Radboud University

Artificial Intelligence Bachelor Thesis

Adaptive Learning

Using the Exclusion Principle

Supervisor:
Prof. P.W.M. Desain

Author:
I. Monster

External Advisor:
Prof. J.M. McQueen

SOW-BKI300 AI Bachelor Thesis
Artificial Intelligence
Faculty of Social Sciences
Radboud University Nijmegen
September 3, 2014

Contents

1 Abstract

This thesis project studied the effects of exclusion learning when learning L2 vocabulary. This method was implemented in a multiple choice flashcard application along with a control algorithm. The exclusion algorithm selected distractors that had already been recognized correctly, the control algorithm random distractors. Forty Dutch 11-to-13-year-old children studied English words with both algorithms and these results were compared within subject. There was no significant difference in the amount of repetitions the children needed to finish studying, the degree of confidence they got while studying or their score on a post-test when comparing conditions. The absence of a significant effect could have been caused by too much noise: the algorithms were too similar after a number of trials. Further testing showed the use of exclusion distractors is related to recognizing the correct answer. This implies the exclusion distractors could increase the chance of success when learning new words.

2 Introduction

Acquisition of the skill to speak and understand the language of people from other countries is important in a global society, and studying foreign languages has become an important part of our educational system and society. Dutch children start learning a second language like English at a very young age (some primary schools teach English to four-year-olds) and in high school children will also study French, German and sometimes Spanish. Unfortunately, learning a foreign language takes a lot of time and effort. O'Malley and Chamot (1990) [4] described many skills that are required to acquire a new language, such as phonology, morphology, syntax and a large vocabulary and each of these skills can be acquired using different strategies. The acquisition of speech sound categories requires listening to a native speaker and self-producing speech aloud, and grammar can be learnt implicitly by inferring rules from texts or by memorizing explicit rules of syntax and using them often. Vocabulary acquisition techniques include: learning the words in context, learning words that are associated and using new words in phrases.

Because it takes so much effort to proficiently learn a new language, it is of great importance to find better and more efficient ways of learning or teaching a foreign language. This study tested one particular new method to teach vocabulary using a word-learning application based on the exclusion principle.

2.1 Background

Computer programs for learning a foreign language are on the rise. A few years ago, high school students started using Teach2000 or Wrts (http://wrts.nl) for studying vocabulary. Now you can learn an entire new language at your own pace with Duolingo (http://duolingo.com). These study environments use different strategies for different purposes. Teach2000 and Wrts use graduated-interval recall, based on Pimsleur (1967) [5] and Duolingo has a gamified skill tree. Graduated-interval recall is mostly used in flashcard applications (such as Teach2000 and Wrts) and it ensures the user learns the words in an efficient way by presenting the words in a certain order; a question is repeated during the study session with increasing intervals of time. The gamified skill tree is a method where lessons, such as learning a new piece of grammar or vocabulary are represented as a new skill and you have to collect skill points in order to unlock the next level of skills.

Aside from these applied study methods that are implemented in widely used applications, researchers have found other ways to enhance vocabulary acquisition, including the exclusion principle. The exclusion principle was e.g. used in a study of Kaminski, Call, and Fischer (2004) [2]. For this study the team used a border collie dog that already knew many objects by name. Seven familiar objects were placed in a room along with one novel object. The team would then ask the dog for the novel object, using a novel name for that object. The dog retrieved the new item correctly. Apparently the dog was able to link the new word to the new object, because either he already knew the names of the other objects or he knew the other objects were not new. That is, he could exclude the other objects and thus deduce that the new name must refer to the new object. Four weeks after the initial exposure the dog could still correctly collect items from a set of novel and familiar items, which indicates the dog actually learnt the names of the new objects.

Wilkinson, Rosenquist, and McIlvane (2009) [7] tested the effects of exclusion learning on subjects with severe language impairments and intellectual disabilities and McIlvane and Stoddard (1981) [3] on subjects with severe retardation. Both studies found significant effects as the subjects recognized the novel item from the familiar items. Effects of exclusion learning have also been found with orthographic processing by Apel, Wolter, and Masterson (2006) [1]. Van Gogh, McQueen, and Verhoeven (2014) [6] have used the exclusion principle in a study with preschoolers. They studied whether early literacy skills can be enhanced by training lexical specificity and used exclusion learning as a strategy for the children to study word pairs that differed minimally.

This present study tests whether the exclusion principle would enhance Dutch primary school childrens ability to learn new English words in a flashcard application. In order to study the effect of exclusion learning without much interference from other factors, like social interactions in a classroom, the experiment was implemented in a single user computer application. This application implemented an adaptive, exclusion algorithm and a random, control algorithm without making the children aware of any differences. The application also had the advantage that it runs the entire experiment and stores the data, which made it possible to run the test on a larger scale.

In this experiment children in group 8 (age 11 - 13) participated, because they had had some experience with English and learning vocabulary. However, these children had only had English lessons for two years at school, so they are still very limited in their English lexicon. This made it easier to select words they had not seen before.

3 Research Question

This study examined the effect of personalized exclusion learning implemented in a multiple choice flashcard application for studying vocabulary. The application ran two sessions each with a wordlist of pairs of to-be-learned English words and their Dutch translations. Both sessions consisted of a learning phase, a short questionnaire and a post-test. During the learning phase one session used the exclusion algorithm and the other a control algorithm. The post-test presented each word one last time and the number of correct responses was measured.

Subjects studied new vocabulary in a multiple choice flashcard environment. Each item contained the stem (a Dutch word) and four possible translations of the stem. One of these options was the correct translation of the stem; the key. The other three words were distractors selected from the other English words in the wordlist. The subjects' task was to select the correct translation.

The two algorithms only differed in which distractors were chosen: the control algorithm selected random distractors from the entire wordlist, the exclusion algorithm selected distractors that were in previous correct items and thus adapted to the subject's responses after a few items. It was hypothesized that the subject would be able to use the exclusion principle to find the correct answer.

The conditions were compared to see if the exclusion algorithm improves L2 vocabulary acquisition. This is the main question that this study aimed to answer. Because the term enhance can be interpreted in various ways, this study was limited to testing only the following factors:

- Is the exclusion principle used in conducting the task?
  This was tested by analysing the difference between the amount of correct and false responses with known distractors versus the amount of correct and false responses when the distractors were unknown.

- Does the adaptive algorithm influence the performance score on a post-test?
  Enhancement would be a higher test score.

- Does the adaptive algorithm influence the time needed to learn all words?
  Enhancement would mean fewer repetitions are needed to finish the learning phase.

- Does the adaptive algorithm influence the confidence in the subjects own performance?
  Enhancement would mean the subject thought he or she would recognize more words on the post-test.

4 Research Methods

This experiment consisted of 2 sessions, a session with the exclusion algorithm and one with the control algorithm. Each session consisted of:

- A learning phase with either the exclusion or control algorithm, where the Dutch children studied a list unknown English words until the entire wordlist was mastered.
  This phase gave an indication of how long the children needed to study the words (the number of trials is the measure)

- A question: You have studied 15 words, how many words do you think you will answer correctly during the test?
  The question indicated if the child felt confident whilst studying the words.

- A post-test, where the learnt vocabulary was tested.
  The test indicated if the child had learnt the words correctly and provided a test score.

The order of conditions was counterbalanced: half of the children started with the exclusion session and the other children with the control session. The combination of algorithm and wordlist per session was also counterbalanced: half of the children had wordlist A with the exclusion algorithm and wordlist B with the control algorithm; the other children the wordlists the other way round. The results of both sessions were compared within subject.

4.1 Application

The application was an executable Java program, see Appendix B for the code. As it was preferable to be able to alter the vocabulary list or the list of children without modifying and recompiling the code, this data was stored externally and was read by the application at start-up.

