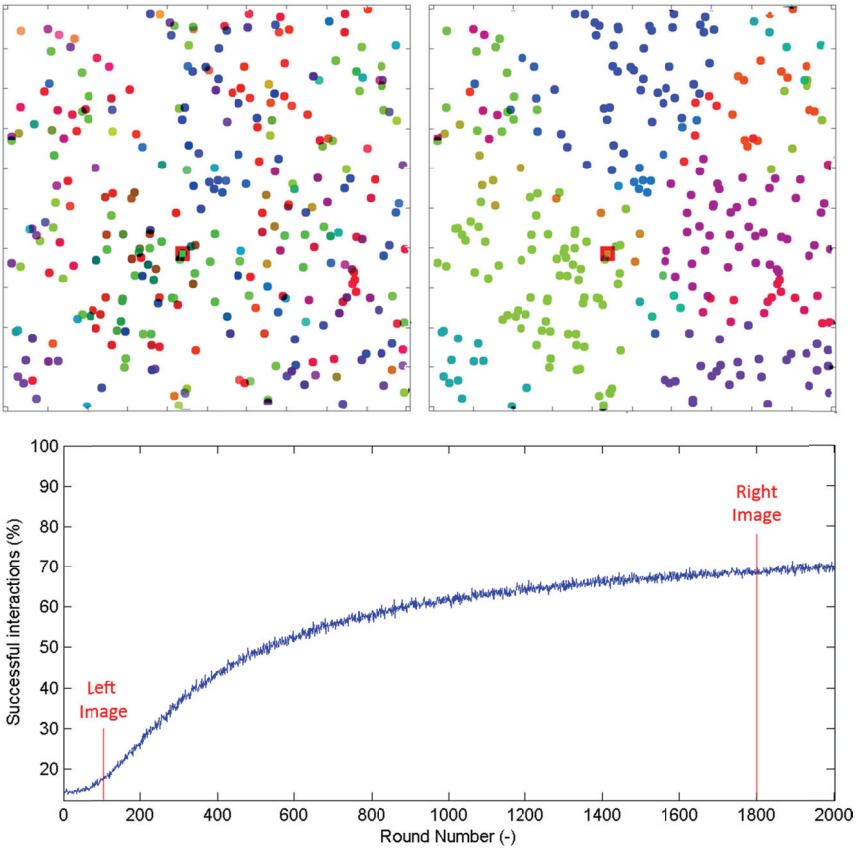
Project Report FS2011

Agent-based Modeling of Language Evolution

Modeling and Simulating Social Systems with MATLAB



Authors:

Manu Gomez, mgomez@ethz.ch, 04-907-929

Silas Menberg, menbergs@ethz.ch, 02-915-825

Supervised by:

Karsten Donnay and Stefano Balietti

SOMS, Chair of Sociology, ETHz

28.05.2011

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Abstract

In this project we dealt with the emergence of language. The main goal was to understand how a population adopts a dictionary, without having a central authority that decides the meaning of every word and how it develops with time and space.

For that purpose, we implemented an agent-based model based on the approach proposed by Gostoli (2008). Gostoli assumes that language evolved as means of communicating information between individuals. Starting point is a population composed of a number of individuals, which possess a set of noise without any meaning. Through interaction the individuals acquire a socially shared convention which connects a noise to a meaning. The model is based on the theoretical framework of evolutionary game theory. The spatial dependence of the language evolution is considered by implementing a 2D grid.

We succeeded to produce results similar to Gostoli (2008), which show that in any case a socially shared convention that connects noise to a meaning emerges with time. It was also possible to show that several different languages can coexist in a population whereby individuals with the same language are locally grouped together.

Table of Content

1 Introduction and Motivations 1

2 Description of the Model 1

3 Model Implementation 2

3.1 Row of processes 2

3.2 Temporal dimension 3

3.3 Agent's dictionary 3

3.4 Interaction 4

3.5 Agent's memory and update of agent's dictionary 6

3.6 Spatial dimension 7

3.7 Color coding 7

4 Methods 8

4.1 Scenario 1a 8

4.2 Scenario 1b 9

4.3 Scenario 2 9

4.4 Scenario 3 9

5 Simulation Results and Discussion 10

5.1 Scenario 1a 10

5.2 Scenario 1b 10

5.3 Scenario 2 11

5.4 Scenario 3 15

6 Conclusion and Outlook 17

7 References ..... A

8 Appendix ..... B

8.1 Simulation result scenario 1 ..... B

8.2 Simulation results scenario2 ..... B

8.3 Simulation results for scenario 2 with communication barrier C

8.4 MATLAB source codes ..... D

8.4.1 Initialisation ..... D

8.4.2 Iteration ..... F

8.4.3 Output ..... J

1 Introduction and Motivations

One of the most important skills that characterize human beings is the ability to communicate among each other through language. Exchanging information on simple and especially complex matter requires a communication system which is well known among all the participants. The human language is based on words whose meaning is a function of both the meaning of the words itself as well as the way they are put together. Different population adopted a different lexicon and different rules of putting the words together. The goal of language emergence research is to increase the understanding of the process through which the language has emerged. The answers of the following main questions are of interest:

1) How does a population adopt a lexicon, without having a central authority that decides the meaning of every word? (Gostoli, 2008)
2) Who decides what the meaning of every word is?
3) How does it emerge over time and space? (Grim, Denis, & Kokalis, 2004)
4) Will it be possible to create a steady state where multiple lexica can coexist?

The approach introduced by Umberto Gostoli (2008) will be used and extended by a spatial component described by Grim et al (2004) in order to address the above mentioned questions. Gostoli assumes that language evolved as means of communicating information between individuals. A language is based on a socially shared convention that associates a noise to a specific meaning (object) and vice versa. Starting point is a population composed of a number of agents which possess a set of noises without any meaning (corresponding object). Communicating information between two agents means that one agent is the speaker and will send a noise to describe an object and the other agent is the listener and will try to assign an object to the receiving noise.

2 Description of the Model

The model presented in this report is based on the approach proposed by Gostoli (2008). In a first step it was attempted to produce results similar to those presented in his work. In a second step the model was extended by a spatial component in order to investigate the emergence of a lexicon over space and a possible coexistence of multiple languages.

The adopted model is an agent based model which has to be described at two different levels: At the first level the agent's characteristics and skills are described, while at the second level the characteristics and rules of interaction between the agents are described (Gostoli, 2008).

The interactive level of the model is based on the evolutionary game theory. The model starts from a population composed of a number of individuals or agents, which come together to repeatedly play a game. In each period (matchround) every agent is matched with every other agent in order to play the game. In this particular case it is a signaling game. One agent is the speaker and will send a noise, chosen from a set of possible noises, in order to describe an observed object. The other agent is the listener and will try to assign an object to the receiving noise. The listener can't observe the object the speaker refers to. From the interaction each agent gets a payoff dependent on the object observed, the noise chosen by the speaker and the object understood by the listener. The interaction is successful when the object observed and communicated by the speaker matches the object understood by the listener. The agents are treated both as speakers as well as listeners.

The problem an agent faces in each interaction is to forecast, in case it plays the role of the speaker, or guess, in case it plays the role of the listener, its partner's action in order to optimize its own payoff. The agent's forecast, respectively guess, depends on the information it could gather from the previous games and on the decision process through which the agent computes this information. Every agent has its own matrix of the size of number of object times number of noises. This matrix summarizes the information about the actions of all game partners in the previous games and hence giving the agent a probability distribution over its partner's action. Based on this information the agent chooses the noise to communicate the observed object, respectively chooses the object to interpret the receiving noise, in the next interaction with the intent to increase its own payoff.

This game is played over and over again until the rate of successful interaction per matchround does not increase anymore or a predefined number of matchround is reached.

The spatial dependence of the language evolution is considered by implementing a 2D grid. Every agent is assigned a fixed location on this plane and an interaction radius which spatially restricts the agent's action range. This allows simulating a population where only the closest neighbors interact with each other.

3 Model Implementation

3.1 Row of processes

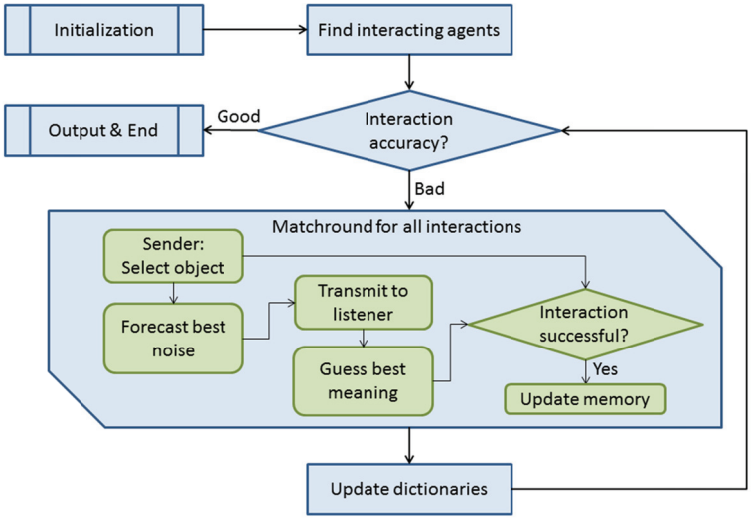


Figure 1: Flowchart of the implemented MATLAB-Model

As shown in Figure 1 the core of the model is where the matchrounds are executed. Within one matchround the sender selects a randomized object which is transformed to a matching noise using the agent's matrix (dictionary). The noise is transmitted to the listening agent where it's been re-translated by the listeners dictionary. The understood object is compared to the initial object. If they match the object-noise-relation is stored in the memory of both agents. After all interactions were done the memories of the agents are checked for the strongest relation, which is finally stored in the agents' dictionary.

3.2 Temporal dimension

The temporal component of the model is simulated by an iterative loop where all agents interact and update their matrices. The more loops are executed the more time has passed. The main problem is to determine the time to stop the simulation where the steady state is reached. An algorithm was needed to interrupt the iterative process.

There were different approaches:

1) The first approach was to check for a certain threshold. Assuming the rate of successful interactions will depend on the action radius, the population density and on a random component the right threshold is not predictable.
2) Numerical functions with a steady state can be checked by a decreasing delta between two steps. As soon as this delta drops below a small value (i.e. $\Delta y < 10^{-6}$) the process can be interrupted. In our case the accuracy is heavily influenced by a stochastic component and is highly fluctuating, so the delta-approach is not applicable.
3) To solve the problem of approach 2 a floating average (over 10 / 30 elements) is computed. The steady state is reached as soon as $\frac{1}{20}\sum_{n=10}^{30} f(n) - \frac{1}{10}\sum_{n=1}^{10} f(n) < 10^{-6}$. This approach leads to some unpredictable problems in certain applications (mainly at the beginning of the iterative process).
4) The last and simplest approach is ignoring the accuracy of the interactions and forces the iterative process to run a fixed amount of cycles. As a rule of thumbs this value is set to $MaxRounds > 1000$.

3.3 Agent's dictionary

Every agent has a probability matrix *P* which stores the agent's experience. This matrix is also called the agent's dictionary. If there's a number of objects *O* and a number of noises *N*, an *O times N* matrix is generated. In the implemented communication game the number of possible noises *N* corresponds to the number of objects *O*. For the simulations presented below in section 5 this parameter value has been set equal to 7. The objects and noises are represented by integers from 1 to 7. The resulting *7 × 7* probability matrix *P* contains the entries p_{ij}, denoting the probability that for the agent in the role of the speaker, object *i* is associated with the noise *j*. For the agent in the role of the listener matrix *P* contains the entries p_{ji} which denote the probability that the noise *j* is associated with object *i*. Matrix 1 shows a general probability matrix *P*. For the sake of convenience a *3 × 3* instead of a *7 × 7* matrix is used to illustrate the main components of the implemented model.

$$\begin{array}{c} \\ \\ object\ i \end{array} \begin{array}{c} \begin{array}{ccc} & noise\ j & \\ 1 & 2 & 3 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \end{array} \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix} \end{array}$$

Matrix 1: General probability matrix

At the beginning of the simulation the probability matrix of each agent is uniform distributed, as there is no socially shared convention that associates an object to a noise. This means that all the entries p_{ij}, have the same start value, in this case the value has been set equal to one. The speaker pseudo-randomly chooses a noise by drawing a number from 1 to n in order to communicate an

object. At the same time the listener can only make a random guess. As a consequence, on average only every N^{th} conversation will be successful in the beginning.

$$\begin{array}{c} \\ object\ i \end{array} \begin{array}{c} \begin{array}{ccc} noise\ j & & \\ 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \end{array} \begin{array}{c} \\ \begin{array}{c} 1 \\ 2 \\ 3 \end{array} \end{array}$$

Matrix 2: Initial state of an agent's dictionary

The agent's dictionary gets periodically updated with the information gathered from the interactions with the other agents. Each agent memorizes the combination of object and noise which led to successful conversation and updates its probability matrix by increasing the value of the corresponding p_{ij}. This process will be described in detail in the next sections. With increasing number of interactions specific combinations of object and noise (respectively combinations of noise and object) will exhibit a higher p_{ij} value and therefor a higher probability that the object i is associated with the noise j.

$$\begin{array}{c} \\ object\ i \end{array} \begin{array}{c} \begin{array}{ccc} noise\ j & & \\ 1 & 2 & 3 \end{array} \\ \begin{bmatrix} \mathbf{324} & 43 & 23 \\ 54 & 21 & \mathbf{422} \\ 89 & \mathbf{511} & 33 \end{bmatrix} \end{array} \begin{array}{c} \\ \begin{array}{c} 1 \\ 2 \\ 3 \end{array} \end{array}$$

Matrix 3: Agent's dictionary after several conversations

3.4 Interaction

In each matchround every agent is matched twice with every other agent in order to exchange information, once in the role of the speaker and once in the role of the listener.

