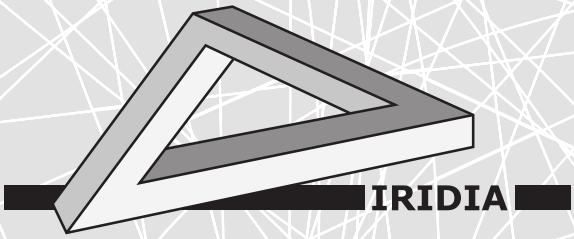

Université Libre de Bruxelles

Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

AutoMoDe, NEAT, and EvoStick: Implementations for the E-puck Robot in ARGoS3

A. Ligot, K. Hasselmann, B. Delhaisse,
L. Garattoni , G. Francesca, and M. Birattari

IRIDIA – Technical Report Series

Technical Report No.
TR/IRIDIA/2017-002

January 2017
Last revision: April 2018

IRIDIA – Technical Report Series
ISSN 1781-3794

Published by:

IRIDIA, *Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle*
Université Libre de Bruxelles
Av F. D. Roosevelt 50, CP 194/6
1050 Bruxelles, Belgium

Technical report number TR/IRIDIA/2017-002

Revision history:

TR/IRIDIA/2017-002.001	January 2017
TR/IRIDIA/2017-002.002	April 2018
TR/IRIDIA/2017-002.003	April 2018

AutoMoDe, NEAT, and EvoStick: Implementations for the E-puck Robot in ARGoS3

Antoine Ligot*, Ken Hasselmann, Brian Delhaisse,
Lorenzo Garattoni, Gianpiero Francesca, and Mauro Birattari

IRIDIA, CoDE, Université Libre de Bruxelles, Brussels, Belgium.

January 2017

1 Introduction

This document describes the packages ARGoS3-AutoMoDe and ARGoS3-NEAT, the implementations of AutoMoDe and NEAT for the ARGoS3 simulator and the e-puck robot. This document also describes `EvoStick`, a specific instance of NEAT.

ARGoS3 (Pinciroli et al., 2012) is a widespread swarm robotics simulator. The main advantages over competitors are its modular architecture, its efficiency with large groups of robots and the ease it provides when transferring control software from simulation to real robots.

The e-puck (Mondada et al., 2009) is a small two-wheeled robot initially developed for education purposes. Its low price and multiple functionalities make this robot commonly used in swarm robotics research.

The e-puck robot can be used in ARGoS3 via the e-puck plugin for ARGoS3 described in Garattoni et al. (2015). This plugin enables the use of a particular version of the original e-puck robot in both simulation and reality. This version of the e-puck comprises 8 infrared transceivers, 3 ground sensors, a range-and-bearing board, an omnidirectional camera and 3 colored LEDs. The infrared transceivers are used by the e-puck to detect the presence of close obstacles and measure the intensity of surrounding light. The ground sensors allow the e-puck to detect the color of the ground below itself. The range-and-bearing board allows the e-puck to communicate with robots in a range of approximately 0.70m. For each received message, the range-and-bearing board computes the distance (range) and relative angle (bearing) of the emitting e-puck. The omnidirectional camera offers a 360 degree view of the surrounding of the e-puck. More details on the capabilities of the improved version of the e-puck can be found in Garattoni et al. (2015).

*corresponding author: aligot@ulb.ac.be

The rest of the document is organized as follows: Section 2 is dedicated to the description of the ARGoS3-AutoMoDe package. Section 3 is dedicated to the description of the ARGoS3-NEAT package. Section 4 is dedicated to the description of `EvoStick`. Section 5 is dedicated to the description of an aggregation mission used as a running example in the different packages descriptions.

2 The ARGoS3-AutoMoDe package

AutoMoDe (automatic modular design) is an automatic design approach for robot swarms. AutoMoDe generates, via an optimization algorithm, control software in the form of a probabilistic finite state machine (PFSM). These PFSMs are created by searching for the best combination of preexisting *modules* and the values of their parameters.