The vocabulary list was a text file and contained 30 Dutch-English word pairs. In each session the child learnt 15 words, so the vocabulary list was split in two equal wordlists: wordlist A, see Appendix A.1 and wordlist B, see Appendix A.2. The list of children was also a text file and contained the names of all children that participated and with each name a number between 1 and 4 to indicate the child's group number. The application used these numbers to select the right sequence and combination of algorithm and wordlist for both sessions.

As the experiment ran, the program wrote data to another external text file. This data contained the name and group number of the child and for each trial, in both the learning phase and the post-test, the item (the stem and the four options) and the child's response. The response to the questionnaire was also stored.

At start-up the child selected his or her name from a list, read a short instructional text and started the first session. After this session, the subject was notified of being halfway and pressed a button to start the second session. When the second session was completed, the subject was notified of being finished and was allowed to close the application.

4.1.1 A Trial

The adaptive exclusion algorithm and control algorithm used the same single trial four-choice format. Both presented the items in the following manner: The stem was a Dutch word from the wordlist and the four options presented consisted of the key and three distractors (e.g. 'kloof' was presented as key along with 'blunt', 'single', 'stage' and 'gap' as distractors; see Figure 1).

4.1.2 Feedback

The child tried to pick the correct translation and clicked it. If the response was correct; the feedback box turned a green color and positive feedback was presented ('Goed zo, kloof is gap'); see Figure 2. If the

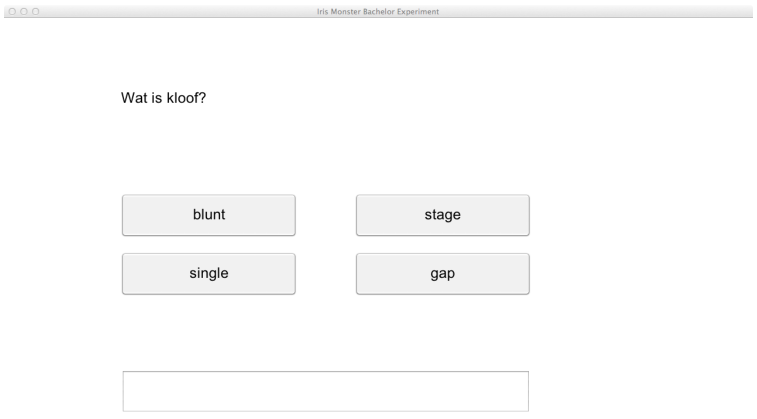

Figure 1: Display of an item

response was not correct; the feedback box turned red and correcting feedback was presented ('Oeps, kloof is gap').

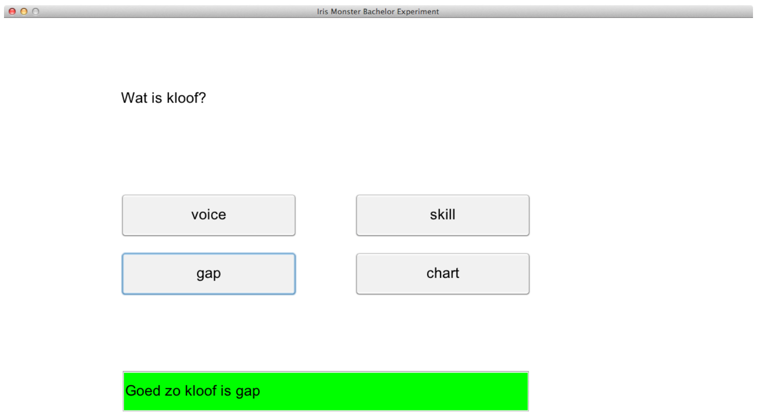

Figure 2: Display of an item with positive feedback

4.1.3 Learning phase procedure

At each trial the algorithms chose a random word from the wordlist as stem to create the next item. Words that were recognized correctly three times were not asked again, thus the learning phase was finished when all words were recognized correctly three times. The program recorded this with a *correct*-vector that initially contained a familiarity value of zero for each word in the wordlist. Every time the child recognized a word correctly the corresponding familiarity value of that word would increase by one. If the child did not recognize a word correctly, the familiarity value of that word remained the same. If the word that was mistaken for the target had a positive value, the familiarity value of that word would decrease by one. Under these circumstances, therefore, the assumption was that the child did not know this word (even though he or she gave the correct answer at least once, this might have been a lucky guess). When a familiarity value reached three, the corresponding word was not asked again, but could occur as a distractor for another word.

It was possible the subject made a mistake and selected the word with value three. The value of that word dropped and needed to be recognized correctly again. When all familiarity values reached three, the learning phase was finished.

4.1.4 Distractors

The difference between the two versions lay in which distractors were chosen. The control algorithm randomly selected English translations from the wordlist. The adaptive algorithm used the *correct*-vector to select distractors. At each trial, the algorithm first calculated how many words had been recognized correctly at least once (is the value in the correct-list greater than zero?) and placed these words in a separate *possible distractors*-list. If this list contained more than four words, it was possible to pick random exclusion principle distractors from this list. Otherwise random distractors were selected from the entire wordlist.

4.1.5 Post-test

The post-test in both sessions presented every word once and selected random distractors on each trial. Positive or negative feedback was not presented anymore. All other settings and appearances were the same as during the learning phase.

4.2 Testing procedure

The experiment was performed with 40 children in group 8 (age 11 to 13 years old). The background of the children varied; some had less or more experience with English and while all children spoke Dutch fluently Dutch was not the native tongue for all children.

4.2.1 Words

The vocabulary list that the children had to study consisted of randomly selected words from two different textbooks for older children. The majority of the list was filled with 20 words from a textbook for children in 4 VWO (pre-university secondary education, which is the highest variant of secondary education in The Netherlands, pupils in class 4 are aged 15-17 years old). The remaining 10 words were selected from a textbook for children in 2 VWO (pupils are 13-14 years old).

This particular mixture was chosen after evaluating the results of a pilot study conducted with children in group 7 (age 10 - 12). These children studied 30 words from the 2 VWO textbook. Some children were already familiar with about 5 words before participating the experiment, which changed the effect of learning the words. All children also answered almost all questions correctly on the post-test. This created a ceiling effect and obstructed finding any differences between the two conditions.

It was not possible to foresee the negative consequences of selecting difficult words. It was possible the children would find the words too difficult, then get frustrated and lose their motivation and self-esteem. Therefore some simpler words were added to the vocabulary list.

All words needed to fulfil the following criteria:

- The word was a singular noun.
- The Dutch and English translations of a word did not share a similar (or almost similar) written form (e.g. help - hulp was discarded).
- The word did not appear on the vocabulary list of the children in group 8.

In order to test both algorithms independently, the wordlist was split in two parts; one for each algorithm (each with 5 words from the easier wordlist). Half of the children started with the adaptive-exclusion algorithm and the other half started with the control algorithm. After this first session, the children did another session with the other algorithm. Although the two wordlists were designed to have the same level of difficulty (the results showed there was no difference between the two wordlists in the amount of trials or performance score), it was best to counterbalance for differences. So in the end there were four groups, see Table 1.

All subjects were assigned at random to one group, with equal numbers of subjects in each group.

Table 1: Sessions per group

Group	*First Session*	*Second Session*
1	Exclusion Algorithm	Control Algorithm
	Wordlist A	Wordlist B
2	Exclusion Algorithm	Control Algorithm
	Wordlist B	Wordlist A
3	Control Algorithm	Exclusion Algorithm
	Wordlist B	Wordlist A
4	Control Algorithm	Exclusion Algorithm
	Wordlist A	Wordlist B

4.2.2 Instructions

Before the children started the experiment they received instructions in Dutch. The instructions explained what the children could expect and what they had to do. The children were not told what the difference between the two sessions was or that they could use the exclusion principle in one session. This particular study tested if the children would subconsciously use the exclusion principle without being told to.

Aside from these practical instructions, the experimenter stressed the importance of paying attention during the learning phase. The program could go on indefinitely, so the children were motivated to finish quickly with rewards. When they had finished the experiment, they got a snack and were allowed to play outside for the rest of the afternoon.

