMinimums = []

mMeanMaximums = []

mMeanAverages = []

mMeanSums = []

MinimumsKs = []

MaximumsKs = []

AveragesKs = []

SumsKs = []

for rewardMode in rewardModes:

for groupMode in groupModes:

for replicateNumber in replicateNumbers:

fileName = "beep"+"\_"+str(rewardMode)+"\_"+str(groupMode)+"\_"+str(replicateNumber)

df = pd.read\_csv(filesFolder+fileName+".csv")

allReplicates = allReplicates.append(df)

rewardGroupModes.append("beep"+"\_"+str(rewardMode)+"\_"+str(groupMode))

bMeanMinimums.append(sum(allReplicates["bMinimumScore"])/((numberOfReplicate+1)\*(numberOfReplicate+1)))

bMeanMaximums.append(sum(allReplicates["bMaximumScore"])/((numberOfReplicate+1)\*(numberOfReplicate+1)))

bMeanAverages.append(sum(allReplicates["bAverageOfScores"])/((numberOfReplicate+1)\*(numberOfReplicate+1)))

bMeanSums.append(sum(allReplicates["bSumOfScores"])/((numberOfReplicate+1)\*(numberOfReplicate+1)))

mMeanMinimums.append(sum(allReplicates["mMinimumScore"])/((numberOfReplicate+1)\*(numberOfReplicate+1)))

mMeanMaximums.append(sum(allReplicates["mMaximumScore"])/((numberOfReplicate+1)\*(numberOfReplicate+1)))

mMeanAverages.append(sum(allReplicates["mAverageOfScores"])/((numberOfReplicate+1)\*(numberOfReplicate+1)))

mMeanSums.append(sum(allReplicates["mSumOfScores"])/((numberOfReplicate+1)\*(numberOfReplicate+1)))

MinimumsKs.append(stats.ks\_2samp(allReplicates["bMinimumScore"],allReplicates["mMinimumScore"]))

MaximumsKs.append(stats.ks\_2samp(allReplicates["bMaximumScore"],allReplicates["mMaximumScore"]))

AveragesKs.append(stats.ks\_2samp(allReplicates["bAverageOfScores"], allReplicates["mAverageOfScores"]))

SumsKs.append(stats.ks\_2samp(allReplicates["bSumOfScores"], allReplicates["mSumOfScores"]))

allReplicates.drop(allReplicates.index, inplace=True)

dict = {'rewardGroupModes': rewardGroupModes,

'bMeanMinimums': bMeanMinimums,

'bMeanMaximums': bMeanMaximums,

'bMeanAverages': bMeanAverages,

'bMeanSums': bMeanSums,

'mMeanMinimums': mMeanMinimums,

'mMeanMaximums': mMeanMaximums,

'mMeanAverages': mMeanAverages,

'mMeanSums': mMeanSums,

'MinimumsKs' : MinimumsKs,

'MaximumsKs' : MaximumsKs,

'AveragesKs' : AveragesKs,

'SumsKs' : SumsKs

}

beepStatistics = pd.DataFrame(dict)

#print(beepStatistics)

beepStatistics.to\_csv(beepStatisticsFolder+"beepStatistics"+".csv", index=False)