In the ARGoS3-AutoMoDe package, we implemented the six *behaviors* and the six *conditions* modules described in Francesca et al. (2014) and Francesca et al. (2015) as constituent modules of `Vanilla` and `Chocolate`. One will also find the necessary elements needed to run the five different experiments described in Francesca et al. (2015). These elements comprise the experiment configurations files, the loop functions classes and the adaptations to irace (López-Ibánez et al., 2011).

The remaining of this section is articulated as follows: Section 2.1 describes the installation procedure and Section 2.2 explains how ARGoS3-AutoMoDe works.

2.1 Installation

In this section, we describe how the user has to proceed in order to download and compile the ARGoS3-AutoMoDe package. The steps to follow will only work if the requirements described in Section 2.1.1 are met. Instructions for compiling the ARGoS3-AutoMoDe package for simulations and real robots purposes are described in Sections 2.1.3 and 2.1.4 respectively.

2.1.1 Requirements

The user needs a GNU/Linux or UNIX-like operating system as Windows in not supported.

Additionally, the user will need to download, compile and install the latest version of ARGoS3 (Pinciroli et al., 2012):

```
https://github.com/ilpincy/argos3
```

and the plugin for the E-pucks robots, ARGoS3-Epuck (Garattoni et al., 2015):

```
https://github.com/lgarattoni/argos3-epuck
```

Finally, the user should follow the installation instructions of irace (López-Ibánez et al., 2011) in its user guide[1].

[1] See http://iridia.ulb.ac.be/irace/

2.1.2 Downloading

The development sources are accessible through git:

```
$ git clone https://github.com/demiurge-project/ARGoS3-
    AutoMoDe argos3-AutoMoDe
```

2.1.3 Compiling for the simulator

The following commands will result in the creation of an executable, `automode_main`. This executable will be placed in the *bin* folder[2] and can be used to launch an experiment with a description of a PSFM.

```
$ cd argos3-AutoMoDe
$ mkdir build
$ cd build
$ cmake ..
$ make
```

2.1.4 Compiling for the robots

```
$ cd argos3-AutoMoDe
$ mkdir build
$ cd build
$ cmake -DCMAKE_TOOL_CHAIN=../src/cmake/TargetEPuck.cmake ..
$ make
```

2.2 Example

In this section we explain how to use the ARGoS3-AutoMoDe package. In Section 2.2.1 we describe the different options that can be used with ARGoS3-AutoMoDe, Section 2.2.2 illustrates the functioning of ARGoS3-AutoMoDe through a small example and Section 2.2.4 explains how to use ARGoS3-AutoMoDe with irace.

2.2.1 Options

The ARGoS3-AutoMoDe options can be specified by command line of the executable created (i.e. ~/argos3-AutoMoDe/bin/`automode_main`) using flags or by setting them in the experiment configuration file (.argos) [3]. In the latter case, the options have to be specified in the `params` node of the `controllers` section of the experiment configuration file. The following list contains the different options for AutoMoDe.

`fsm-config` *flag:* `--fsm-config` *default:* `""` `[MANDATORY]`
The description of a PFSM.

[2] That is ~/argos3-AutoMoDe/bin

[3] See http://www.argos-sim.info/user_manual.php for more details about ARGoS3 and how it works.

`radom_seed` *flag:* `--seed` or `-s` *default:* 0 `[OPTIONAL]`

The random seed that is used by ARGoS3.
If not present or set to 0, the value of the random seed used is taken from the internal clock time. Specifying a value is necessary to obtain the same results for repetitions of an experiment.
Differently from the other options, the value of `random_seed` has to be specified in the `experiment` node of the `framework` section of the configuration file.
Specifying a value for the random seed as command line argument will overwrite the value specified in the configuration file.

`readable` *flag:* `--readable-fsm` or `-r` *default:* `"false"` `[OPTIONAL]`

When set to `true`, AutoMoDe will display a URL containing a DOT description of the PFSM given in parameter. The user can copy-paste this URL in its favorite web browser to have a graphical representation of the PFSM. See Figure 2 for an example.
It is important to note that the URL will be displayed as many times as there are robots when this option is specified via the experiment file.