4.3 Pilot study

Before the main experiment was performed, two pilot studies were done. The first pilot study was very small and conducted on 2 children aged 13 years old. This study showed which feedback exposure time was best. The second pilot study was of a larger scale and conducted on 22 children in group 7 (age 10 - 12). This pilot study tested whether the application would work properly, and established which vocabulary lists were good to use and what the right amount of repetitions per word was.

4.3.1 Feedback exposure

During the learning phase the subject got feedback after selecting an answer. This feedback was presented for 1,5 seconds. This amount appeared to be a good balance. Some other exposure times were also tested during the first pilot study: 1 second, 3 seconds and 4 seconds. The first pilot study showed that an exposure time of less than 1,5 seconds made it too difficult for the child to properly read the feedback. Especially if the feedback was correcting feedback, the children complained they could not read the correct translation. Exposure of more than 1,5 seconds was actually very good when the children got correcting feedback, but the duration was too long in case of positive feedback: the children got distracted when they had to wait 3 or 4 seconds for the next word to appear. Apparently the children did not read and learn the positive feedback as thoroughly as with the correcting feedback. The child saw the green feedback box and did not bother reading the feedback message.

4.3.2 Repetition

During the pilot study with group 7 the number of correct repetitions per word was set to four. This high amount of repetition was not necessary. Almost all children had a perfect score on the post-test, which indicated they had studied the words too well. Also, it took the children quite a long time to finish both learning phases. A small mistake was easily made and because the program was quite strict, the child repeated words over and over. Because this frustrated the children, they would make more mistakes and almost got stuck in the learning phase.

Therefore the amount of correct repetitions was set to three. As can be seen below this was a good choice. The children were finished after fewer repetitions, which was positive for their concentration level. The post-test showed the children really learnt the words with fewer repetitions as well.

5 Results

5.1 Pre-processing

Before the data could be analysed it had to be pre-processed. 40 Children participated in the experiment, but one of these children accidentally closed the application halfway through the experiment. His results were removed from the data. The raw data showed some large outliers in the trial variables (e.g. subjects needing more than 399 trials). All subjects with more trials during the learning-phase than the trial mean plus two times the standard deviation were removed from the data, see Equation 1.

$$Number of trials > (mean(trials) + 2 * stdev(trials)) \tag{1}$$

Four subjects (10 %) were removed from the data due to this rule (35 subjects remain). The resulting data is summarized in Table 2.

Table 2: Data summary: Mean (Standard Deviation)

	Number of learning trials [1]	*Child's confidence* [2]	*Post-test (amount correct)*[2]
Exclusion condition	70.2 (25.49)	10.75 (3.10)	14.34 (0.87)
Control condition	68.85 (16.09)	10.66 (2.90)	14.22 (1.23)

[1]Minimum score = 45, no maximum score
[2]Maximum score = 15

5.2 Effect of conditions

5.2.1 Test for normal distribution

After pre-processing the data was subjected to a Shapiro-Wilk test, which concluded the data is not normally distributed ($p < 0.05$ for all variables). The assumption that the data is normally distributed can therefore not be met and a non-parametric test has been selected to test the data for a significant difference between the related samples in this experiment.

5.2.2 Analysis

The selected analysis is a non-parametric test that compares two related samples: the Wilcoxon Signed Ranks test. This test showed that the exclusion algorithm, in comparison to the control algorithm, did not elicit a statistically significant change in:

- The number of trials needed in the learning-phase *(Z = 0.000, p = 1.000)*.
  The median score was 61 in the exclusion trials and 64 in the control trials.
- The confidence based on the questionnaire *(Z = -0.362, p = 0.717)*.
  The median score was 10 for both sessions.
- The performance on the post-test *(Z = -0.014, p = 0.989)*.
  The median score was 15 for both sessions.

5.3 Effect of distractors

A second analysis was performed to study the effect of distractor-familiarity. For this analysis each trial (of both conditions) was categorized by a program based on the familiarity values of the distractors computed by the application:

- Distractor familiarity 0: At least one of the distractors has familiarity 0 (e.g. 0 0 0, 0 1 0, 0 3 3) *These were no exclusion-distractor sets.*
- Distractor familiarity 1: All distractors had familiarity 1 or higher (e.g. 1 1 1, 1 2 1, 1 3 3)
- Distractor familiarity 2: All distractors had familiarity 2 or higher (e.g. 2 2 2, 2 2 3, 2 3 3).
- Distractor familiarity 3: All distractors had familiarity 3. (3 3 3).

The familiarity value of the key (i.e. the correct word in a given display) was also taken into account. Table 3 summarizes of the percentage of correct trials.

Table 3: Percentage correct trials

Distractor Familiarity	0	1	2	3
Key Familiarity				
0	60.3 %	79.7 %	77.3 %	79.1 %
1	75.6 %	82.2 %	83.7 %	81.4 %
2	88.0 %	84.1 %	84.8 %	89.6 %

A logistic regression analysis (an analysis to test the influence of predictors on a dichotomous dependent variable) was performed to predict the outcome (correct or not correct) of a trial with the familiarity of the distractors and key as predictors (see Table 4). Crude odds ratios with corresponding 95 % confidence intervals (CI) were calculated for these two predictors. The validity of the resulting model was assessed using a Chi-square analysis. Interaction effects and collinearities were checked for both significant factors.

Table 4: Output of logistic regression analysis

	B	*S.E.*	*Upper*	*Exp(B)*	*Upper*
Constant	0.538	0.059	1.712		
Distractor	0.473 *	0.064	1.416	1.605	1.818
Key	0.621 *	0.068	1.627	1.860	2.126
*Distractor * Key*	-0.218 *	0.048	0.731	0.804	0.884

* $p < 0.001$
R^2 (Nagelkerk) = 0.067
Model predicted 77.8 % of the cases correctly
Chi-square = 217.251 ($p < 0.001$)

The results of the analysis showed that the outcome of the trial (right or wrong answer) depends on the familiarity of the distractor and the familiarity of the key. However, this study focuses on the effect of the distractors. The analysis indicated that the distractors influenced the outcome of the trial, but it is difficult to directly interpret the influence of key-familiarity. Therefore the probability of a correct trial based on familiarity of the key and distractors was calculated using the following probability formula (with *key* = 0, 1 or 2 and *distractor* = 0, 1, 2 or 3), see Equation 2.

$$p_{correct} = \frac{e^{0.538+0.621*key+0.473*distractor-0.218*key*distractor}}{e^{0.538+0.621*key+0.473*distractor-0.218*key*distractor}+1} \quad (2)$$

The resulting probabilities are visualized in Figure 3.

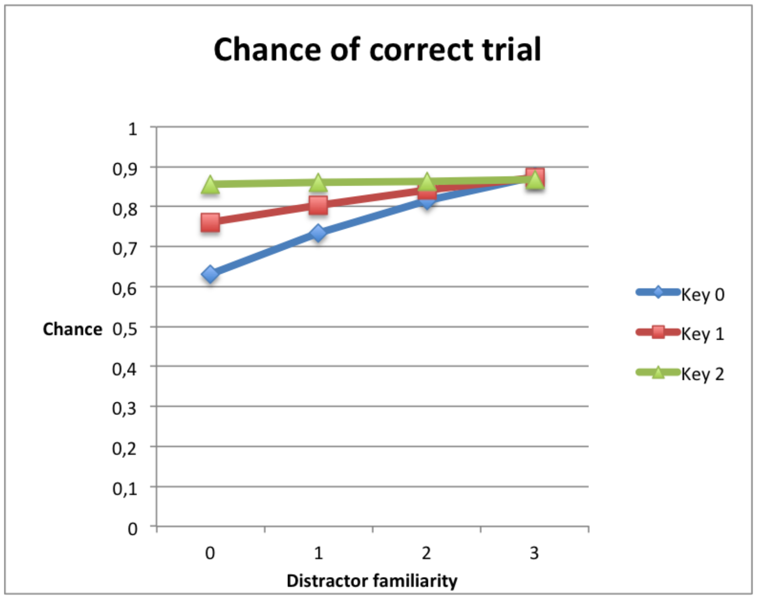

Figure 3: Chance of correct trial based on B-values.