At the beginning of a matchround every agent pseudo-randomly selects an object $O_{selected}$, which the agent wants to communicate as speaker, by drawing a number from 1 to O. In order to send the information it has to produce a noise. Therefore every agent creates a matchround specific speaker dictionary *Object2Noise* which relates every object to a certain noise. This is done by pseudo-randomly drawing a noise for each object weighted by the agent's probability matrix.

First the cumulative distribution for every object is calculated. This is illustrated with Matrix 4.

$$\begin{array}{cc} Agents\ Dict. & CumSum, horzontal \\ \begin{bmatrix} 324 & 43 & 23 \\ 54 & 21 & 422 \\ 89 & 511 & 33 \end{bmatrix} & \rightarrow \begin{bmatrix} 324 & 367 & 390 \\ 54 & 75 & 497 \\ 89 & 600 & 633 \end{bmatrix} \end{array}$$

Matrix 4: Probability matrix (left) and the corresponding matrix with the cumulative values for each object (right)

In a second step the agent pseudo-randomly draws a number x_i between zero and the maximum cumulative value for each object i.

$$x_1 = 215 \qquad \text{where } x_1 \text{ is } 0 \leq x_1 < 390$$
$$x_2 = 342 \qquad \text{where } x_2 \text{ is } 0 \leq x_2 < 497$$
$$x_3 = 487 \qquad \text{where } x_3 \text{ is } 0 \leq x_3 < 633$$

Finally for each object the noise has to be found where the value of the cumulative exceeds the random value as illustrated in Matrix 5.

$$\begin{bmatrix} 324 > 215? & 367 > 235? & 390 > 235? \\ 54 > 342? & 75 > 342? & 497 > 342? \\ 89 > 487? & 600 > 487? & 633 > 487? \end{bmatrix}$$

Matrix 5: Comparison of the random value to CumSum

Matrix 5 leads to the corresponding Boolean matrix where only the first "1" per row is used for the final Object-to-Noise-Matrix.

$$\underset{Boolean\ Matrix}{\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}} \rightarrow first\ only \rightarrow \underset{Object2Noise}{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}}$$

Matrix 6: Generation of the *Object2Noise*-Matrix

As a result the agent gets a distinct speaker dictionary which assigns object 1 to the noise 1, object 2 to noise 3 and object 3 to noise 2. This dictionary will be used by the agent to communicate the $O_{selected}$ to all its dialog partners.

The agent in the role of the listener, who receives the noise, has to transform the noise into an object $O_{understood}$. For this transformation every agent needs to create a matchround specific listener dictionary which relates every noise to a certain object. The process is analogous to the one described above for creating the speaker dictionary. But this time the cumulative distribution for each noise is calculated vertically.

$$\underset{Agents\ Dict.}{\begin{bmatrix} 324 & 43 & 23 \\ 54 & 21 & 422 \\ 89 & 511 & 33 \end{bmatrix}} \rightarrow \underset{CumSum,\ vertical}{\begin{bmatrix} 324 & 43 & 23 \\ 396 & 64 & 445 \\ 485 & 575 & 478 \end{bmatrix}}$$

Matrix 7: Probability matrix (left) and the corresponding matrix with the cumulative values for each noise (right)

Analogous to the steps above, three pseudo-random values (one per column) will be drawn and compared to the CumSum-value of each cell. A further correction of the Boolean matrix leads to the *Noise2Object*-Matrix, which is used as the listener dictionary.

$$\underset{random\ pick}{\begin{matrix} x_1 = 145 \\ x_2 = 452 \\ x_3 = 254 \end{matrix}} \qquad \underset{Noise2Object}{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}}$$

Matrix 8: Generation of the *Noise2Object*-Matrix

At the end of a matchround, after every agent interacted twice with every other agent, the success of all conversations has to be evaluated. A conversation is successful if the $O_{selected}$ chosen by the speaker is equal to $O_{understood}$ by the listener. Every agent memorizes the conversations and the object-noise combinations (respectively noise-object combinations) which achieved success and updates its probability matrix accordingly. This process is described in the next section.

3.5 Agent's memory and update of agent's dictionary

Besides the probability matrix every agent has an additional memory matrix of the same size as the probability matrix. The memory included and described here is maybe the most basic type of a memory possible. All the memory does is remembering which object-noise combination (respectively noise-object combination) achieved success in how many conversations within one matchround. At the beginning of every matchround the memory matrices get erased and start from zero. This is done for two reasons: Firstly to "forget" the influence of the previous matchround where the number of successful conversations on the average is lower than in the next one and secondary to enhance only the most successful dictionary and neglecting minor matches.

For every successful conversation the agent increases its memory matrix at the corresponding position [object/noise] by a value of one. Matrix 9 shows a possible memory matrix after one matchround.

$$\begin{bmatrix} 10 & 0 & 10 \\ 0 & 15 & 13 \\ 15 & 28 & 0 \end{bmatrix}$$

Matrix 9: Example of a memory matrix after a matchround

In order to decide at which positions p_{ij} of the probability matrix P an update should take place, the maxima of the memory matrix are identified. This procedure is illustrated by Matrix 10.

$$\begin{array}{lcc} & \textit{memory matrix} & \textit{update matrix} \\ i = 1: & \begin{bmatrix} 10 & 0 & 10 \\ 0 & 15 & 13 \\ 15 & \mathbf{28} & 0 \end{bmatrix} \rightarrow & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \\ i = 2: & \begin{bmatrix} 10 & \cancel{0} & 10 \\ 0 & \cancel{15} & \mathbf{13} \\ \cancel{15} & \cancel{28} & \cancel{0} \end{bmatrix} \rightarrow & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \\ i = 3: & \begin{bmatrix} \mathbf{10} & \cancel{0} & \cancel{10} \\ \cancel{0} & \cancel{15} & \cancel{13} \\ \cancel{15} & \cancel{28} & \cancel{0} \end{bmatrix} \rightarrow & \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \end{array}$$

Matrix 10: Generating an update matrix by looking for an unique maxima

After every matchround the probability matrix P gets updated according to memory matrix as follows:

$$Agent's\ Dict\ (t) + update\ matrix(t)\ =\ Agent's\ Dict\ (t+1)$$

$$\begin{bmatrix} 324 & 43 & 23 \\ 54 & 21 & 422 \\ 89 & 511 & 33 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 325 & 43 & 23 \\ 54 & 21 & 423 \\ 89 & 512 & 33 \end{bmatrix}$$

Matrix 11: Merge the update matrix to the agent's dictionaries

3.6 Spatial dimension

In order to simulate the language evolution in space the agents are put on a 2D grid. For the simulation presented below the grid has a shape of a square with a size of 100×100 units. Each agent is assigned an x- and y-coordinate by pseudo-randomly drawing an number between zero and 100.

Additionally a model parameter, called *agent_radius*, is defined which spatially restricts the operating range of every agent. An agent interacts only with agents, which are located within a distance smaller than the *agent_radius*, if the distance between two agents is bigger than *agent_radius* they do not interact.

3.7 Color coding

To visualize the agents' dictionaries a color coding process is implemented, where the highest probabilities of the dictionary are converted to values c which is further transformed to a unique color.

$$Probability\ matrix \qquad Maximum, horizontal$$

$$\begin{bmatrix} 324 & 43 & 23 \\ 54 & 21 & 422 \\ 89 & 511 & 33 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Matrix 12: Finding the highest probabilities of each row

$$Maximum \quad Transformation \qquad\qquad Synthetic\ matrix\ M$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} .* \begin{bmatrix} 100 & 10 & 1 \\ 200 & 20 & 2 \\ 300 & 30 & 3 \end{bmatrix} = M = \begin{bmatrix} 100 & 0 & 0 \\ 0 & 0 & 2 \\ 0 & 30 & 0 \end{bmatrix}$$

Matrix 13: Generation of the synthetic matrix M

$$sum(M) = 132 \ \ and \ \max\_M = 333$$

Where max_M is the highest possible value of sum(M).

$$c(M) = \frac{sum(M)}{\max\_M} \cdot 2\pi$$

In the example above the base of 10 is used. In the model itself, a base of O (amount of objects) is used instead.

$$color(c) = [0.5 + 0.5 \cdot \sin(c + s)]^{sat}$$

$$\text{Where } s = \begin{cases} 0/3 * \pi & red \\ 2/3 * \pi & green \\ 4/3 * \pi & blue \end{cases} \text{ and sat=1.3}$$

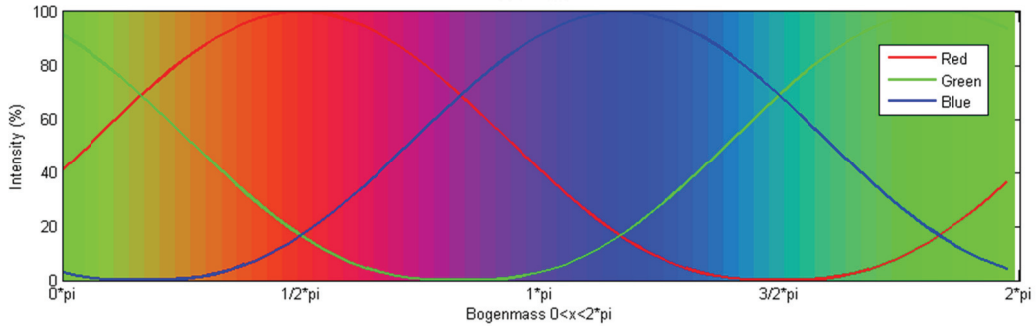


Figure 2: Rainbow-palette (in background) as a combination of sinusoidal functions of red, green and blue.

4 Methods

Different scenarios with different parameter values are analyzed. The following section lists the different parameter sets.

The aim of the simulations is to show if a population can develop a socially shared convention that makes the agents chose the same noise for the same objects and vice versa. In this case the all interactions in one matchround would be successful. Therefore the rate of successful interactions between the agents is of interest. In order to examine the evolution, the rate of successful interactions depending on the matchround is computed. For every scenario 100 runs are executed. The results are statistically analyzed by computing the mean and standard deviation.

The spatial development is examined by visually comparing the matchround specific maps of the agents' dictionary. The process of visualizing the agents' dictionaries is explained in section 3.7.

4.1 Scenario 1a

To compare the model to Gostolli's results[1], every agent has to interact with every other agent. This is done by choosing an *agent_radius* bigger than the diagonal of the 2D grid:

$$agent\_radius > \sqrt{Width^2 + Height^2}$$

Table 1: Parameters of scenario 1a

Parameter	Population	Width	Height	Action Radius	Objects	MaxRounds	SeedValue
Value	300	100	100	200	7	1400	8713

[1] See (Gostoli, 2008)

4.2 Scenario 1b

To show the influence of the world population the first scenario is modified. In the scenario 1b the world population is set to 32. This value will lead to the same amount of interactions per agents as seen in scenario 2.

The result of scenario 1b is compared to 1a and 2.

Table 2: Parameters of scenario 1b

Parameter	Population	Width	Height	Action Radius	Objects	MaxRounds	SeedValue
Value	32	100	100	200	7	1400	8713

4.3 Scenario 2

In a second scenario the *agent_radius* of the world population is decreased. As set to 20% of the grid size each agent talks more or less to its neighbors. All interactions within the interaction radius of an agent get the same weight of influence.