`history` *default:* `"false"` `[OPTIONAL]`

Enable/disable the recording of the visited states of the PFSM.
When enabled, ARGoS3-AutoMoDe will create a file for each epuck containing its behavioral history. This history file will contain, for each time step, the behavioral module controlling the robot along side with the conditional modules tested and their values (either O or 1).
Currently, this option can only by specified from the experiment file (.argos).

`history-folder` *default:* `"./"` `[OPTIONAL]`

The folder where the history of each robot will be saved.

2.2.2 Running the example

In this section we show how to use ARGoS3-AutoMoDe with an example. The example is the aggregation mission described in Section 5. ARGoS3-AutoMoDe requires two elements in order to work: an experiment configuration file (.argos) and a *loop functions* class. The user can find these two elements for the example mission in a designated folder in the *experiments* and *loop-functions* folders of the package [4].

For the loop functions class, the user has to create a class that inherits from the *AutoMoDeLoopFunctions*[5] and that implements at least two methods: *GetObjectiveFunction()* and *GetRandomPosition()*. The first method should return the fitness of the controller while the second one is in charge of returning a position vector for a robot in the arena. Our implementation of the *GetRandomPosition()* method allows `number_robots` robots to be uniformly distributed in

[4] That is ~/argos3-AutoMoDe/experiments/example/example_aggregation_visu.argos for the experiment configuration file and ~/argos3-AutoMoDe/experiments/example/ExampleAggregationLoopFunc.* for the loop functions class.

[5] Located in ~/argos3-AutoMoDe/src/core/

a circle of radius `dist_radius`. The value of these parameters can be changed in the *params* node of the *loop_functions* section in the experiment configuration file.

For convenience, the provided experiment configuration file contains a PFSM description. In order to use that PFSM as the robots controller, the user can follow the steps:

```
$ cd ~/argos3-AutoMoDe/experiments/example
$ argos3 -c example_aggregation_visu.argos
```

The user can also launch the same experiment with the executable `automode-main`. In this case, the PFSM configuration has to be specified as command line argument (see Section 2.2.1). In the instructions below, the PFSM following the `--fsm-config` flag consists in a single state machine.

```
$ cd ~/argos3-AutoMoDe/experiments/example
$ ../../bin/automode-main -c example_aggregation_visu.argos
   --fsm-config --nstates 1 --s0 4 --att0 5
```

2.2.3 Encoding of the Probabilistic Finite State Machines

The PFSM considered by AutoMoDe are encoded as chains of characters written in a language that consists in the succession of variable-value pairs.

A PFSM description is composed of the specification of the number of states S (`--nstates`), followed by the successive description of the behavior modules. The description of the behavior modules comprises the encoding of the behavior itself and the encoding of its outgoing transition modules. The nomenclature of the variables composing the description of a behavior module consists of the concatenation of a variable identifier with the index of the module in the PFSM. We note b the index of the behavior modules, with $b \in [0,\text{S})$. The encoding of a behavior module is composed of the behavior identifier (`--s`b), its potential parameter (see Table 1), and the number of outgoing transitions T_b (`--n`b).

Similarly, the variables describing the outgoing transitions of a behavior module are written as the concatenation of the variable identifier, the index of the behavior module and the index of the outgoing transition within the behavior module, with both indexes separated by the character `x`. We note t_b the index of the transitions going out of behavior module b, with $t_b \in [0,\text{T}_b)$, and we define V_b as the vector containing the possible behavior indexes to which a transition t_b can point. Since the origin and destination behaviors of a transition should be different, we have $\text{V}_b = \{0,...,\text{S}-1\} \setminus \{b\}$. The description of an outgoing transition is composed of the index of an arrival behavior module in V_b (`--n`b`x`t_b), the transition identifier (`--c`b`x`t_b) and its potential parameters (see Table 2).

Identifier	Behavior	Parameter
0	Exploration	`--rwm`b
1	Stop	none
2	Phototaxis	none
3	Anti-Phototaxis	none
4	Attraction	`--att`b
5	Repulsion	`--rep`b

Table 1: Identifier and parameters of the behaviour modules with $b \in [0, S)$.