Figure 3 shows the effect of distractor familiarity is most present when the key has not been recognized correctly before (0.631 for unknown distractors and 0.876 for distractor familiarity 3). The effect of the distractors is less when the key gains familiarity (0.855 for unknown distractors and 0.868 for familiar distractors).
Figure 4 shows the percentage of correct predicted trials in the data according to the model, categorized per familiarity group (key and distractor familiarity). This shows the model fitted the categories with a low key and distractor familiarity worst and got better as the familiarities grew.

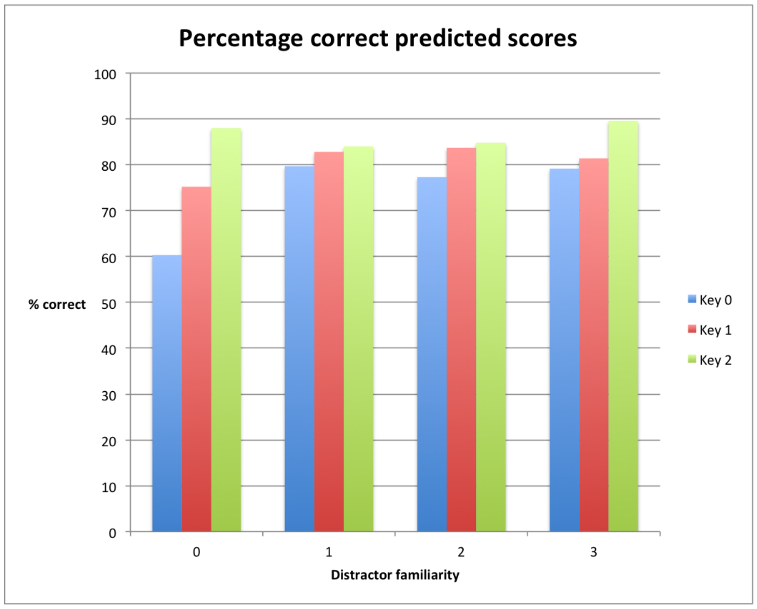

Figure 4: Percentage correct predicted scores

6 Discussion

Comparison of the two algorithms did not elicit a significant effect of the adaptive, exclusion algorithm while other studies (the studies on item recognition [3] [7] and the study on orthographic processing [1]) have found effects of the exclusion principle. A possible explanation is that the algorithms were too similar in this particular study. The first trials of the learning-phase during the exclusion session were the same as the first trials of the control algorithm, as there were no previously recognized words to choose distractors from. And after a number of trials, the control algorithm started to resemble the exclusion algorithm, as the number of correctly recognized words grew and were randomly chosen as distractors.

Another explanation could be that other studies, like the study of Van Gogh et al [6] informed their subjects they could use exclusion learning as a tool and in this study the children did not know they could do this. Maybe the effect of the exclusion principle is greater when it is consciously used?

The logistic regression analysis showed that the use of exclusion distractors does influence the chance of a correct trial and this chance increases as the distractors gain familiarity. Especially when the key was unknown, distractors familiarity improved the chance the children would select the correct answer. This implies the children did use the exclusion principle: they do not know what the right answer was on a given trial, but they knew the other options could not be the right answer so they could still select the right answer.

6.1 Future Work

In this exact setting the algorithms could be improved to differentiate the two sessions better and thereby increasing the chance of finding a more visible effect:

- Instead of selecting distractors from all words that have been recognized at least once correctly, the exclusion algorithm could favour the words that have been recognized most often for words that have been recognized the least. The control algorithm could chose the worst set of distractors (the ones with the lowest familiarity value). This will prevent the subjects from using the exclusion principle.
- Some children repeated words more than 8 times. This ensured the child knew the right translation at the end of the learning phase, but it also created a ceiling effect: most children answered all questions correct at the post-test. The stop condition of three correct repetitions could be extended to a maximum of ten repetitions or three correct repetitions. This would result in fewer trials, which would also be beneficial for the childrens mood and concentration during the learning phase.

The positive influence of familiar distractors could be used to help students learn L2 vocabulary more easily. New vocabulary would first be introduced to the student by surrounding the key with familiar distractors from a previous lesson (which is also beneficial to refresh the students memory on the old words). At a certain point the student has recognized the new vocabulary a number of times and he or she might not need the help of exclusion distractors anymore. The study environment that the student is using could then adapt to a higher difficulty level by using random distractors.

This study did not test if there are negative effects of just using exclusion distractors to study. It could be possible that the help of the distractors causes the student to pay less attention to the new word and impair the students ability to find the correct answer without exclusion distractors. This should be tested before exclusion learning can be implemented in the teaching methods for children and high school students.

It could also be interesting to study if all subjects respond in the same manner to familiar distractors. Maybe some children did not use the exclusion principle at all and others did. To do this one could make the same plot in the results section for the percentage of correct predicted scores, but per subject instead of all subjects combined. And does this difference in strategy influence their learning performance?

6.2 Conclusion

The conclusion of this study is that although there were no effects found in the main analysis and although the post-hoc test did not show an effect of learning itself, it is possible that the positive effect seen in the post-hoc analysis could mean that exclusitivity would also help with vocabulary learning.

A Wordlists

A.1 Wordlist A

gap = kloof
single = alleenstaand
chart = lijst
blunt = bot
orphan = wees
humanity = mensheid
paralysis = verlamming
skill = vaardigheid
voice = stem
bliss = geluk
acquaintance = kennis
stage = podium
prejudice = vooroordeel
ally = bondgenoot
exception = uitzondering

A.2 Wordlist B

performer = artiest
attitude = houding
widow = weduwe
kidney = nier
prescription = recept
corridor = gang
matrimony = huwelijk
support = steun
gig = optreden
gratitude = dankbaarheid
ancestor = voorouder
tribe = stam
germ = bacterie
track = nummer
rage = woede

B Code

B.1 Algoritme

```
package aiThesis;

public class Algoritme
{
	private int repeatTraining;
	private int repeatTesting;
	private int feedbackTime;

	/**
	 * Create new algoritme.
	 */
	public static void main(String[] args)
	{
		new Algoritme();
	}

	/**
	 * Set changeable settings and start model.
	 */
	public Algoritme()
	{
		repeatTraining = 3;
		repeatTesting = 1;
		feedbackTime = 2000;

		new Model(repeatTraining, repeatTesting, feedbackTime);
	}
}
```

B.2 Control

```java
package aiThesis;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Random;

public class Control extends Experiment
{
	private Random rand = new Random();

	public Control (ArrayList<Word> wordList, int repeat)
	{
		super(wordList, repeat);
	}

	/**
	 * Selects the distractors (just chooses 3 random distractors and the correct answer and
	     shuffles them).
	 */
	@Override
	public ArrayList<Integer> indexDistractors (ArrayList<Integer> distractorIndices,
	    ArrayList<Integer> words)
	{
		while (distractorIndices.size() < 4)
		{
			int newDistractor = rand.nextInt(wordList.size());
			if (! (distractorIndices.contains(newDistractor)))
			{
				distractorIndices.add(newDistractor);
			}
		}

		Collections.shuffle(distractorIndices);
		return distractorIndices;
	}
}
```

B.3 Controller

```
package aiThesis;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class Controller implements ActionListener
{
	private Model model;

	/**
	 * Sets the model
	 * @param model
	 */
	public Controller(Model model)
	{
		this.model = model;
	}

	/**
	 * In case of action, model is called to process action.
	 */
	@Override
	public void actionPerformed(ActionEvent e)
	{
		model.processAction(e.getActionCommand());
	}
}
```