It is expected that a steady state can be reach where multiple lexica coexist. The rate of successful interactions might be lower compared to scenario 1.

Table 3: Parameters of scenario 2

Parameter	Population	Width	Height	Action Radius	Objects	MaxRounds	SeedValue
Value	300	100	100	20	7 (3)	1400	8713 (4732)

4.4 Scenario 3

To analyze the influence of the action radius (or population density) a series of runs is executed where the *agent_radius* is steadily increased.

The accuracy of the dictionaries is expected to increase with growing action radius.

Table 4: Parameters of scenario 3

Parameter	Population	Width	Height	Action Radius	Objects	MaxRounds	SeedValue
Value	300	100	100	10:5:80	7	1200	8713

5 Simulation Results and Discussion

5.1 Scenario 1a

Figure 3 shows the mean rate of successful interactions and the standard deviation for the 100 realizations depending on the matchround. The parameter values are summarized in the box on the right side of the figure. The graph is slightly different from Gostoli's result. The amount of successful interaction increases faster and stagnates at a lower level compared to Gostoli. This deviation can be explained with the differences during the implementation. Gostoli used a different approach for the decision process through which the agent computes the information for choosing the action for the next interaction. But also the memory and dictionary update algorithm used here differs from the one implemented by Gostoli.

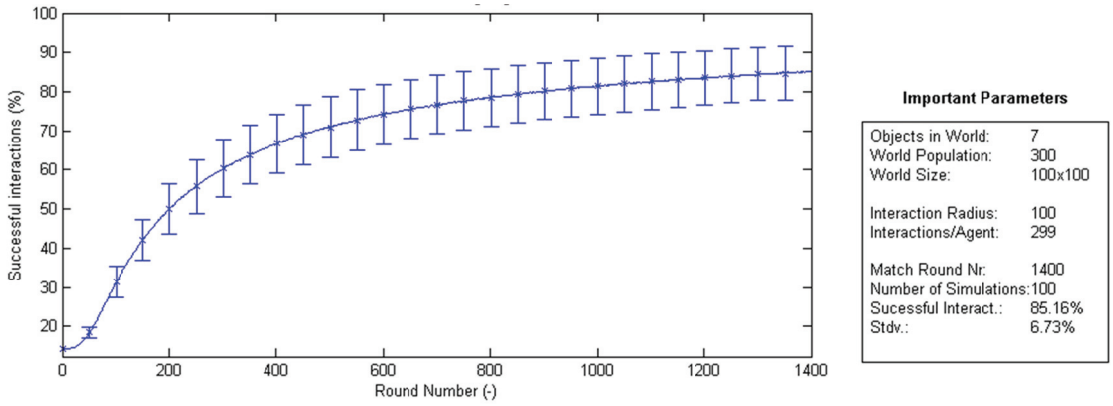


Figure 3: Evolution of language, mean and std.dev. of the interaction accuracy of 100 simulations (population: 300)

5.2 Scenario 1b

In this scenario the population size has been decreased by 90% compared to scenario 1.a. Figure 4 shows the mean rate of successful interactions and the standard deviation for the 100 realizations depending on the matchround. The language formation process is somewhat slower compared to scenario 1.a. More matchrounds are needed in order to reach the same level of success. This contradicts the results produced by Gostoli (2008) where the language formation process is faster in a smaller population.

A reason for this discrepancy might bethe implemented memory and the dictionary update algorithm.

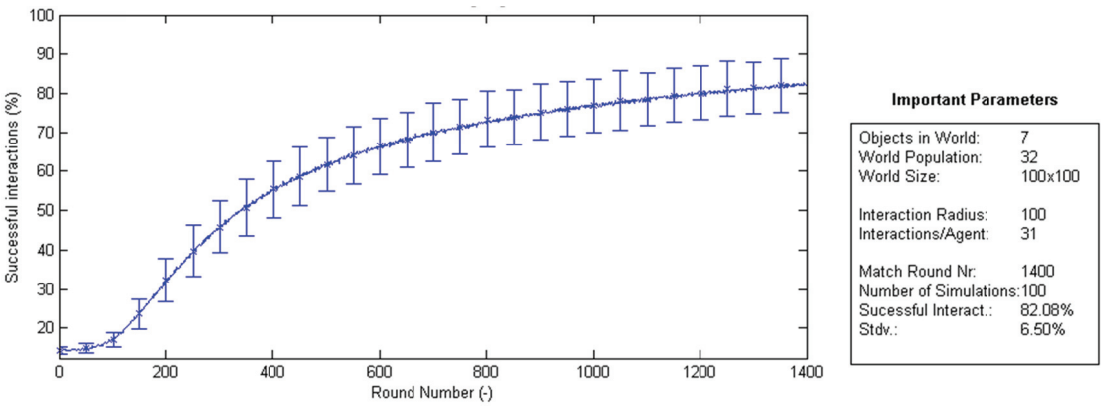


Figure 4: Evolution of language, mean and std.dev. of the interaction accuracy of 100 simulations (population: 32)

5.3 Scenario 2

For scenario 2 the *agent_radius* has been decreased by 90% compared to scenario 1.a. As a consequence of this, an agent does no longer interact with all other agents but only with the closest neighbors. Figure 5 reveals that the mean rate of successful interactions after 1400 matchrounds is clearly lower compared to scenario 1 where every agent interacts with every other agent of the population.

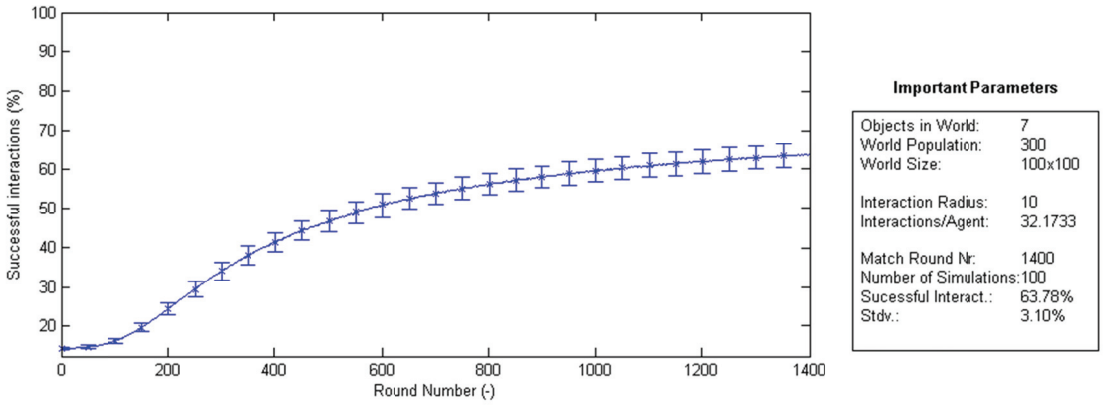


Figure 5: Evolution of language, mean and std.dev. of the interaction accuracy of 100 simulations (population 300, SeedValue 8718)

In order to examine the spatial development of the language formation the maps of the agents' dictionaries are computed. The visualization process for dictionaries with 7 objects described in section 3.7 leads to more than 800'000 colors. It is not possible to visually distinguish the different colors and therefore identify the different dictionaries of the agents. For that reason scenario 2 was run for 3 objects only. The results are shown in Figure 6 up to Figure 15. The agents' world illustrates the spatial development of the language formation. Every point reflects an agent and the color indicates its dictionary.