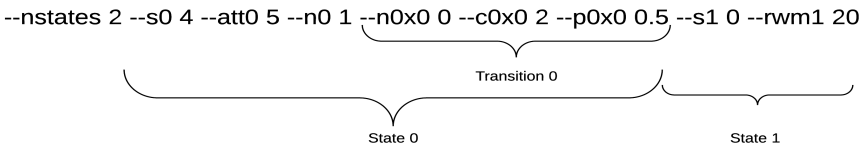

Figure 1: Example of PFSM description. The PFSM is composed of two behavior modules – *attraction* and *exploration* – and one transition module – *white floor*. The destination behavior index of the transition, specified by `--n0x0`, equals to 0. Since $V_b = \{0, 1\} \setminus \{0\} = \{1\}$, the transition goes to the state 1, hence from *attraction* to *exploration*.

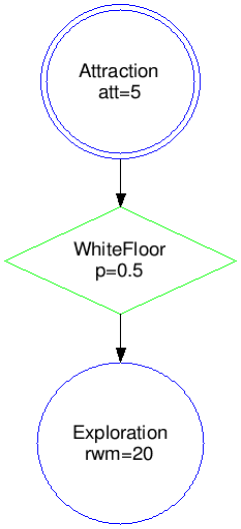

Figure 2: Example of a PFSM generated with the `readable` option. Corresponds to the description of the PFSM of Figure 1. The robots starts in the behavior circled twice.

Identifier	Condition	Parameter
0	Black floor	`--p`bxt_b
1	Gray floor	`--p`bxt_b
2	White floor	`--p`bxt_b
3	Neighbors-count	`--p`bxt_b and `--w`bxt_b
4	Inverted-neighbors-count	`--p`bxt_b and `--w`bxt_b
5	Fixed-probability	`--p`bxt_b

Table 2: Identifier and parameters of the transition modules with $b \in [0, S)$ and $t_b \in [0, T_b)$.

2.2.4 Optimization

In the ~/argos3-AutoMoDe/optimization/example folder, one can find the necessary elements to optimize a PFSM controller for the aggregation mission pre-

sented in Section 5. The optimization algorithm used is irace (López-Ibánez et al., 2011). The user should refer to the irace user guide for more information on how irace works[6].

One can also find a useful bash script that generates the grammar used by irace to build PFSM. This script takes three parameters: the maximal number of states, the maximal number of transitions and the file name in which the grammar will be written.

In order to start the optimization process, the user can simply execute

```
$ cd ~/argos3-AutoMoDe/optimization/example
$ $IRACE_HOME/bin/irace
```

[6]See http://iridia.ulb.ac.be/irace/

3 The ARGoS3-NEAT Package

ARGoS3-NEAT is an implementation of NEAT for ARGoS3 simulator and e-puck robotic platform. This package allows to optimize neural networks with the NEAT evolutionary algorithm and to test it in simulation and on real robots. NEAT (Stanley and Miikkulainen, 2002) is a method of neuroevolution in which an evolutionary algorithm optimizes a neural network. In NEAT, differently from other methods, the topology of this neural network is not fixed.

The remaining of this section is organized as follows: Section 3.1 is dedicated to instruct the user on how to install ARGoS3-NEAT. Section 3.2 illustrates how ARGoS3-NEAT works and Section 3.3 is dedicated to the configuration and output files of ARGoS3-NEAT.

3.1 Installation

In this section we describe how the user has to proceed in order to download and compile the ARGoS3-NEAT package. The user has to make sure the requirements described in Section 3.1.1 are met before following the compiling instructions of Sections 3.1.3 and 3.1.4.

3.1.1 Requirements

Currently we support only UNIX-like operating systems. Windows in currently not supported. The following procedures applies to GNU/Linux but are similar in all UNIX-like systems.

The lastest version of the ARGoS3 simulator should be installed and compiled. See, https://github.com/ilpincy/argos3 for details on installation. The specific plugin for the E-puck robots (ARGoS3-Epuck) is also required. See, https://github.com/lgarattoni/argos3-epuck for details on installation.