B.4 Exclusion

```
package aiThesis;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Random;

public class Exclusion extends Experiment
{
        private Random rand = new Random();

        public Exclusion (ArrayList<Word> wordList, int repeat)
        {
                super(wordList, repeat);
        }

        /**
         * returns the indices of the distractors, see comments below
         */
        @Override
        public ArrayList<Integer> indexDistractors (ArrayList<Integer> distractorIndices,
            ArrayList<Integer> words)
        {
                ArrayList<Integer> exclusionDistractors = new ArrayList<Integer>();
                ArrayList<Integer> normalDistractors = new ArrayList<Integer>();

                for (int i = 0; i < words.size(); i++)
                {
                        if (words.get(i) != 0)                              // A word can be a
                            distractor if it has been recognized correctly at least once
                        {
                                exclusionDistractors.add(i);        // So then add it to the possible
                                    exclusion distractors
                        } else { normalDistractors.add(i); } // Otherwise add it to the possible
                            normal distractors
                }

                if(exclusionDistractors.size() > 3)                 // Do we have enough possible
                    exclusion distractors?
                {
                        return randomDistractors(distractorIndices, exclusionDistractors); // Case
                            yes, call for random distractors with exclusion distractors
                } else { return randomDistractors(distractorIndices, normalDistractors); } // Else,
                    call for random distractors with normal distractors
        }

        /**
         * Picks random distractors from the possible distractors, puts them in the distractor
             indices list and shuffles them up.
         * @param distractorIndices this list contains the index of the correct answer
         * @param possibleDistractors this list contains the possible distractors
         * @return distractor indices
         */
        private ArrayList<Integer> randomDistractors (ArrayList<Integer> distractorIndices,
            ArrayList<Integer> possibleDistractors)
        {
                while (distractorIndices.size() < 4)
                {
                        int indexNewDistractor = rand.nextInt(possibleDistractors.size());
                        int newDistractor = possibleDistractors.get(indexNewDistractor);
                        if (! (distractorIndices.contains(newDistractor)))
                        {
                                distractorIndices.add(newDistractor);
                        }
                }

                Collections.shuffle(distractorIndices);
                return distractorIndices;
        }
}
```

B.5 Experiment

```
package aiThesis;

import java.util.ArrayList;
import java.util.Random;

public abstract class Experiment
{
	private boolean finished = false;
				// Boolean for finished experiment
	protected ArrayList<Word> wordList;
				// The wordlist with the actual words
	protected ArrayList<Integer> words = new ArrayList<Integer>();			// The words
	    list (words[i] = amount of times wordlist[i] has been correctly recognized)
	private Random rand = new Random();
				// Random, used a lot to select a random word or index
	private int questionIndex;
					// The index of the presented word in the words list
	private ArrayList<String> distractors = new ArrayList<String>();		// A string
	    array with the English translations of the distractors
	private Word currentWord;
					// The current word
	private Word previousWord;
					// The previous word (needed to display feedback)
	private ArrayList<Integer> distractorIndices = new ArrayList<Integer>(); // Array with
	    indices of the distractors
	private boolean correct;
					// Boolean that is true if the user has selected the
	    correct answer
	private int repeat;
						// Number of times each word has to be recognized
	    correctly
	private String userAnswer;
					// String with the answer of the user

	/**
	 * Construction of a basic experiment. Needs to make new array with zeros length of
	     wordlist.
	 * @param wordlist
	 * @param repeat
	 */
	public Experiment (ArrayList<Word> wordlist, int repeat)
	{
		this.wordList = wordlist;

		this.repeat = repeat;

		for (int i = 0; i < wordList.size(); i++)
		{
			words.add(0);
		}
	}

	/**
	 * Checks if all words have been recognized correctly enough times
```

```
     * @return finished
     */
    public boolean finished ()
    {
            finished = true;
            for (int i = 0; i < words.size(); i++)
            {
                    if (words.get(i) < repeat)
                    {
                            finished = false;
                    }
            }
            return finished;
    }

    /**
     * Abstract function to generate random distractors
     * @param distractorIndices
     * @param words
     * @return Arraylist with position of distractors in words
     */
    public abstract ArrayList<Integer> indexDistractors (ArrayList<Integer> distractorIndices,
        ArrayList<Integer> words);

    /**
     * Simply returns correct (user gave correct response)
     * @return
     */
    public boolean getCorrect()
    {
            return correct;
    }

    /**
     * Sets boolean correct (user gave correct response)
     * @param e String with actioncommand of button pushed.
     */
    public void setCorrect(String e)
    {
            int answer = distractorIndices.get(Integer.parseInt(e) - 1);
            setUserAnswer(wordList.get(answer).getTranslation());
            if(answer == questionIndex)
            {
                    words.set(questionIndex, (words.get(questionIndex) + 1));
                    correct = true;
            }
            else
            {
                    if(repeat > 1)
                    {
                            if (words.get(answer) > 0)
                            {
                                    words.set(answer, (words.get(answer) - 1));
                            }
                    }
                    else
```

```
		{
			words.set(questionIndex, (words.get(questionIndex) + 1));

		}
		correct = false;
	}
}

/**
 * Set the user answer
 * @param userAnswer
 */
private void setUserAnswer(String userAnswer)
{
	this.userAnswer = userAnswer;
}

/**
 * Return the answer of the user
 * @return userAnswer
 */
public String getUserAnswer()
{
	return userAnswer;
}

/**
 * Returns array with the string value of the distractors (all four options)
 * @return distractors
 */
public ArrayList<String> getDistractors ()
{
	return distractors;
}

/**
 * Sets distractors, first empties distractor lists, then puts in index of correct answer,
     calls for indexDistractors and fills list with string value of distractor
 */
public void setDistractors ()
{
	distractorIndices.clear();
	distractors.clear();
	distractorIndices.add(questionIndex);

	distractorIndices = this.indexDistractors(distractorIndices, words);

	for(int i = 0; i < (distractorIndices.size()); i++)
	{
		distractors.add(wordList.get(distractorIndices.get(i)).getTranslation());
	}
}

/**
 * Sets current word. Makes array of possible words, then from that list picks one randomly.
 */
```

```java
public void setCurrentWord ()
{
        ArrayList<Integer> possibleQuestions = new ArrayList<Integer>();
        for (int i = 0; i < (words.size()); i++)
        {
                if(words.get(i) < repeat)
                {
                        possibleQuestions.add(i);
                }
        }

        int qIndex;
        if(possibleQuestions.size() == 1)
        {
                qIndex = 0;
        }
        else
        {
                qIndex = rand.nextInt(possibleQuestions.size());
        }
        questionIndex = possibleQuestions.get(qIndex);

        currentWord = wordList.get(questionIndex);
}

/**
 * Sets the previous word to the current word. After current word is modified, feedback can
     still be shown.
 */
protected void setPreviousWord()
{
        previousWord = this.getCurrentWord();
}

/**
 * Returns the previous word.
 * @return previousWord
 */
public Word getPreviousWord()
{
        return previousWord;
}

/**
 * Returns the current word
 * @return current word
 */
public Word getCurrentWord ()
{
        return currentWord;
}

/**
 * Run function that sets the previous word and if the user is not finished yet, sets the
     current word and the distractors
 */
```

```
        public void run() {
                this.setPreviousWord();

                if(!this.finished())
                {
                        this.setCurrentWord();
                        this.setDistractors();
                }
        }
}
```