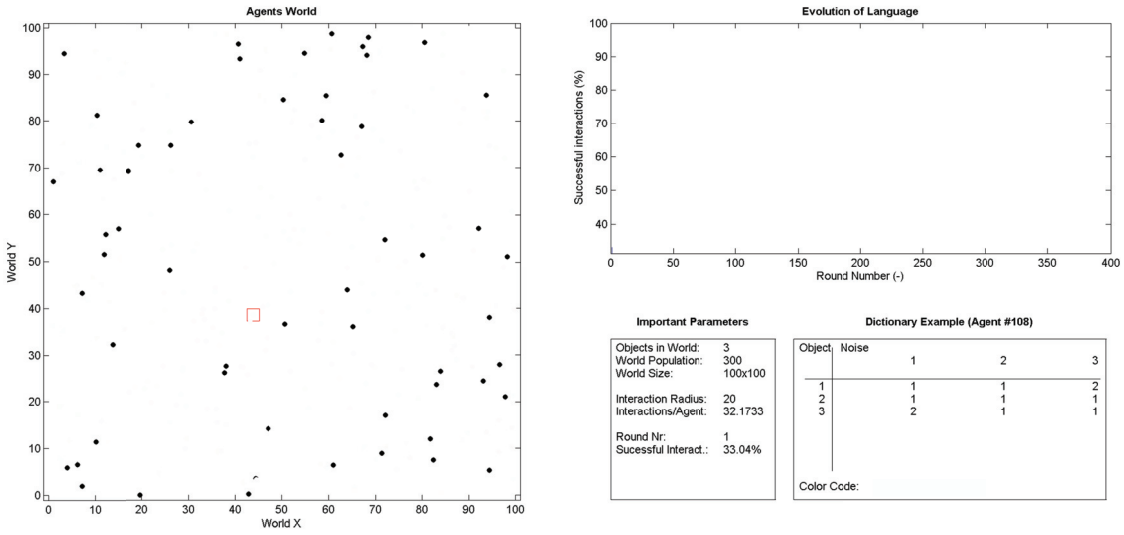


Figure 6: Matchround 1

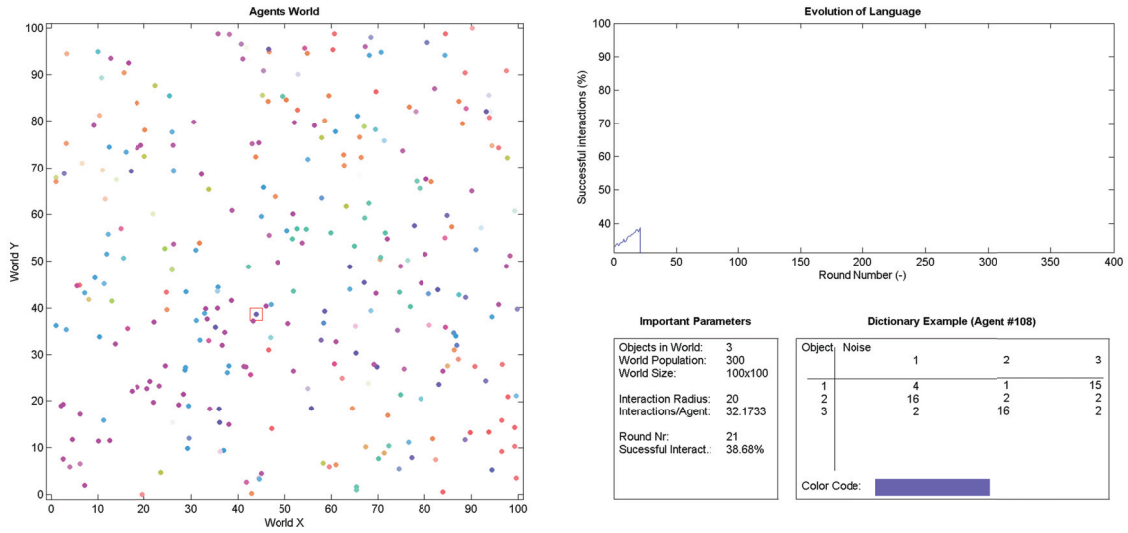


Figure 7: Matchround 21

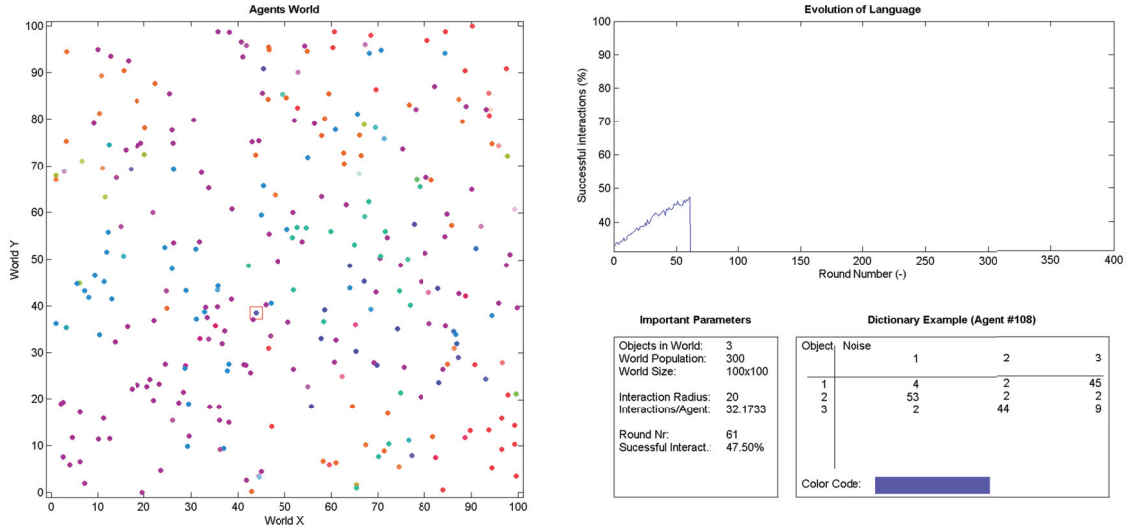


Figure 8: Matchround 61

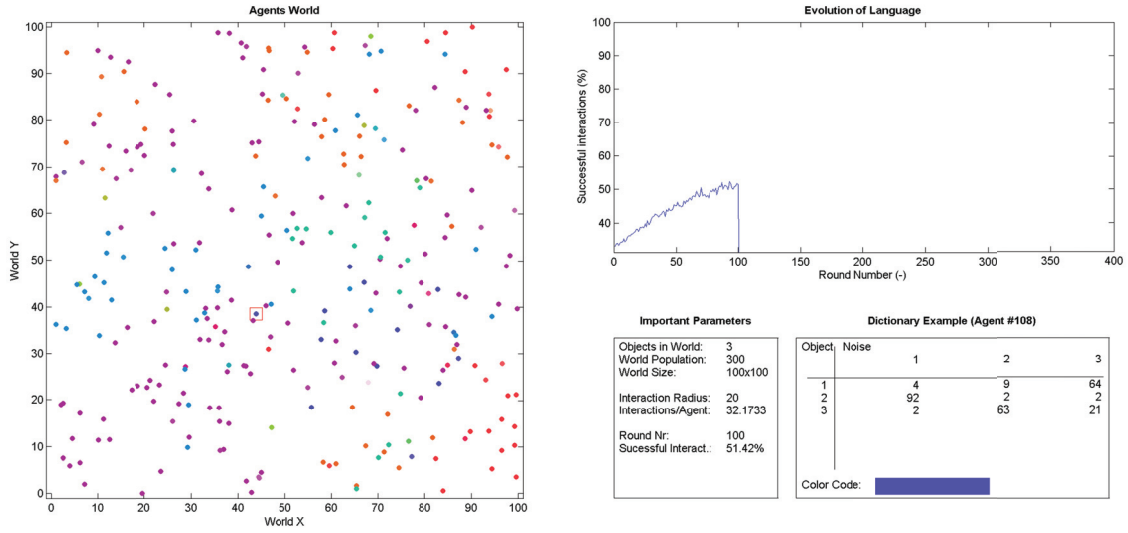


Figure 9: Matchround 100

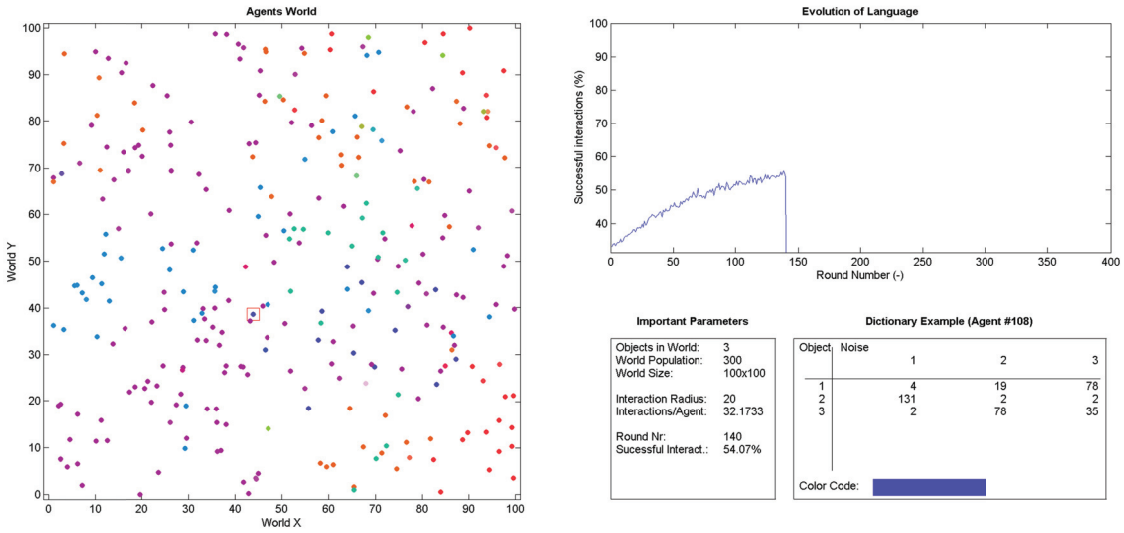


Figure 10: Matchround 140

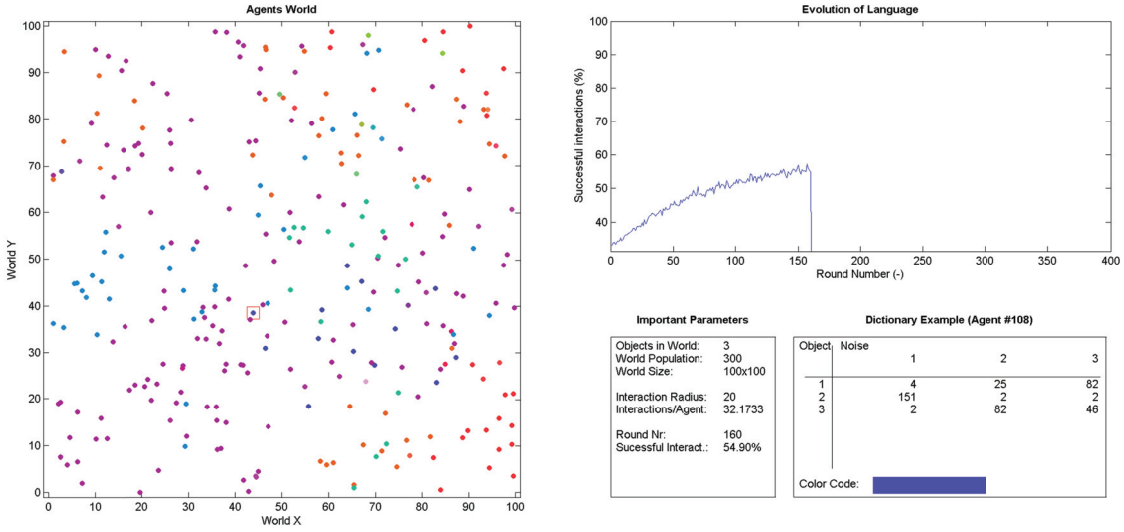


Figure 11: Matchround 160

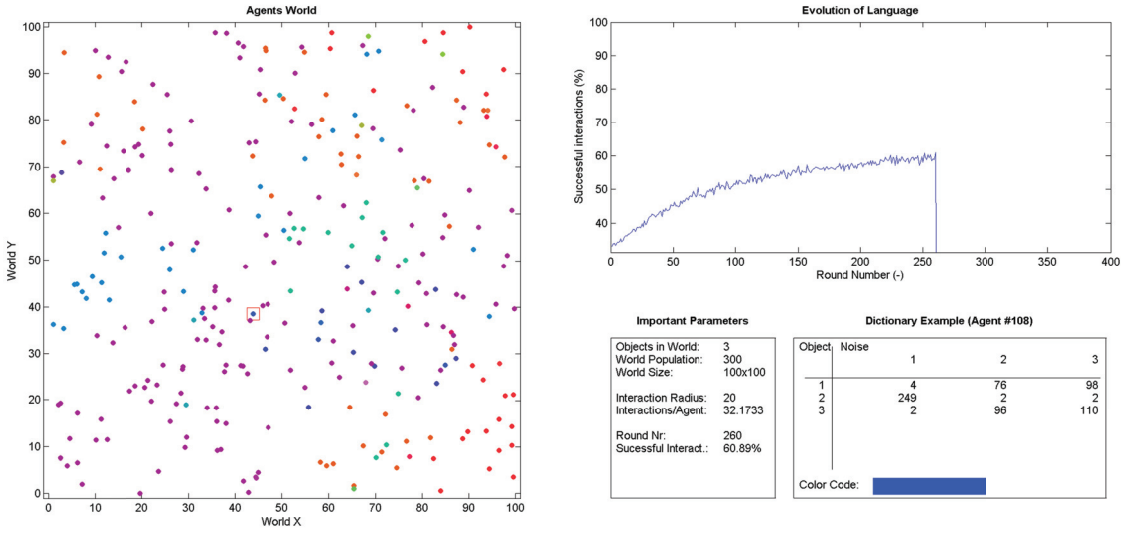


Figure 12: Matchround 260

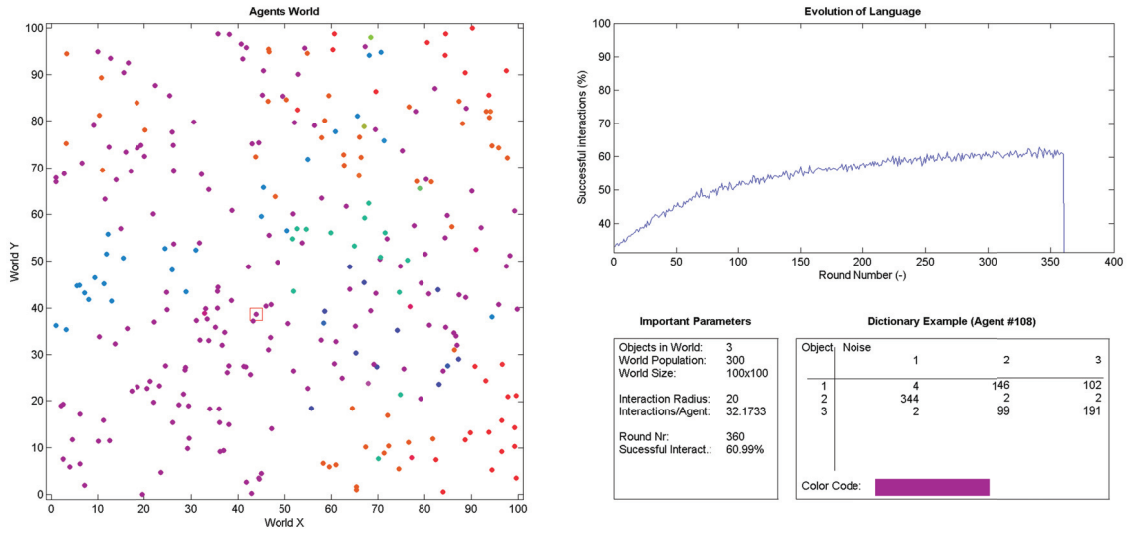


Figure 13: Matchround 360

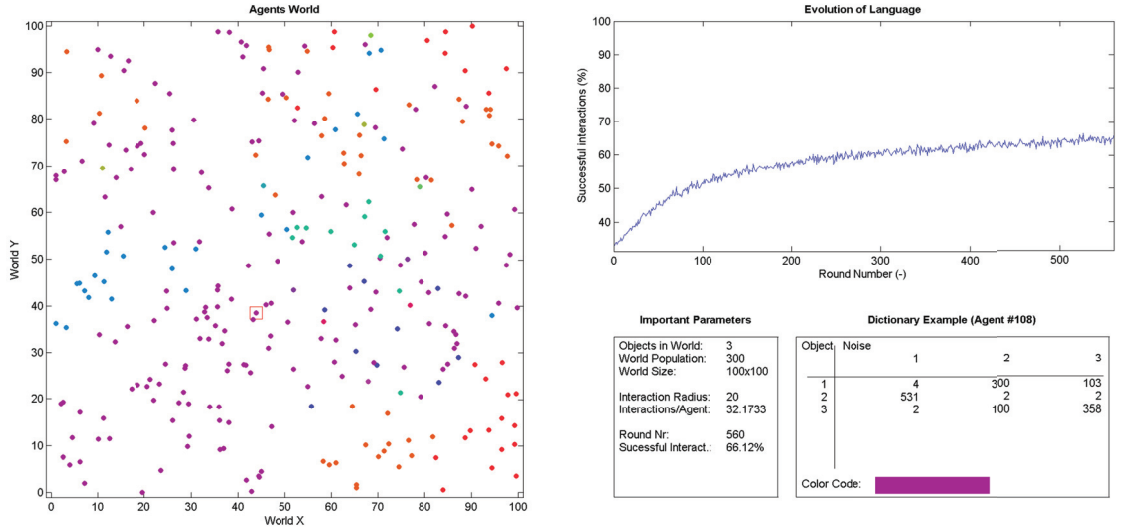


Figure 14: Matchround 560

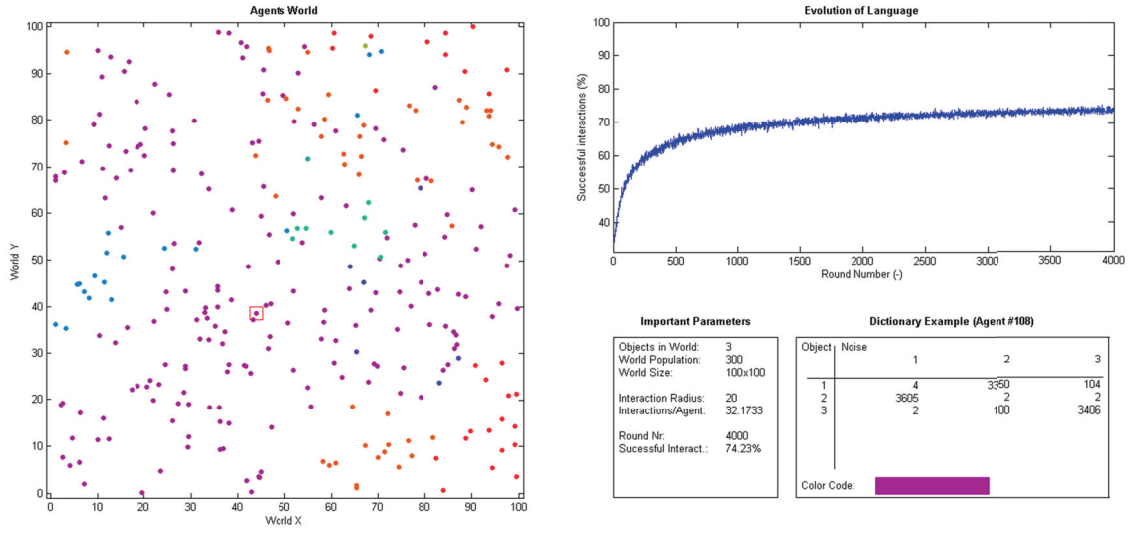


Figure 15: Matchround 4000 of realization 1

Figure 15 illustrates that different languages coexist. Agents with the same lexicon are locally grouped together. This is a clear difference to scenario 1 where only one language emergence and almost 100% interaction success is reached (see Figure 20 in Appendix 8.1). Figure 16 shows the agents' world after 4000 matchrounds and the evolution of the language for realization 2 (where SeedValue 4713 was used).

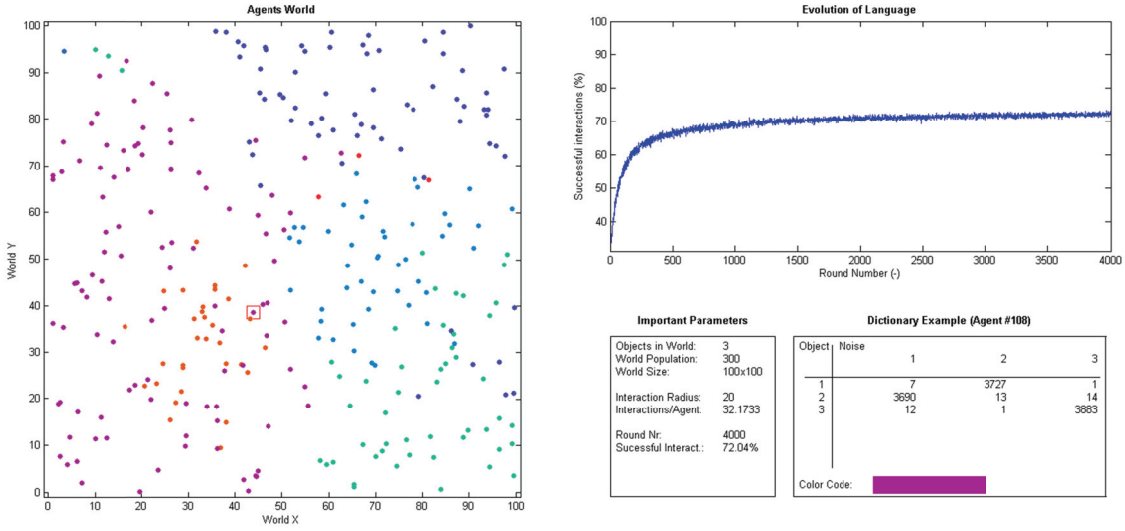
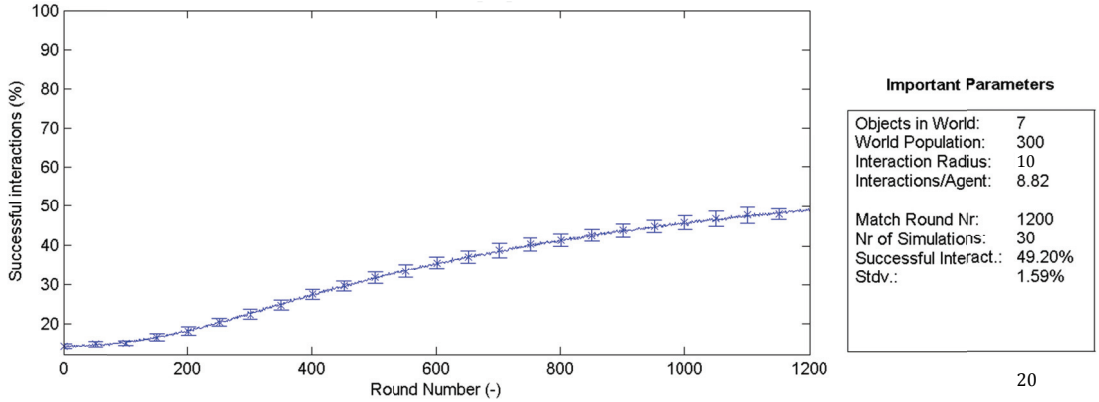


Figure 16: Matchround 4000 of realization 2

5.4 Scenario 3



20

Figure 17: Language Evolution, mean and std.dev. of the interaction accuracy with Agent_Radius=10

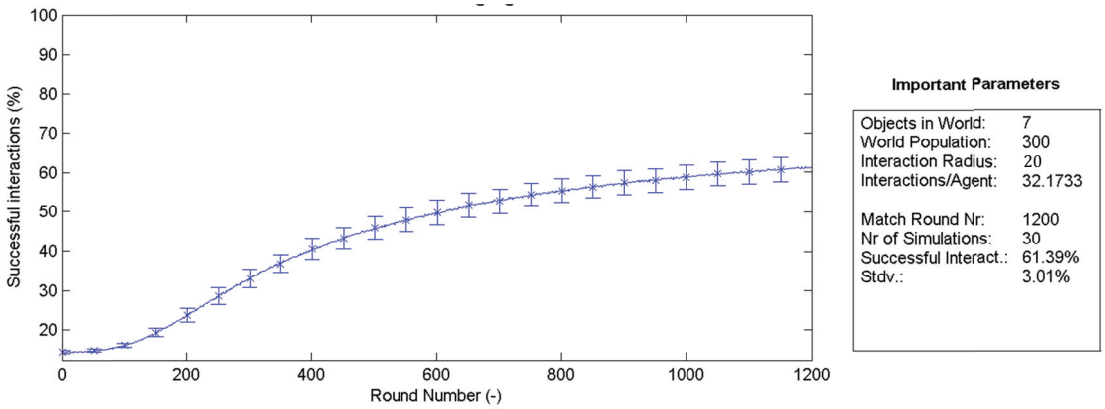


Figure 18: Language Evolution, mean and std.dev. of the interaction accuracy with Agent_Radius=20

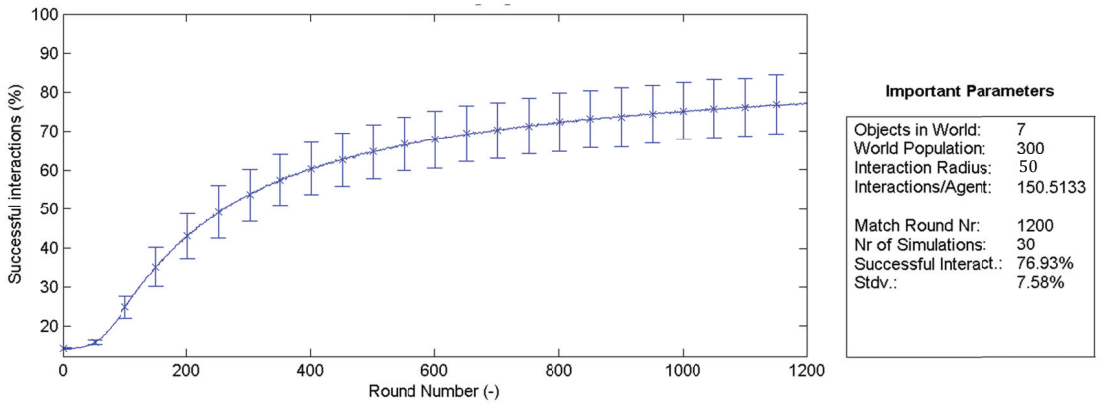


Figure Language Evolution, mean and std.dev. of the interaction accuracy with Agent_Radius=50

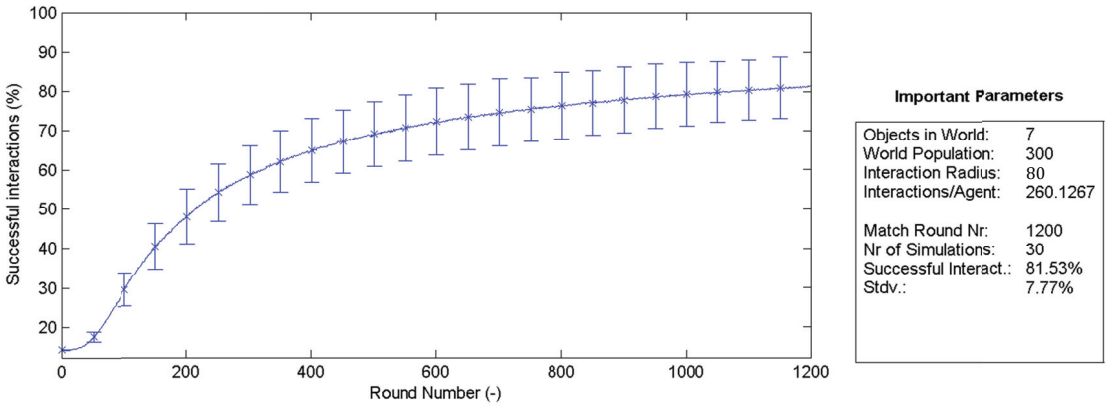


Figure 19: Language Evolution, mean and std.dev. of the interaction accuracy with Agent_Radius=80

The series from Figure 17 to Figure 19 presents the influence of the agent radius. (The randomizer used a seed number of 8713.) Obviously the more agents are within the interaction radius, the faster the steady state is reached. At the same time, the standard deviation increases.

For each radius 30 runs were processed. As seen in Figure 19 the final interaction accuracy values varies between 70% and 90%. There are seven objects and seven noises in the agents' world, which gives more than 800'000 combinations. The more they interact, the more they get in conflict with foreign dictionaries.

It was very time consuming to run 30 complete iterations with 300 agents and 7 objects. (And the higher the agent radius was, the more interactions were processed…), so the program was limited to run only a few radians.

The evolution and influence of the interaction radius can be guessed by interpreting the shown figures. To get significant results, more data is required.

6 Conclusion and Outlook

Compared with the literature, the developed model produced quiet similar results in scenario 1 (where the spatial component was neglected). We never reached a full 100% accuracy due to the random process.

With scenario 2 the coexistence of different languages and dialects was successfully simulated. As an extension of scenario 3, the influence of the interaction radius was shown in scenario 3.

In general one can say that the mobility and the size/density of the population heavily influence the amount of different languages evolving in the world.