3.1.2 Download

```
$ git clone https://github.com/demiurge-project/ARGoS3-NEAT
    argos3-NEAT
```

For multiprocessing support for running parallel jobs, the user should also install openmpi :

```
$ sudo apt install openmpi-bin
```

3.1.3 Compiling for the Simulator

```
$ cd argos3-NEAT
$ mkdir build
$ cmake ..
$ make
```

3.1.4 Compiling for the Robots

```
$ cd argos3-NEAT
$ mkdir build
$ cd build
$ cmake -DCMAKE_TOOL_CHAIN=../src/cmake/TargetEPuck.cmake ..
$ make
```

3.2 Example

In this section, we illustrate how ARGoS3-NEAT works with the help of an example mission. The mission used as example is the one described in Section 5. The goal of the mission is to make robots aggregate on a black spot in a dodecagonal arena. We use 5 e-puck robots and an experiment duration of 60 seconds.

3.2.1 Running the Example

The user can train the neural network with the evolutionary process with NEAT and ARGoS3:

```
$ bin/train experiments/example/example_aggregation.xml params
    /evostickParams.ne startgen/evostickstartgenes
```

The first argument is the argos xml file that describes the experiment.
The second one is the configuration file that drives the evolutionary process of NEAT.
The third one is the genome that describes the topology of the neural network.

For running jobs in parallel, the user can change the previous command with the following :

```
$ bin/train experiments/example/example_aggregation.xml params
    /evostickParams.ne startgen/evostickstartgenes N bin/
    scheduler
```

where N is the number of parallel jobs.

After the end of the process (after few minutes), the user will find the champion genomes under the gen/ folder.
The files are named with the number of the generation (the last generation's name contains the word "last"), the number of the run and the "_champ" suffix for the champion.

Finally, in order to run a simulation with a specified genome (e.g. to visualize a champion genome) the user can launch the following command:

```
$ bin/evostick_launch -c experiments/example/
    example_aggregation_visu.xml -g gen/gen_last_1_champ
```

3.3 Configuration and Output Files

3.3.1 NEAT parameters file

The parameter file (.ne) contains the configuration for neuroevolution for NEAT. The different options are the following :

TRAITS:

- `trait_param_mut_prob`: probability to mutate a trait parameter.
- `trait_mutation_power`: power of mutation on a single trait parameter, that is, power about how much a trait can change/mutate.

- `linktrait_mut_sig`: Amount that `mutation_num` changes for a trait change inside a link.
- `nodetrait_mut_sig`: Amount a `mutation_num` changes on a link connecting a node that changed its trait.
- `weigh_mut_power`: power about how much a weight can change.
- `recur_prob`: probability that a link mutation which does not have to be recurrent will be made recurrent.

COMPATIBILITY - SAME SPECIE:

- `disjoint_coeff`: coefficient used to determine the compatibility between 2 genomes.
- `excess_coeff`: coefficient used to determine the compatibility between 2 genomes.
- `mutdiff_coeff`: coefficient used to determine the compatibility between 2 genomes.
- `compat_thresh`: threshold under which 2 genomes are considered to belong to the same species.
- `age_significance`: How much does age matter? If it's = 1 then young species will get no fitness boost.
- `survival_thresh`: only the top `survival_thresh*100\%` of each species is allowed to reproduce.

MUTATION:

- `mutate_only_prob`: probability of mutating without reproduction.
- `mutate_random_trait_prob`: probability to mutate a random trait.
- `mutate_link_trait_prob`: probability to change a link's trait.
- `mutate_node_trait_prob`: probability to change a node's trait.
- `mutate_link_weights_prob`: probability to mutate a link's weight.
- `mutate_toggle_enable_prob`: probability to toggle genes/connections on/off.
- `mutate_gene_reenable_prob`: probability to reenable a gene/connection/link between 2 nodes.
- `mutate_add_node_prob`: probability to add a node.
- `mutate_add_link_prob`: probability to add a link between two nodes.

MATING (type of mating):

- `interspecies_mate_rate`: probability of a mate being outside species.