B.6 InfoView

```
package aiThesis;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Toolkit;

import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

@SuppressWarnings("serial")
public class InfoView extends JPanel
{
	private Controller listener;

	// Components
	public JTextField number;
	public JTextField guess;
	public JComboBox name;

	/**
	 * Set the settings of the basis panel
	 * @param model
	 */
	public InfoView(Model model)
	{
		Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();

		this.setBackground(Color.WHITE);
		this.setPreferredSize(screenSize);
	}

	/**
	 * Set the actionlistener
	 * @param listener
	 */
	public void setListener(Controller listener)
	{
		this.listener = listener;
	}

	/**
	 * Puts all the introduction components on the panel.
	 */
	public void runIntro()
	{
		JLabel welkomMessage = new JLabel ("Hoi en welkom! Kies je naam en druk op start");
		welkomMessage.setBounds(100, 100, 400, 75);
		welkomMessage.setFont(new Font("Arial", Font.PLAIN, 20));
		this.add(welkomMessage);
```

```
        name = new JComboBox ();
        name.setBounds(100, 250, 500, 100);
        name.setFont(new Font("Arial", Font.PLAIN, 20));
        this.add(name);

        JButton start = new JButton ("Start");
        start.setBounds(1000, 500, 100, 75);
        start.setFont(new Font("Arial", Font.PLAIN, 20));
        start.addActionListener(listener);
        start.setActionCommand("start");
        this.add(start);
}

/**
 * Puts all the questionnaire components on the panel.
 */
public void runQuestionnaire()
{
        JLabel welkomMessage = new JLabel ("Goed gedaan! Vul alsjeblieft deze vragen in en
            druk daarna op OK");
        welkomMessage.setFont(new Font("Arial", Font.PLAIN, 20));
        welkomMessage.setBounds(100, 100, 600, 75);
        this.add(welkomMessage);

        JLabel q1Message = new JLabel ("Je hebt net 15 woordjes geleerd. Vul hieronder in
            hoeveel woordjes je op de toets denkt goed te hebben");
        q1Message.setFont(new Font("Arial", Font.PLAIN, 20));
        q1Message.setBounds(100, 150, 1000, 75);
        this.add(q1Message);

        guess = new JTextField ("... woordjes");
        guess.setFont(new Font("Arial", Font.PLAIN, 20));
        guess.setBounds(100, 250, 150, 75);
        guess.setActionCommand("guess");
        this.add(guess);

        JButton start = new JButton ("OK");
        start.setFont(new Font("Arial", Font.PLAIN, 20));
        start.setBounds(500, 500, 100, 75);
        start.addActionListener(listener);
        start.setActionCommand("proceed");
        this.add(start);

        repaint();
}

/**
 * Puts all the necessary components on the panel.
 */
public void runBetweenSessions()
{
        JLabel betweenMessage = new JLabel ("Je bent nu op de helft! We gaan nog een keer 15
            woordjes leren. Druk op start");
        betweenMessage.setFont(new Font("Arial", Font.PLAIN, 20));
        betweenMessage.setBounds(100, 150, 800, 75);
        this.add(betweenMessage);
        JButton next = new JButton ("Start");
        next.setFont(new Font("Arial", Font.PLAIN, 20));
        next.setBounds(1000, 500, 100, 75);
        next.addActionListener(listener);
        next.setActionCommand("nextExperiment");
        this.add(next);
    }

    /**
     * Puts the finished message on the panel.
     */
    public void runFinished()
    {
        JLabel endMessage = new JLabel ("Super gedaan! Je bent klaar. Heel erg bedankt voor
            je hulp.");
        endMessage.setFont(new Font("Arial", Font.PLAIN, 20));
        endMessage.setBounds(100, 100, 600, 75);
        this.add(endMessage);
    }
}
```

B.7 Kids

```java
package aiThesis;

public class Kids
{
        private String name;
        private int groupNumber;

        public Kids (String name, int groupNumber)
        {
                this.name = name;
                this.groupNumber = groupNumber;
        }

        /**
         * Returns the name of the kid
         * @return name
         */
        public String getName ()
        {
                return name;
        }

        /**
         * Returns the group number of the kid
         * @return groupNumber
         */
        public int getGroupNumber()
        {
                return groupNumber;
        }
}
```

B.8 MainView

```java
package aiThesis;

import java.awt.Dimension;
import java.awt.Toolkit;

import javax.swing.JFrame;

@SuppressWarnings("serial")
public class MainView extends JFrame
{
	private Model model;

	/**
	 * Sets title, screensize and other settings.
	 * @param model
	 */
	public MainView(Model model)
	{
		super ("Iris Monster Bachelor Experiment");

		Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();

		this.setPreferredSize(screenSize);
		this.setResizable(false);
		this.pack();
		this.setVisible(true);
		this.setDefaultCloseOperation(EXIT_ON_CLOSE);

		this.model = model;
	}

	/**
	 * Show start screen
	 * @param startView, information view that needs to be placed and called.
	 */
	public void showStart(InfoView startView)
	{
		this.add(startView);
		startView.runIntro();
		startView.setLayout(null);
		this.repaint();
	}

	/**
	 * Show an experiment
	 * @param infoView, this information view will be removed
	 * @param expView, this experiment view will be placed
	 */
	public void showExperiment(InfoView infoView, View expView)
	{
		this.remove(infoView);

		model.addObserver(expView);
		this.add(expView);
```

```
        expView.setLayout(null);
        this.setVisible(true);

        this.repaint();
}

/**
 * Show between sessions screen
 * @param infoView, this information view will be placed
 * @param expView, this experiment view will be removed
 */
public void showBetweenSessions(InfoView infoView, View expView)
{
        this.remove(expView);
        model.deleteObserver(expView);

        this.add(infoView);
        infoView.runBetweenSessions();
        infoView.setLayout(null);
        this.setVisible(true);

        this.repaint();
}

/**
 * Show finished screen
 * @param infoView, this information view will be placed
 * @param expView, this experiment view will be removed
 */
public void showFinished(InfoView infoView, View expView)
{
        this.remove(expView);
        model.deleteObserver(expView);

        this.add(infoView);
        infoView.runFinished();
        infoView.setLayout(null);
        this.setVisible(true);

        this.repaint();
}

/**
 * Show questionnaire screen
 * @param quesView, this information view will be placed
 * @param expView, this experiment view will be removed
 */
public void showQuestionnaire(InfoView quesView, View expView)
{
        this.remove(expView);
        model.deleteObserver(expView);

        this.add(quesView);
        quesView.setLayout(null);
        quesView.runQuestionnaire();
        this.setVisible(true);
```

```
                this.repaint();
        }
}
```