Watching the real world, one can see a lot of dialects disappearing and a more common language becomes dominant. Compared to the civilizations of the middle age, the present size of population and interaction radius is much, much higher.

So the model developed in this thesis is somehow comparable to the real word, even its still on a very basic level of complexity.

The critical part of the model is the procedure where the memory is analyzed and the agent's dictionary is updated. Further research should primary focus this algorithm due to its heavy influence to the results.

To simulate the present globalization of the real world, a fourth scenario is proposed where the agent radius is increased over time.

Compared with the model of Gostolli the memory size and update lag are fixed. These two features could also be implemented in a future research.

7 References

Gostoli, U. (2008). *A Cognitively Founded Model of the Social Emergence of Lexicon.* Journal of Artificial Societies and Social Simulation vol. 11, no. 1 2.

Grim, P., Denis, P. S., & Kokalis, T. (2004). *Information and Meaning: Use-Based Models in Arrays of Neural Nets.* Group for Logic and Formal Semantics, Department of Philosophy, SUNY at Stony Brook: Kluwer Academic Publishers.

Nowak, M. A., & Krakauer, D. C. (1999). *The evolution of language.* Institute for Advanced Study, Princeton, NJ 08540: University of Oxford, Oxford, United Kingdom.

8 Appendix

8.1 Simulation result scenario 1

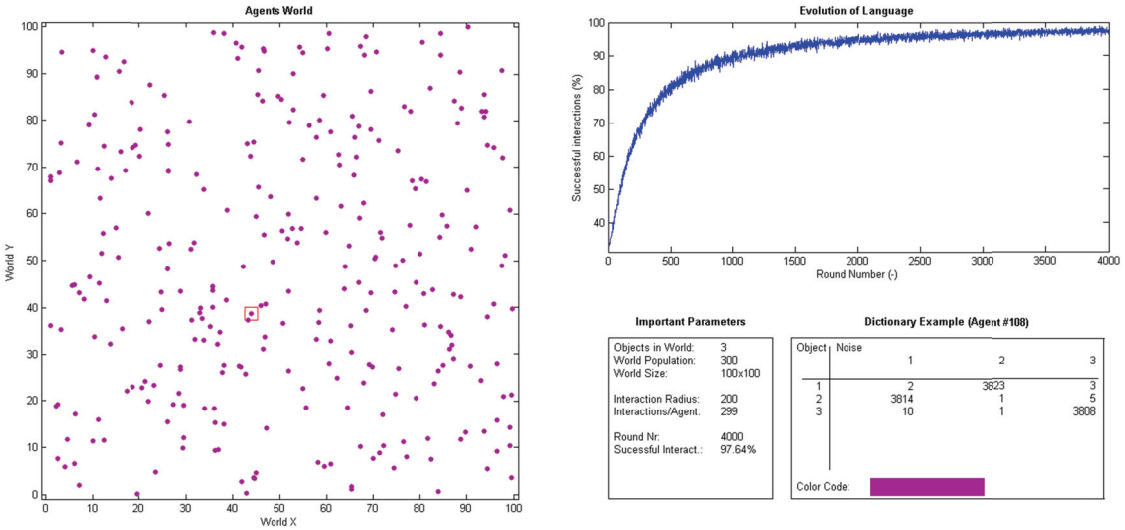


Figure 20: ONE single run of scenario 1 with only 3 objects

8.2 Simulation results scenario2

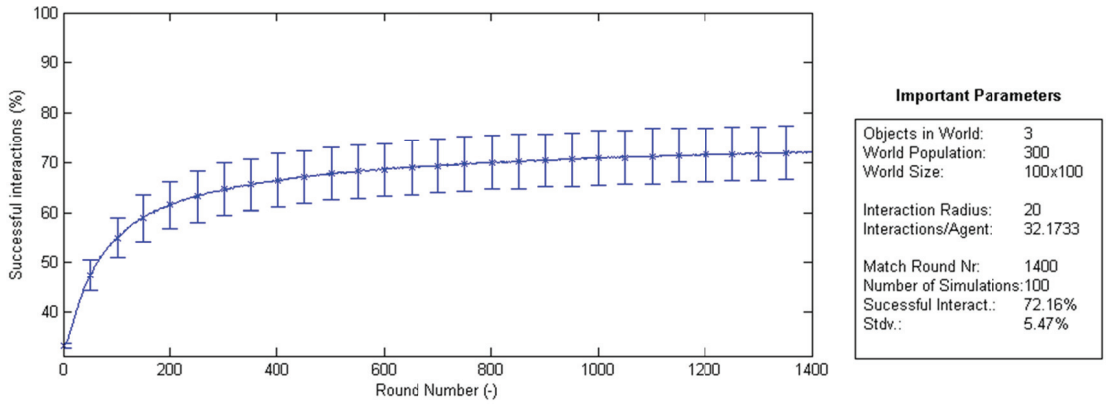


Figure 21: Evolution of language for 100 simulations with a population of 300 and with 3 objects only (SeedValue 4732)

8.3 Simulation results for scenario 2 with communication barrier

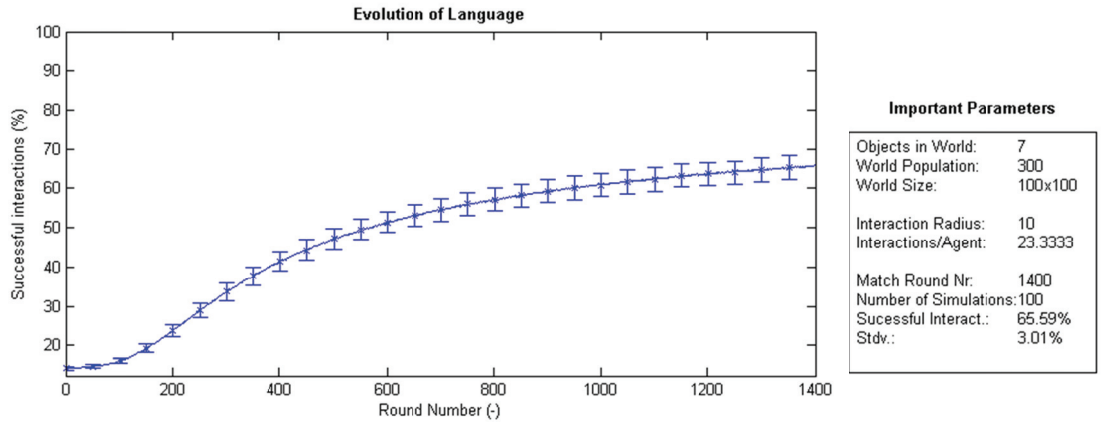


Figure 22: 100 simulations, interaction radius 20, with interaction barrier

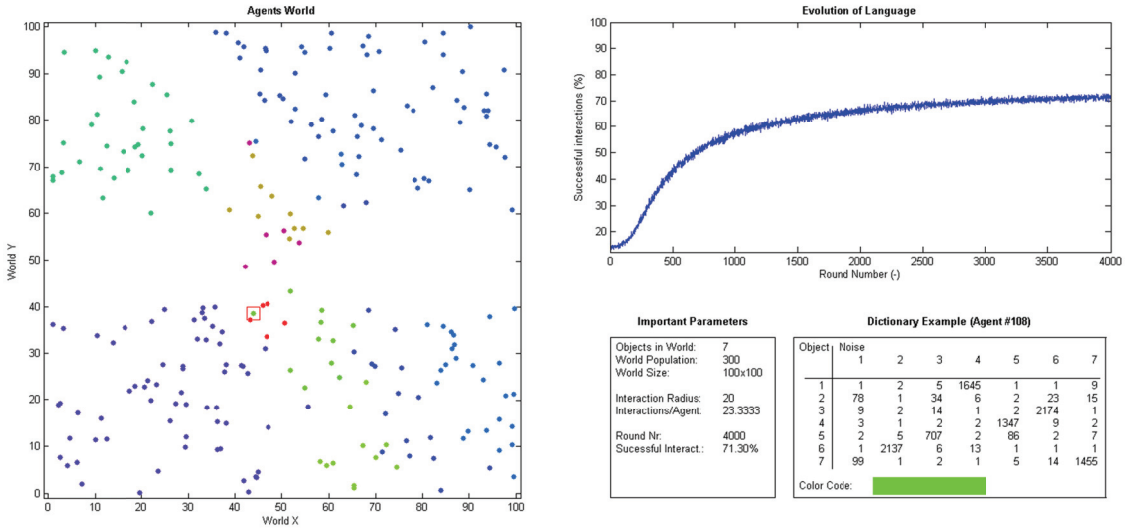


Figure 23: one simulation, interaction radius 20, with interaction barrier

8.4 MATLAB source codes

8.4.1 Initialisation

Script name: SINGLE_RUN_multiple_simulations.m

Script description: This Script will initialize and set some parameters for a single execution.

```
clear all;          % Clear Workspace
clc;
disp('AGENT BASED MODELING OF LANGUAGE EVOLUTION');
disp('Preparing Stuff for Language Evolution.');
disp('You can always interrupt the process by pressing CTRL+C');

% directory, where the graphical outputs should get stored.
% comment out or set to '' if you dont want to save the images.
% path is relative to the current folder
PATH_TO_FIGURES = ''; %'figures\';   % dont forget the ending \
PREFIX_OF_FIGURES = 'bild_';   % --> stored as "prefix_roundID.png"
% define world dimension
WORLD_X = 100;
WORLD_Y = 100;

WORLD_POP = 300;   % amount of agents
AGENT_RADIUS = 200;  % radius of agents interaction

OBJECTS = 3;          % amount of words/objects
MAXROUNDS = 4000;    % emergency break, if max rounds are reached

SeedValue3 = 8713;    % seed value for drawing agents x- and y-coordinates

SeedValue = 8713;     % Stream for iteration
STREAM=RandStream('mt19937ar','Seed',SeedValue);
RandStream.setDefaultStream(STREAM);

% set some randomized X and Y coordinates for the world population
STREAM3=RandStream('mt19937ar','Seed',SeedValue3);
Agents_XY = [rand(STREAM3,WORLD_POP,1)*WORLD_X
rand(STREAM3,WORLD_POP,1)*WORLD_Y];

% %-----------------------------------------------------------------------
% % 'delete' some agents in oderer to simulate a communication barrier
% for i=1:WORLD_POP;
%     if (Agents_XY(i,1)<=40 || Agents_XY(i,1)>=60) && (Agents_XY(i,2)>=40 &&
Agents_XY(i,2)<=60);
%         Agents_XY(i,1)= Agents_XY(i,1)*-10000;
%     end
% end
% %-----------------------------------------------------------------------

MAXSTATROUNDS=1;
statistical_round=1;
while(statistical_round<= MAXSTATROUNDS);
% create an initional dictionary where
%   x: noise-ID
%   y: meaning of the noise
%   z: agent-ID
Agent_Dict = ones(OBJECTS,OBJECTS,WORLD_POP);

% calculate the distance of an Agent to all other Agents
% an example, using all x-coordinates of 3 agents: [1 3 6]
% repmat them as many times as there are agents. (3 times)
% and subtract the transposed matrix.
%               1 3 6      1 1 1       0  2  5
%               1 3 6  --  3 3 3  ==  -2  0  3
%               1 3 6      6 6 6      -5 -3  0
% note the zeros which are the distance to the agent itself.
```

```
Prepared = repmat(Agents_XY(:,1),1,WORLD_POP);  % x-component
Dist_Each_X = Prepared - Prepared';

Prepared = repmat(Agents_XY(:,2),1,WORLD_POP);  % y-component
Dist_Each_Y = Prepared - Prepared';

Agent_Distance = sqrt(Dist_Each_X.^2+Dist_Each_Y.^2);
% remove temporary variables to save memory
clear Prepared Dist_Each_X Dist_Each_Y;

count_rounds = 0;
stored_quote = zeros(MAXROUNDS,1);  % für zeitlichen verlauf

% Find all interacting Agents
% where the distance is smaller than action radius:
% When agent 1 interacts with agent 2 out of three agents,
% the interaction matrix would look like:    1 1 0
%                                            1 1 0
%                                            0 0 1
% note the ones/trues on diagonal...
Interact_Bool = Agent_Distance<=AGENT_RADIUS;

% Prevent monologues (set diagonal to zero)
% so the previous example would look like:   0 1 0
%                                            1 0 0
%                                            0 0 0
Interact_Bool = Interact_Bool-eye(WORLD_POP);
% upper right triangular of the matrix means: x listens to y
%  lower left triangular of the matrix means: x talks to y
Interact_Index=find(Interact_Bool);

Average_Interactions_per_Agent = mean(sum(Interact_Bool));



%=========================================================================%
%    END OF INITIALIZATIONS, now run the iterative process                %
%=========================================================================%
run([pwd '\Iteration.m']);

PATH_TO_FIGURES='';  % reset, because we do not want to store the last plot
run([pwd '\Output.m']);
statistical_round=statistical_round+1;
end
reset(STREAM);
```