- `mate_multipoint_prob`: probability that the mating between 2 genomes will be a multipoint crossover, that is, for every point in each Genome, where each Genome shares the innovation number, the Gene is chosen randomly from either parent. If one parent has an innovation absent in the other, the child will inherit the innovation. Interspecies mating leads to all genes being inherited. Otherwise, excess genes come from the most fit parent.
- `mate_multipoint_avg_prob`: probability that the mating is like the multipoint but instead of selecting one or the other genome when the innovation numbers match, it averages their weights.
- `mate_singlepoint_prob`[7]: probability that the mating is a single crossover point.
- `mate_only_prob`: probability of mating without mutation.
- `recur_only_prob`: probability of forcing selection of ONLY links that are naturally recurrent. Decide whether to make the link recurrent (It seems that this parameter is replacing the `recur_prob` parameter defined above).
- `pop_size`: size of the population.
- `dropoff_age`: age at which a species starts to be penalized if it is not making any progress, that is, it is no longer allowed to reproduce.
- `newlink_tries`: number of tries `mutate_add_link` will attempt to find an open link.
- `print_every`: print at every `print_every` generations, a population file containing the species/organisms at the current generation.
- `babies_stolen`: number of offsprings to steal from poorer species and to redistribute to better species. The more babies stolen, the more biased the search.
- `num_runs`: number of times to run the experiment.

NEW PARAMETERS (was not present in the original version):

- `num_gens`: number of generations (This parameter has been added from the original version).
- `num_runs_per_gen`: number of times to run the experiment before going to the next generation. It is useful when we want to compute the average score of a neural network.
- `num_runs_post_eval`: number of times to run the experiment to reevaluate the whole population. This is the post-evaluation, that is, it will be run after the whole evolutionary process.
- `weight_lower_bound`: lower bound for the weights.
- `weight_upper_bound`: upper bound for the weights.
- `elitism_percentage`: Percentage of the population that is copied unchanged to the next generation.

[7] $mate\_multipoint\_prob + mate\_multipoint\_avg\_prob + mate\_singlepoint\_prob = 1$

3.3.2 Genome Description

In evolutionary robotics the genome is the description of the structure of a neural network. In NEAT the genome is described in a genome file. A genome file is needed to run an experiment since the training algorithm uses it as a starting point for the evolution algorithm. The different organisms (individuals) created by the algorithm share the structure of the initial genome file.

Each genome file contains traits, nodes and genes and looks like this:

```
genomestart id
trait
...
node
...
gene
...
genomeend id
```

where *id* is a natural number representing the id of the genome.

TRAITS:
In the future, traits will allow the system to evolve highly complex neural models, much more sophisticated than the simple sigmoidal neurons currently used. The neurons will be able to evolve plasticity parameters (like real neurons in the brain) by pointing to a trait that describes the plasticity of the neuron.

```
trait traitId x y y y y y y y
```

- `traitId`: id of the trait, this natural number should be unique.
- `x`: todo
- `y`: todo

NODES:
list of nodes that are present in the neural network.

```
node id traitId nodeType geneticNodeType
```

- `id`: id of the node, this natural number should be unique
- `traitId`: id of the trait to use.
- `nodeType`: 0 if neuron, 1 if sensor.
- `geneticNodeType`: 0 if hidden, 1 if input, 2 if output, and 3 if bias.

GENES:
list of connections between nodes.

```
gene traitId connectInNode connectOutNode weight recur_flag
    innov# mutation# enableBit
```

- `traitId`: id of the trait to use.

- `connectInNode`: id of the inNode.
- `connectOutNode`: id of the outNode.
- `weight`: weight of the connection. Real number.
- `recur_flag`: flag for the recurrence. 0 no recurrence, 1 there is a recurrence.
- `innov#`: innovation number - historical marking. This natural number should be unique.
- `mutation#`: Redundant parameters, it is the same as the weight of the connection.
- `enableBit`: 1 if the connection is enabled, 0 if disabled.

3.3.3 Population Description

A population file contains all the organisms at a certain generation.
The population file starts with the species:

```
Species <<id>> : (Size <<size>>) (AF <<AF>>) (Age <<age>>)
```

- `id`: id of the species.
- `size`: size of the species.
- `AF`: average fitness of the species.
- `age`: age of the species.