B.9 Model

```
package aiThesis;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Date;
import java.util.Observable;

public class Model extends Observable
{
       // Create all variables.
       private Exclusion exclExperiment;          // Exclusion experiment for learning the words
       private Control cntrExperiment;            // Control experiment for learning the words
       private Control exclTest;                          // Control experiment for testing
           performance in exclusion condition
       private Control cntrTest;                          // Control experiment for testing
           performance in control condition
       private Experiment currentExperiment; // The current experiment (this changes during the
           session)
       private Experiment currentTest;            // The current test (this changes during the
           session)
       private static ArrayList<Word> words; // List of words
       private static ArrayList<Kids> kids; // List of kids
       private int groupNumber;                           // Groupnumber of subject (see verslag
           for explanation on which group is which number)
       private MainView mainview;                         // Mainview for general screen,
           initiates the other views as well.
       private int counter;                               // Counter to show number of trial in
           output file
       private View expView;                              // Experiment view
       private InfoView infoView;                         // View for questionnaire, welcome and
           thank you page.
       private Controller controller;             // Controller
       private FileWriter writer;                         // File writer to write to files
       private File file;                                         // File
       private int repeatTraining;                        // Amount of times the word has to be
           recognized correctly for training
       private int repeatTesting;                         // Amount of times the word has to be
           recognized correctly for testing
       private int feedbackTime;                          // Amount of milliseconds the feedback
           will be shown
       private int process;                               // 1 = First study-phase, 2 = First
           test-phase, 3 = Second study-phase, 4 = Second test-phase

       /**
        * Constructor of Model sets parameters, sets a few variables, reads file with kids and the
            file with words. Starts up first view
        * @param repeatTraining Amount of times the word has to be recognized correctly for
            training
        * @param repeatTesting Amount of times the word has to be recognized correctly for testing
        * @param feedbackTime Amount of milliseconds the feedback will be shown
```

```
 */
public Model(int repeatTraining, int repeatTesting, int feedbackTime)
{
       // Settings

       this.repeatTraining = repeatTraining;
       this.repeatTesting = repeatTesting;
       this.feedbackTime = feedbackTime;
       process = 0;
       controller = new Controller(this);
       groupNumber = 0;
       words = new ArrayList<Word>();
       kids = new ArrayList<Kids>();

       // Read file with names, file with words and creates output file
       readNameFile();
       readFile();
       createFile();

       // Start up screen
       mainview = new MainView(this);
       infoView = new InfoView(this);
       infoView.setListener(controller);

       mainview.showStart(infoView);

       for(int i = 0; i < kids.size(); i++) // Add kids names to combobox in start-up view
       {
              infoView.name.addItem(kids.get(i).getName());

       }
}

/**
 * Create experiments. So first split the wordlist in 2 parts and then set the experiments
     with the correct wordlist and repetition
 */
private void createExperiments()
{
       ArrayList<Word> wordsPart1 = new ArrayList<Word>();
       ArrayList<Word> wordsPart2 = new ArrayList<Word>();

       wordsPart1.addAll(words.subList(0,15));
       wordsPart2.addAll(words.subList(15,30));

       if(groupNumber == 1 || groupNumber == 3)
       {
              exclExperiment = new Exclusion(wordsPart1, repeatTraining);
              exclTest = new Control(wordsPart1, repeatTesting);
              cntrExperiment = new Control(wordsPart2, repeatTraining);
              cntrTest = new Control(wordsPart2, repeatTesting);

       }
       else
       {
              exclExperiment = new Exclusion(wordsPart2, repeatTraining);
```

```
                exclTest = new Control(wordsPart2, repeatTesting);
                cntrExperiment = new Control(wordsPart1, repeatTraining);
                cntrTest = new Control(wordsPart1, repeatTesting);
        }
}

/**
 * Run training. This sets up an experiment View with an actionListener and starts the
     training.
 */
private void runTraining()
{
        process++;
        counter = 0;
        currentExperiment.run();
        expView = new View(this);
        expView.setListener(controller);
        expView.setActionListener();
        expView.setFeedback(true);

        mainview.showExperiment(infoView, expView);

        expView.showNextWord(currentExperiment);
}

/**
 * Run Testing. The same as run training, but now a test will be performed.
 */
private void runTesting()
{
        process++;
        counter = 0;

        currentExperiment = currentTest;
        currentExperiment.run();
        expView = new View(this);
        expView.setListener(controller);
        expView.setActionListener();
        expView.setFeedback(false);

        mainview.showExperiment(infoView, expView);

        expView.showNextWord(currentExperiment);
}

/**
 * Run Questionnaire. A new infoView is created, an actionListener is set and mainview is
     called to make sure the right screen is presented.
 */
private void runQuestionnaire()
{
        infoView = new InfoView(this);
        infoView.setListener(controller);
        mainview.showQuestionnaire(infoView, expView);
}
```

```
/**
 * Run between sessions. The same as run questionnaire, but the betweenSessions screen
     shows another textmessage.
 */
private void runBetweenSessions()
{
       infoView = new InfoView(this);
       infoView.setListener(controller);
       mainview.showBetweenSessions(infoView, expView);
}

/**
 * Run finished. The same as the run questionnaire, but again another message is displayed.
 */
private void runFinished()
{
       infoView = new InfoView(this);
       infoView.setListener(controller);
       mainview.showFinished(infoView, expView);
}

/**
 * Called by an experiment view, to set the feedback Time.
 * @return feedbackTime
 */
public int getFeedbackTime()
{
       return feedbackTime;
}

/**
 * Called by an experiment view, to get information from the experiment (like which word is
     next)
 * @return the current Experiment
 */
public Experiment getCurrentExperiment()
{
       return currentExperiment;
}

/**
 * This function is called by controller when a button is pressed and calls for next steps.
 * @param e the actioncommand
 */
public void processAction (String e)
{
       if(e.length() < 2)                                                      // Case
           experiment response button was pushed
       {
              currentExperiment.setCorrect(e);                  // Set correct of current
                  experiment
              processAnswer();                                              // Write the
                  answer in the output file

              if(currentExperiment.finished())                  // Current experiment is
                  finished, then based on process call for next move.
```

```
                {
                        if(process == 1 || process == 3)
                                {
                                        runQuestionnaire();
                                }
                        else if(process == 2)
                        {
                                runBetweenSessions();
                        }
                        else if(process == 4)
                        {
                                runFinished();
                        }
                }
        else                                                                            //
            Current experiment is not finished, so run the current experiment and
            set changed
        {
                currentExperiment.run();
                setChanged();
                notifyObservers();
        }

}
if(e.equals("start"))                                               // Start means the
    start button is pushed, selected name can be processed and training started.
{
        processName();
        runTraining();
}
if(e.equals("proceed"))                                             // Proceed means
    questionnaire is filled out, so write answer to output file and start testing
{
        processQuestionnaire();
        runTesting();
}
if(e.equals("nextExperiment"))                              // Next experiment means
    subject is done with first session and can start with the other training
{
        if(groupNumber == 1 || groupNumber == 2)
        {
                currentExperiment = cntrExperiment;
                currentTest = cntrTest;
        }
        if(groupNumber == 3 || groupNumber == 4)
        {
                currentExperiment = exclExperiment;
                currentTest = exclTest;
        }
        runTraining();
}
}

/**
 * Process Answer. This function created the line with the counter, presented word,
     distractors and chosen answer. Then calls writeToFile with this line.
 */
private void processAnswer()
{
counter++;
String count = String.valueOf(counter);
String answerLine = new String();
String currentWord = currentExperiment.getCurrentWord().getWord();
String distractors = currentExperiment.getDistractors().toString();
String userAnswer = currentExperiment.getUserAnswer();
answerLine = count.concat(". ".concat(currentWord).concat("
    ").concat(distractors).concat(" ").concat(userAnswer));

try {
        writeToFile(answerLine);
} catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
}
}

/**
 * Process Name. Figures out which kid the subject is to get the groupNumber. Then writes
     both of these to the file. Then creates the experiments and with the groupNumber sets
     which experiment will be done first.
 */
private void processName()
{
int indexKid = infoView.name.getSelectedIndex();
Kids selectedKid = kids.get(indexKid);

String name = selectedKid.getName();
groupNumber = selectedKid.getGroupNumber();

try {
        writeToFile("name: ".concat(name));
} catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
}

try {
        writeToFile("group: ".concat(String.valueOf(groupNumber)));
} catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
}

createExperiments();


if(groupNumber == 1 || groupNumber == 2)
{
        currentExperiment = exclExperiment;
        currentTest = exclTest;
}
if(groupNumber == 3 || groupNumber == 4)
{
        currentExperiment = cntrExperiment;
        currentTest = cntrTest;
}
}

/**
 * Create a new output file with the current time as title
 */
private void createFile()
{
Date date = new Date();
String fileName = String.valueOf(date.getTime());
file = new File(fileName);
try {
        file.createNewFile();
} catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
}
}

/**
 * Writes a line to the output file
 * @param line that needs to be logged
 * @throws IOException
 */
private void writeToFile(String line) throws IOException
{
writer = new FileWriter(file,true);
writer.write(line.concat("\n"));
writer.flush();
writer.close();
}

/**
 * Process the answer on the questionnaire by simply making a nice line of the answer and
     writing it to a file.
 */
private void processQuestionnaire()
{
String answer = "Question 1 ".concat(infoView.guess.getText());

try {
        writeToFile(answer);
} catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
}
}

/**
 * Reads wordlist.txt and sets the words.
 */
private static void readFile()
{
		try {
			File file = new File("wordlist.txt");
			FileReader fileReader = new FileReader(file);
			BufferedReader bufferedReader = new BufferedReader(fileReader);
			String line;
			while ((line = bufferedReader.readLine()) != null) {
				String[] parts = line.split(" = ");
				String nl = parts[1];
				String eng = parts[0];
				Word word = new Word(nl, eng);
				words.add(word);
			}
			fileReader.close();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	/**
	 * Reads from kinderen.txt and sets kids.
	 */
	private static void readNameFile()
	{
		try {
			File file = new File("kinderen.txt");
			FileReader fileReader = new FileReader(file);
			BufferedReader bufferedReader = new BufferedReader(fileReader);
			String line;
			while ((line = bufferedReader.readLine()) != null) {
				String[] parts = line.split(" = ");
				String name = parts[0];
				int groupNumber = Integer.valueOf(parts[1]);
				Kids kid = new Kids(name, groupNumber);
				kids.add(kid);
			}
			fileReader.close();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}


}
```