8.4.2 Iteration

Script name: Iteration_multiple_simulations.m

Script description: This script iterates the agents interactions until the language, gets stable on a high success rate

```
% check if the parameters are defined, if not, quit the script
if (exist('Interact_Bool','var')==0)
    clc;
    disp('AGENT BASED MODELING OF LANGUAGE EVOLUTION');
    disp('Iteration.m is just a part of a whole set of scripts and');
    disp('cannot be executed on its own. Some initialisations are');
    disp('required! You may like to start with SINGLE_RUN.m');
    return;
end

% everythings fine. so we'll start the iteration.
disp(['Agents now start to interact. Mean interactions per agent: '...
        num2str(Average_Interactions_per_Agent) ]);
%=========================================================%
% begin round
%=========================================================%
% main process:
% 1) choose object (randomly)
% 2) look in dictionary and get the "best" noise of that object
% 3) make that noise to the other agent
% 4) other agent will do a lookup in his dictionary to get its meaning
% 5) understood object is compared to the choosened object in step 1
% 6) if they're equal, increasing the temporary memory
% 7) check the memory for absolute maximums
% 8) update dictionary for next round

while(count_rounds< MAXROUNDS)
    % set a random object for all interactions
    % --> random integer x, with 1<=x<=objects
    Object_Selected = floor(rand(size(Interact_Index))*OBJECTS)+1;

    % all agents will try to create a (slightly randomized) dictionary
    % to use in this round. the random process is weighted by the
    % dictionary itself.

    % -------------------------
    % PROCESS DICTIONARIES
    % Get the Max weight of each Objects for all Agents at once
    % matrix dimensions will become: OBJECTS * 1 * WORLD_POP
    % e.g. a 3-object-dictionary like ___  will get the rowsums
    %   324   43   23        390
    %    54   21  422        497
    %    89  511   33        633
    rowsum = sum(Agent_Dict, 2);

    % Random value, weighted by the row sum of the dictionary
    % "rand" picks a random float with 0 <= x < 1
    % taking the 3-object-dict above, randompick would look like:
    %    235   (where x1 is 0 <= x1 < 390)
    %    342    ...
    %    487   (where x2 is 0 <= x2 < 633)
    randompick = rand(OBJECTS,1,WORLD_POP).*rowsum;
    % cumulative of all objects in the dictionary
    % applying this cumsum on the 2-object-dictionary, rowcum becomes
    %    0+324  0+324+43   0+324+43+23          324  367   390
    %    0+54   0+54+21    ...+422               54   75   497
    %    0+ 89  0+ 89+511  ...+33                89  600   633
    rowcum = cumsum(Agent_Dict,2);
```

```
    % find all noises, where the cumulative exceeeds the random value
    % just compare each row of the rowcum with the randomly picked value:
    % like: 324>235?  367>235?  390>235?  as boolean:    1 1 1
    %        54>342?   75>342?  497>342?                 0 0 1
    %        89>487?  600>487?  633>487?                 0 1 1
    bool = rowcum>repmat( randompick, 1, OBJECTS );
    % each column should only store the FIRST found value
    % we use a special trick to reset the followed trues to false
    % cumsumm of the boolean matrix will become like:
    %     1 2 3    the equation applied, it will become:
    %     0 0 1
    %     0 1 2
    %           1>1?   2>1?   3>1?    as boolean:     0 1 1
    %           0>1?   0>1?   1>1?                    0 0 0
    %           0>1?   1>1?   2>1?                    0 0 1
    % the first boolean subtracted by the second boolean will result in:
    %               1 1 1  minus    0 1 1     equals    1 0 0
    %               0 0 1           0 0 0               0 0 1
    %               0 1 1           0 0 1               0 1 0
    Object2Noise = bool - (cumsum(bool,2)>1);
    % so, the first object is expressed by the first noise, the second
    % object by the third noise and the third object by the second noise.

    % what we need is a transformation vector of the kind of  [1 3 2]'
    %   1 0 0     [a b]=find( )       1       1
    %   0 0 1                   a = 3    b = 1    all we need is vector a
    %   0 1 0                       2       1

    % create the Agents transformation vector
    % (replace agent z's object #x by noise #y)
    transform = sum( Object2Noise.*repmat(1:OBJECTS,[OBJECTS 1 WORLD_POP]),2);
    % so we apply the transformation to the selected objects to transform
    % them to noises.


    % the preselected objects for all interactions will get transformed
    % to individual noises (depending on the transformation-vectors)
    TransIndex= mod(Interact_Index-1, WORLD_POP).*OBJECTS+Object_Selected;
    Noises = transform(TransIndex);

    % DO THE SAME CRAP AGAIN... but this time vice versa:
    % the reciever-agent tries to interprete the heared noise.
    % read comments in the simmilar code above
    colsum = sum(Agent_Dict, 1);
    randompick = rand(1,OBJECTS,WORLD_POP).*colsum;
    colcum = cumsum(Agent_Dict,1);
    bool = colcum>repmat( randompick, OBJECTS, 1 );
    Noise2Object = bool - (cumsum(bool,1)>1);
    transform = sum( Noise2Object.*repmat([1:OBJECTS]', [1 OBJECTS
WORLD_POP]));
    TransIndex= floor((Interact_Index)/WORLD_POP).*OBJECTS+Noises;
    Understood = transform(TransIndex);


    % -------------------------
    % OK. Interactions are done: Objects were selected,
    % translated to noises and retranslated to interpreted meanings.

    % First Step: compare the understood object to object of sender:
    Matched = Understood==Object_Selected; % boolean matrix

    % Get Indexes of all successful interactions (where matched is true)
    Matches_Vec=find(Matched);

    % get the used combination of objects and noise which led to success
    % they are used to update the dictionaries
    Objects_Match = Object_Selected(Matches_Vec);
    Noises_Match = Noises(Matches_Vec);
```

```
% figure out the Indexes of all successfull agents. this step is a
% little complicated due to MATLABs style of indexing multi dimensional
% arrays.
% the index of a 3D-Array (with 3 objects) is something like:
% (:,:,1)= 1 3 5        (:,:,2)= 7  9 11         and so on.
%          2 4 6                 8 10 12
% so the entity with "coordinates" 1,3,2 gets the index number 11,
%           11 = 1*1 + (3-1)*2 + (2-1)*(3*2) = 1 + 4 + 6
Sender_Match = mod(Interact_Index(Matches_Vec)-1,WORLD_POP)+1;
Listener_Match = floor((Interact_Index(Matches_Vec)-1)/WORLD_POP)+1;

UpdateDictionary  = (Objects_Match) + ...
                    (Noises_Match-1).*OBJECTS;
SenderDict   = UpdateDictionary + (Sender_Match-1).*OBJECTS.^2;
ListenerDict = UpdateDictionary + (Listener_Match-1).*OBJECTS.^2;
UpdateDictionary = [SenderDict; ListenerDict];
% Now we need to update the dictionaries by the amount of successful
% combination of objects and noises.
% these combinationes were already stored in the variable Index.
% first, we need to sort this index:
% UpdateDict = [6 4 5 6 6 6 1 4]'   ==>   Sorted = [1 4 4 5 6 6 6 6]'
Sorted = sort(UpdateDictionary);
% some combinations could be successfull multiple times.
% Amount = [1 2 1 4]  and
%  Index = [1 4 5 6]
Amount=diff(find(diff([-Inf Sorted' Inf])))';
Index=Sorted(cumsum(Amount));
% now we know which combination of object/noise were successful

% ----------------------------
% to merge these combinations into the agents dictionaries, a
% workaround via a temporary memory matrix is used:
updating = zeros(size(Agent_Dict));
memory   = zeros(size(Agent_Dict));
memory(Index)=Amount;

for current=1:WORLD_POP
    % check the memory of each agent.
    curmat = memory(:,:,current);
    %  curmat =    10     0    10
    %               0    15    13
    %              15    28     0
    % look for a maximum for each object
    for round=1:OBJECTS
        maxima = curmat==max(max(curmat));
        % from the example above, the first maximum will be 28!
        % the second maximum is 13 (because both 15 are eliminated
        % later.) and the third maximum is 10 ...
        if sum(sum(maxima))>=2
            % there's no unique maxima. so ignore this round
            round=OBJECTS;  % skip the rest and go to the next agent!
        else
            % there's an unique maximum. find it and set EVERYTHING
            % on the same row and column to zero. this makes sure that
            % no other object uses the same noise (and vice versa).
            % looking at the example above three steps are:

            %  curmat =    10     0    10       with a=3, b=2
            %               0     0    13
            %               0     0     0
            %  after the second step it will become
            %  curmat =    10     0     0       with a=2, b=3
            %               0     0     0
            %               0     0     0
            % and the third step will result in a=1, b=1
            [a b]=find(maxima);
            curmat(a,:)=zeros(1,OBJECTS);
            curmat(:,b)=zeros(OBJECTS,1);
            % after three steps, the memory-matrix above is transformed
```

```
                % to the update-matrix below:
                %  curmat =     1     0     0
                %               0     0     1
                %               0     1     0
                updating(a,b,current)=1;
            end
        end
    end


    % merge the updates to the dictionaries:
    Agent_Dict = Agent_Dict + updating;


    % -------------------------
    % finishing the round, analyse the success-ratio and so on.
    count_rounds=count_rounds+1;
    success_ratio = sum(sum(Matched))/sum(sum(Interact_Bool));
    stored_quote(count_rounds,statistical_round)=success_ratio;

    % store plots
    if (exist('PATH_TO_FIGURES','var')==1 && size(PATH_TO_FIGURES,2)>0 ...
          && (mod(count_rounds,5)==1 || count_rounds<30) )
        run([pwd '\Output.m']);
    end
    disp(['Round #' num2str(count_rounds,'%05d') ' finished with '...
          num2str(success_ratio*100,'%2.2f') ...
          '% successful interactions']);
end

% -------------------------
% cleanup some useless or temporary variables.
 clear colcum colsum rowcum rowsum j randompick transform;
 clear bool object Noise CurrentAgent AgentNoises;

 clear Objects_Match Noises_Match Sender_Match Listener_Match;
 clear SenderDict ListenerDict UpdateDictionary;
 clear Index Amount AgentUnderstood Agent_Best Current

 clear Matched Matches_Vec Sorted TransIndex curmat current round updating;
 clear a b ans maxima;
```