Followed by the description of the organisms inside this species.

```
Organism <<id>> Fitness: <<fitness>> Error: <<error>>
```

- `id`: id of the organism inside the population (not the one inside the specie).
- `fitness`: the value of its fitness.
- `error`: how many times it got it wrong.

Each organism is then described by its genome (see section 3.3.2 for more information)

4 EvoStick

`EvoStick` is implemented as a specific instance of NEAT. The neural networks generated by `EvoStick` have a predefined and fixed topology. The neural networks are feed-forward and do not contain hidden neurons. To generate such neural networks with NEAT, we disabled the parameters that have an impact on the topology. More precisely, the following parameters are set to zero: `mutate_add_node_prob`, `mutate_add_link_prob`, `interspecies_mate_rate`, `mate_multipoint_prob`, `mate_multipoint_avg_prob`, and `mate_only_prob`. Furthermore, the following parameters are set to 1.0: `mutate_only_prob` and `mutate_link_weights_prob`.

5 Example: Aggregation mission

In this section, we describe the aggregation experiment used as example in the packages ARGoS3-AutoMoDe and ARGoS3-NEAT. The goal of this mission is to have as many robots as possible positioned on a single circular black spot placed on the ground, as depicted in Figure 3. The dodecagonal area in which the robots are evolving is delimited by walls of length equal to 66 centimeters. The black spot has a 30 centimeter radius and is placed on the center of the arena. The epucks can detect the color of the ground below them (i.e. gray or black) thanks to their ground sensor.

The objective function of this mission is the fraction of the robots situated on the black spot and has to be maximized: $F_{obj} = \frac{N_b}{N}$, where N_b represents the number of robots on the black spot and N represents the total number of robots. This function is evaluated at the end of the experiment.

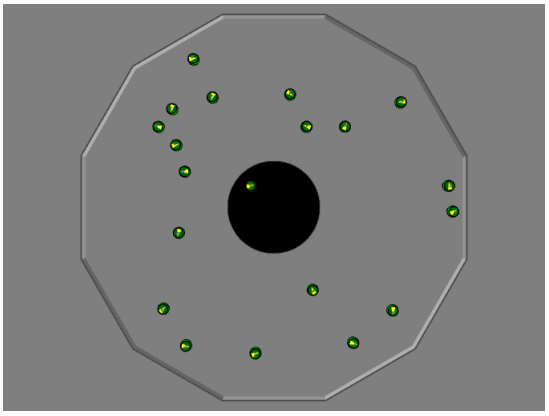

Figure 3: Screenshot of ARGoS3 when simulating 20 epucks on the aggregation mission that we use as illustrative example.

6 Concluding remarks

This document is not the final version and it will be updated along with the technical development of the packages ARGoS3-AutoMoDe and ARGoS3-NEAT.

References

Francesca, G., Brambilla, M., Brutschy, A., Garattoni, L., Miletitch, R., Podevijn, G., Reina, A., Soleymani, T., Salvaro, M., Pinciroli, C., Mascia, F., Trianni, V., and Birattari, M. (2015). AutoMoDe-Chocolate: automatic design of control software for robot swarms. *Swarm Intelligence*, 9(2/3):125–152.

Francesca, G., Brambilla, M., Brutschy, A., Trianni, V., and Birattari, M. (2014). AutoMoDe: a novel approach to the automatic design of control software for robot swarms. *Swarm Intelligence*, 8(2):89–112.

Garattoni, L., Francesca, G., Brutschy, A., Pinciroli, C., and Birattari, M.

(2015). Software infrastructure for e-puck (and TAM). Technical Report 2015-004, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium.

López-Ibánez, M., Dubois-Lacoste, J., Stützle, T., and Birattari, M. (2011). The irace package, iterated race for automatic algorithm configuration. Technical report, Citeseer.

Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., Zufferey, J.-C., Floreano, D., and Martinoli, A. (2009). The e-puck, a robot designed for education in engineering. In *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, pages 59–65, Castelo Branco, Portugal. IPCB: Instituto Politécnico de Castelo Branco.

Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Di Caro, G., Ducatelle, F., Birattari, M., Gambardella, L. M., and Dorigo, M. (2012). ARGoS: A modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intelligence*, 6(4):271–295.

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