B.10 View

```
package aiThesis;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Toolkit;


import java.util.Observable;
import java.util.Observer;
import java.util.Timer;
import java.util.TimerTask;

import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

@SuppressWarnings("serial")
public class View extends JPanel implements Observer
{
	// Components
	private JLabel question = new JLabel ("Voorbeeld");
	private JButton answer1 = new JButton ("answer1");
	private JButton answer2 = new JButton ("answer2");
	private JButton answer3 = new JButton ("answer3");
	private JButton answer4 = new JButton ("answer4");
	private JTextField feedback = new JTextField (" ");

	// Other variables
	private Model model;
	private Controller listener;
	private boolean isTraining;				// Is true when subject is learning (then
	    feedback must be shown)
	private Timer timer;
	private int feedbackTime;

	/**
	 * Set settings, shape components and add them to panel
	 * @param model
	 */
	public View (Model model)
	{
		this.model = model;

		Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
		isTraining = false;
		feedbackTime = model.getFeedbackTime();

		this.setBackground(Color.WHITE);
		this.setPreferredSize(screenSize);

		question.setBounds(200, 100, 400, 75);
		question.setFont(new Font("Arial", Font.PLAIN, 25));
```

```
        this.add(question);

        answer1.setBounds(200, 300, 300, 75);
        answer1.setFont(new Font("Arial", Font.PLAIN, 25));
        this.add(answer1);

        answer2.setBounds(200, 400, 300, 75);
        answer2.setFont(new Font("Arial", Font.PLAIN, 25));
        this.add(answer2);

        answer3.setBounds(600, 300, 300, 75);
        answer3.setFont(new Font("Arial", Font.PLAIN, 25));
        this.add(answer3);


        answer4.setBounds(600, 400, 300, 75);
        answer4.setFont(new Font("Arial", Font.PLAIN, 25));
        this.add(answer4);

        feedback.setBounds(200, 600, 700, 75);
        feedback.setFont(new Font("Arial", Font.PLAIN, 25));

        this.add(feedback);

        this.repaint();

}

/**
 * Set the actionlistener to the components, this can be done after the controller is set
     as the actionListener of this view.
 */
public void setActionListener()
{
        answer1.addActionListener(listener);
        answer1.setActionCommand("1");

        answer2.addActionListener(listener);
        answer2.setActionCommand("2");

        answer3.addActionListener(listener);
        answer3.setActionCommand("3");

        answer4.addActionListener(listener);
        answer4.setActionCommand("4");
}

/**
 * Called when model calls setChanged() and notifyObservers(). It is a messy function, see
     comments below
 */
@Override
public void update(Observable o, Object arg1)
{
        answer1.setFocusPainted(true);
        answer2.setFocusPainted(true);
        answer3.setFocusPainted(true);
        answer4.setFocusPainted(true);

        model = (Model) o;
        timer = new Timer();

        if(isTraining)                         // If true than feedback can be shown, otherwise
            it's a test and no feedback must be shown.
        {
                giveFeedback(model.getCurrentExperiment());
        }


        if(!model.getCurrentExperiment().finished()) // If the experiment is not finished
            first wait the feedbackTime and then perform run (show new word)
        {
                timer.schedule(new TimerTask()
                {
                        public void run ()
                        {
                                showNextWord(model.getCurrentExperiment());
                                feedback.setText(" ");
                                feedback.setBackground(Color.WHITE);
                                timer.cancel();
                                answer1.setFocusPainted(false);
                                answer2.setFocusPainted(false);
                                answer3.setFocusPainted(false);
                                answer4.setFocusPainted(false);
                        }
                }, feedbackTime);
        }
}

/**
 * Set isTraining (or feedback)
 * @param isTraining true in case of training, false in case of testing
 */
public void setFeedback(boolean isTraining)
{
        this.isTraining = isTraining;
}

/**
 * Set the actionListener for this view
 * @param listener
 */
public void setListener(Controller listener)
{
        this.listener = listener;
}

/**
 * Show feedback (change color of feedback frame and set the text).
 * @param experiment
 */
private void giveFeedback(Experiment experiment)
```

```
        {
                if(experiment.getCorrect())
                {
                        feedback.setBackground(Color.GREEN);
                        feedback.setText("Goed zo
                            ".concat(experiment.getPreviousWord().getWord()).concat(" is
                            ").concat(experiment.getPreviousWord().getTranslation()));
                }
                else
                {
                        feedback.setBackground(Color.RED);
                        feedback.setText("Oeps
                            ".concat(experiment.getPreviousWord().getWord()).concat(" is
                            ").concat(experiment.getPreviousWord().getTranslation()));
                }

        }

        /**
         * Is called by model to show the first word, then only called by this update function.
         * @param experiment
         */
        public void showNextWord(Experiment experiment)
        {
                question.setText("Wat is
                    ".concat(experiment.getCurrentWord().getWord()).concat("?"));
                answer1.setText(experiment.getDistractors().get(0));
                answer2.setText(experiment.getDistractors().get(1));
                answer3.setText(experiment.getDistractors().get(2));
                answer4.setText(experiment.getDistractors().get(3));

        }
}
```

B.11 Word

```
package aiThesis;

public class Word
{
	private String nederlands;
	private String engels;

	public Word(String nederlands, String engels)
	{
		this.nederlands = nederlands;
		this.engels = engels;
	}

	/**
	 * Return the Dutch word
	 * @return nederlands
	 */
	public String getWord ()
	{
		return nederlands;
	}

	/**
	 * Return the English translation
	 * @return engels
	 */
	public String getTranslation()
	{
		return engels;
	}
}
```

References

[1] K. Apel, J. A. Wolter, and J. J. Masterson. Effects of phonotactic and orthotactic probabilities during fast mapping on 5-year-olds' learning to spell. *Developmental Neuropsychology*, 29(1):21–42, 2006.

[2] J. Kaminski, J. Call, and J. Fischer. Word learning in a domestic dog: Evidence for “fast mapping”. *Science*, 304(5677):1682–1683, 2004.

[3] W. J. McIlvane and T. Stoddard. Acquisition of matching-to-sample performances in severe retardation: Learning by exclusion. *Journal of Mental Deficiency Research*, 25(1):33–48, March 1981.

[4] J. M. O'malley and A.U. Chamot. *Learning strategies in second language acquisition.* Cambridge University Press, 1990.

[5] P. Pimsleur. A memory schedule. *The Modern Language Journal*, 51(2):73–75, February 1967.

[6] M. M. Van Gogh, J. M. McQueen, and L. Verhoeven. Learning phonologically specific new words fosters rhyme awareness in dutch preliterate children. *In Press*, 2014.

[7] K. M. Wilkinson, C. Rosenquist, and W. J. McIlvane. Exclusion learning and emergent symbolic category formation in individuals with severe language impairments and intellectual disabilities. *The Psychological Record*, 59(2), 2009.