8.4.3 Output

Script name: Output_multipl_simulations.m

Script description: This script plots the results of the iteration (or its results)

```
if statistical_round==MAXSTATROUNDS;
% check if the parameters are defined, if not, quit the script
if (exist('Interact_Bool','var')==0 || exist('Object_Selected','var')==0)
    clc;
    disp('AGENT BASED MODELING OF LANGUAGE EVOLUTION');
    disp('Output.m is just a part of a whole set of scripts and');
    disp('cannot be executed on its own. Some initialisations and');
    disp('Iterations are required! You may start with SINGLE_RUN.m');
    return;
end


%=========================================================%
% Graphical Output of the Language Evolution
%=========================================================%
% set "example point" to zero to disable the dictionary-output
if (exist('EXAMPLE_POINT','var')==0)
    EXAMPLE_POINT=floor(rand(1)*WORLD_POP)+1;
end
ASPECT_RATIO = 1680/850;            % fixed aspect ratio of plot


%=========================================================%
% Compute Agents Colors (depending on their dictionary)
%=========================================================%

% compute the weakness of the agents dictionaries.
% --> will become = 1 for extremely strong dictionaries
%     and approximates zero for extremely weak dictionaries (initial dict)
Agent_Dict_Strength = max(max(Agent_Dict));
Agent_Dict_Weakness = min(min(Agent_Dict));
test=(Agent_Dict_Strength-Agent_Dict_Weakness)./Agent_Dict_Strength;
test=test.^6; % increase the difference between strong and weak...
clear Agent_Dict_Strength Agent_Dict_Weakness;



% Convert Best Dictionaries to an unique value:
% first step, find Max of Dictionaries:
% in example:               0     0     1
%                           1     0     0
%                           0     1     0
Agent_Best = (Agent_Dict==repmat(max(Agent_Dict),OBJECTS,1));
Agent_Best = Agent_Best.*(cumsum(Agent_Best)==Agent_Best);
Agent_Value = zeros(WORLD_POP,1);


for Current = 1:WORLD_POP
% convert maximums to:      2     3     1
    line = sum(Agent_Best(:,:,Current).*repmat([1:OBJECTS]',1,OBJECTS));
% and combine it with:      27    9     3
    scaler = OBJECTS.^[OBJECTS:-1:1];
% to:                       27    18    0
% and sum them up to:                         45
    Agent_Value(Current) = sum(scaler.*(line-1));
% each Combination of Objects and Noises will lead to an unique Value
end
%  Normalize it to a value between 0 and 2*pi (bogenmass)
Agent_Value = Agent_Value./(OBJECTS^(OBJECTS+1)) .* 6.28318530717958647692;
clear line scaler;
```

```
R = 0.5 + sin(Agent_Value + 0.0)*0.5;                          % 0/3*pi
G = 0.5 + sin(Agent_Value + 2.094395102393195492308428 9)*0.5; % 2/3*pi
B = 0.5 + sin(Agent_Value + 4.188790204786390984616857 8)*0.5; % 4/3*pi
R = R.^1.3;  % increases the saturation of each color
G = G.^1.3;  % read more about the color pattern in the colors.m-file
B = B.^1.3;
alpha = 1-test(:);

R = (1-R).*alpha+R;
G = (1-G).*alpha+G;
B = (1-B).*alpha+B;
clear alpha test;

%=========================================================%
% Initialize the PLOT
%=========================================================%


% Reset the figure. the size of the figure depends on the screen resolution
% and should be calculated only on the first run.
if (exist('CURRENTFIGURE','var')==0)
    % get screen resolution
    scr=get(0,'ScreenSize');
    % and force a 16:9 display port, so the plots will look the same on
    % all computer systems and screens. optimized for: win seven

    %   X/Y of bottomleft-corner, width, height of figure
    if (floor(scr(3)/ASPECT_RATIO)>scr(4)-160)
        FIGUREPOS = [10; 60; floor((scr(4)-160)*ASPECT_RATIO); scr(4)-180];
    else
        FIGUREPOS = [10; 60; scr(3)-20; floor((scr(3)-20)/ASPECT_RATIO) ];
    end
    CURRENTFIGURE = 1;
    % converts action-radius to pixel
    GRAPHSCALING = FIGUREPOS(3)/1680 * 20;
    clear scr;
end

figure1 = figure('position', FIGUREPOS, ...
                 'CurrentObject',CURRENTFIGURE, ...
                 'Name','Graphical Output of Language Evolution', ...
                 'NumberTitle', 'off', ...  % removes "figure 1:"
                 'Color',[1 1 1]);          % background color: white

%=========================================================%

% get the positions of the world map, depending on world size
if (exist('WORLDPLOT','var')==0)
    maxwidth = 0.42;
    maxheight = maxwidth*ASPECT_RATIO;
    if WORLD_X>WORLD_Y
       width = maxwidth;
       height = maxheight * WORLD_Y/WORLD_X;
    elseif WORLD_X<WORLD_Y
       width = maxwidth * WORLD_X/WORLD_Y;
       height = maxheight;
    else
       width = maxwidth;
       height = maxheight;
    end
    WORLDPLOT = [0.46-width 0.925-height width height];
    clear width height maxwidth maxheight;
end
```

```
%=========================================================%
% DRAW WORLD MAP
%=========================================================%
axes1 = axes('Parent',figure1, 'Position',WORLDPLOT, ...
             'Color',[0.94 0.94 0.94]);
box(axes1,'on');    % draw a black border around the plot
hold(axes1,'all');
axis(axes1, [-1 WORLD_X+1 -1 WORLD_Y+1]);


% highlight example-point (if desired)
if (exist('EXAMPLE_POINT','var')==1 && EXAMPLE_POINT>0)
    % highlight the point by a red sqare
    plot(Agents_XY(EXAMPLE_POINT,1),Agents_XY(EXAMPLE_POINT,2), ...
        'Marker','s', 'MarkerSize',15, ...
        'Color',[1 0 0]);
end

%plot(1:count_rounds, stored_quote(1:count_rounds,statistical_round));
for Current = 1:WORLD_POP
    plot(Agents_XY(Current,1),Agents_XY(Current,2), ...
        'Marker','.', 'MarkerSize',15, ...
        'Color',[R(Current) G(Current) B(Current)]);
end

% Create labels and title for WORLD MAP
xlabel({'World X'});
ylabel({'World Y'});
title({'Agents World'},'FontWeight','bold');


%=========================================================%
% DRAW SUCCESSFUL INTERACTIONS
%=========================================================%
SUCCESSPLOT = [ WORLDPLOT(1)+WORLDPLOT(3)+0.08, ...
                WORLDPLOT(2)+WORLDPLOT(4)-0.4, ...
                1-WORLDPLOT(1)-WORLDPLOT(3)-0.1 , 0.4];
axes2 = axes('Parent',figure1, 'Position',SUCCESSPLOT, ...
             'Color',[0.94 0.94 0.94]);
box(axes2,'on');

nonzero = find(stored_quote>0);
cutpos = size(nonzero,MAXSTATROUNDS);
cutpos=max( min(400,MAXROUNDS), cutpos );

cuttedquote = stored_quote(1:MAXROUNDS,1:MAXSTATROUNDS).*100;
errorbar(1:50:MAXROUNDS,mean(cuttedquote(1:50:MAXROUNDS,:).'), ...
          std(cuttedquote(1:50:MAXROUNDS,:).'),'xb');
hold
plot(1:1:MAXROUNDS,mean(cuttedquote.'));

axis(axes2, [0 MAXROUNDS 100/OBJECTS-2 100]);
xlabel({'Round Number (-)'});
ylabel({'Successful interactions (%)'});
title({'Evolution of Language'},'FontWeight','bold');
```

```
%=========================================================%
% DRAW INFO-BOX
%=========================================================%
INFOPLOT=[  SUCCESSPLOT(1) WORLDPLOT(2) ...
            0.145 SUCCESSPLOT(2)-WORLDPLOT(2)-0.15];

% Create textbox
annotation(figure1,'textbox',...
    [INFOPLOT(1) INFOPLOT(4)-0.14 INFOPLOT(3) INFOPLOT(4)],...
    'String',{'Important Parameters'},...
    'FontWeight','bold',...
    'HorizontalAlignment','center',...
    'LineStyle','none');

annotation(figure1,'textbox',...
    INFOPLOT,...
    'String',{'Objects in World:','World Population:','World Size:',...
              '','Interaction Radius:','Interactions/Agent:',...
              '','Match Round Nr:','Number of Simulations:'...
              'Successful Interact.:','Stdv.:'},...
    'FitBoxToText','off');

annotation(figure1,'textbox',...
    [INFOPLOT(1)+0.095 INFOPLOT(2) INFOPLOT(3) INFOPLOT(4)],...
    'String',    {num2str(OBJECTS),num2str(WORLD_POP),...
                  [num2str(WORLD_X) 'x' num2str(WORLD_Y)],...
                 '',num2str(AGENT_RADIUS/2),...
                   num2str(Average_Interactions_per_Agent),...
                 '',num2str(count_rounds),...
                   num2str(MAXSTATROUNDS),...

[num2str(mean(stored_quote(count_rounds,1:statistical_round).')*100,'%2.2f') '%'],...

[num2str(std(stored_quote(count_rounds,1:statistical_round).')*100,'%2.2f') '%']},...
    'FitBoxToText','off',...
    'EdgeColor','none');



if (exist('EXAMPLE_POINT','var')==1 && EXAMPLE_POINT>0)
%=========================================================%
% DRAW DICTIONARY OF EXAMPLE POINT
%=========================================================%
    EXAMPLEPLOT=[  INFOPLOT(1)+INFOPLOT(3)+0.016 INFOPLOT(2) ...
                   SUCCESSPLOT(3)-INFOPLOT(3)-0.02  INFOPLOT(4)];
    if (OBJECTS>7)
        displayobjects=7;
    else
        displayobjects=OBJECTS;
    end

    %title
    annotation(figure1,'textbox',...
        [EXAMPLEPLOT(1) EXAMPLEPLOT(4)-0.14 EXAMPLEPLOT(3) EXAMPLEPLOT(4)],...
        'String',{['Dictionary Example (Agent #' num2str(EXAMPLE_POINT)
')']},...
        'FontWeight','bold',...
        'HorizontalAlignment','center',...
        'LineStyle','none');
    % box with initial column
    annotation(figure1,'textbox',...
        EXAMPLEPLOT,...
        'String',{'Object   Noise'},...
        'FitBoxToText','off');
    annotation(figure1,'textbox',...
        [EXAMPLEPLOT(1) EXAMPLEPLOT(2) 0.032 EXAMPLEPLOT(4)],...
        'String',{'','','',1:displayobjects},...
        'HorizontalAlignment','right',...
        'LineStyle','none');
    % 1st noise column
```

```
    DICTWIDTH=(EXAMPLEPLOT(3)-0.035)/displayobjects;
    for Current=1:displayobjects
        annotation(figure1,'textbox',...
        [EXAMPLEPLOT(1)+0.033+DICTWIDTH*(Current-1) EXAMPLEPLOT(2) ...
            DICTWIDTH EXAMPLEPLOT(4)],...
        'String',{'',num2str(Current),'',...
                    Agent_Dict(:,Current,EXAMPLE_POINT)},...
        'HorizontalAlignment','right',...
        'LineStyle','none');
    end
    % Create lines
    annotation(figure1,'line', ...
        [EXAMPLEPLOT(1)+0.034 EXAMPLEPLOT(1)+0.034],...
        [EXAMPLEPLOT(2)+0.05 EXAMPLEPLOT(2)+EXAMPLEPLOT(4)-0.015]);
    annotation(figure1,'line', ...
        [EXAMPLEPLOT(1)+0.01 EXAMPLEPLOT(1)+EXAMPLEPLOT(3)-0.005],...
        [EXAMPLEPLOT(2)+EXAMPLEPLOT(4)-0.07 EXAMPLEPLOT(2)+EXAMPLEPLOT(4)-0.07]);

    % color code information
    annotation(figure1,'textbox',...
        [EXAMPLEPLOT(1) EXAMPLEPLOT(2) 0.07 0.039],...
        'String',{'Color Code:'},...
        'LineStyle','none');
    annotation(figure1,'textbox',...
        [EXAMPLEPLOT(1)+0.07 EXAMPLEPLOT(2)+0.005 0.1 0.03],...
        'String',{''},...
        'VerticalAlignment','bottom',...
        'LineStyle','none',...
        'BackgroundColor',...
            [R(EXAMPLE_POINT) G(EXAMPLE_POINT) B(EXAMPLE_POINT)]);
end

% -------------------------
% store the figure to PNG (only if PATH is set)
if (exist('PATH_TO_FIGURES','var')==1 && size(PATH_TO_FIGURES,2)>0)
    if (exist('imageindex','var')==0)
        imageindex=1;
    else
        imageindex=imageindex+1;
    end
    imagepath = [pwd '\' PATH_TO_FIGURES];
    if (exist('PREFIX_OF_FIGURES','var')==1)
      imagepath=[imagepath PREFIX_OF_FIGURES num2str(imageindex) '.png'];
    else
      imagepath=[imagepath num2str(count_rounds) '.png'];
    end
    set(gcf,'PaperPositionMode','auto');
    saveas(gcf, imagepath);
    close(figure1);
end


% -------------------------
% cleanup some useless or temporary variables.
% clear cuttedquote cutpos nonzero;
clear R G B Agent_Value Agent_Best;
clear Current DICTWIDTH EXAMPLEPPLOT INFOPLOT SUCCESSPLOT;
clear axes1 axes2 displayobjects figure1;
end
```

Eigenständigkeitserklärung

Hiermit erklären die Autoren, dass diese Arbeit selbständig verfasst wurde, keine anderen als die angegebenen Quellen und Hilfsmittel verwendet, und alle Stellen, die wörtlich oder sinngemäss aus veröffentlichen Schriften entnommen wurden, als solche kenntlich gemacht wurden. Darüber hinaus erklären die Autoren, dass die Arbeit nicht, auch nicht auszugsweise, für andere Prüfung genutzt wurde.

Agreement for free-download

We hereby agree to make our source code of this project freely available for download from the web pages of the SOMS chair. Furthermore, we assure that all source code is written by ourselves and is not violating any copyright restrictions.

Zürich, 28.05.2011

Silas Menberg Manu Gomez