A Visual Method for Teaching Grammatical Concepts to Primary and Secondary Schoolers: An Interactive Sentence Assembly Tool on the Internet *

Master's Thesis by
Camiel van Breugel
http://www.wi.leidenuniv.nl/home/cvbreuge/
cvbreuge@cs.leidenuniv.nl
Department of Computer Science
Leiden University, Netherlands

September 1, 1998

* http://www.wi.leidenuniv.nl/home/cvbreuge/bin/spookjes.html

Contents

Preface

There are several serious problems with conventional grammar instruction for young children. The biggest problem of all is that grammatical concepts are often too abstract for them. Furthermore, conventional grammar instruction proves to be de-motivating and causes many children to lose interest in language education. In traditional grammar instruction the children's primary task consists mainly of the rather monotonous exercise of analyzing sentences. One concept that might help is that of *edutainment*, which is, not very surprisingly, the combination of *education* and *entertainment*. Edutainment might turn learning into a more attractive process and draw the children's attention more easily than conventional educational methods. What I propose is an alternative approach by which children develop grammar skills by *building sentences*, i.e. constructing them out of several different building blocks which results in complete sentence diagrams. My work resulted in the design and implementation of a prototype of such a computer application in the form of an interactive grammar puzzle. The big advantages in this design are that abstract names for grammatical concepts can be replaced by visual shapes like the pieces of a puzzle, and that the underlying cognitive models of the grammar visualization method are very suitable as building blocks. These models were designed by Kempen [11] and are explained briefly in this document. To maximize accessibility I try to exploit the current rise in multimedia possibilities and availability by presenting the grammar exercises by means of an interactive computer puzzle that will be accessible through the internet.

1 Introduction

This document describes the prototype that I designed and implemented of a computer application for computer assisted grammar learning and training for children in the higher grades of primary education and the lower grades of secondary education. It forms part of my graduation project that was done at the Department of Computer Science at Leiden University in cooperation with the Experimental and Theoretical Psychology Unit of the Department of Psychology at Leiden University. It deals with the main psycholinguistic and educational aspects of this project, the prototype's design and design philosophy and the major aspects of the implementation. It is concluded by an overview of the possibilities of this system and some ideas for future development. I will start this paper with a short overview of the current state of grammar instruction in section 2. Section 3 is a short introduction to initial grammar visualization by means of sentence diagrams. Next, in section 4, I describe in short a psychological model of cognitive structures named lexical frames that can serve as building blocks for constructing these sentence diagrams. Section 5 deals with the interface design of a computer game for grammar instruction for children that is based on these lexical frames. The actual design and implementation of the system and the main data structures and algorithms of the computer application are described in section 6. This paper ends with a short summary and some ideas for further progress in section 7.

2 Initial Grammar Instruction

Language is certainly the most powerful medium that we humans possess to express ourselves. Even if we wanted to we would never be able to stop joking, chatting, tittle-tattling, debating and discussing with others. We write things down in order to preserve the best (or worst) of our thoughts and studies, to send letters to people at distance or just to keep notes. More rambling souls may write poetry, dramatic plays or even fiction. Serving all kind of purposes spoken and written language play a very central role in the life of all speakers, listeners, readers and writers. It is the very foundation of our culture. Ludwig Wittgenstein has put it this way: *The limits of language are the limits of my world.* For innumerable things such as convincing, selling, keeping acquaintances, explaining, tale telling and many other things it is a huge advantage for one to be an expert user of his or her language. This is why during basic education children should be drilled extensively in speaking, listening, orthography, reading and writing to try and make them experienced language users. It would be more than an educational goal alone to teach students to comprehend and use their language fully and experience all of its rich possibilities. However, as one observes the numerous errors and poor quality that students produce in writing, one is almost forced to believe that this is not possible for the larger part of the students. They lack the necessary insights and fail to see what their mistakes are and why. The Dutch language contains lots of word pairs that are pronounced the same while their orthography differs and depends on their role in the sentence. This is probably the main source of misspelling in Dutch [11]. This problem can only be tackled by some form of grammar instruction that unveils the structure of a sentence and the relations between the words. But not only writing skills benefit by grammar instruction. Reading does require a lot of grammatical experience too.

Unfortunately, it cannot be said that grammar instruction has been very successful in the past: There is often poor feedback and it can be de-motivating for both teachers and students. The lack of enough time at schools for teaching grammar to pupils is not the only reason for this. To begin with there is the pitiful delimitation that grammar instruction in the sense of simply learning the rules by heart does not yield the benefits wished for. Children manage to learn a language without consciously knowing the rules and imposing these rules on them only tends to confuse them. With this in mind many methods were developed for the Dutch language over the years, which I shall now and then refer to as *traditional grammar instruction*.

2.1 Traditional grammar instruction

For Dutch grammar instruction there are a number of several commonly applied methods such as *Taal-actief*, *Montessori*, *Grammatica in balans*, and *Taaltoren*. What most of these have in common is that the developing language user is provided with exercises that concentrate on the orthography of the finite verb, analysis of the sentence structure, the parts of speech, while the difficulty level is increased gradually. There is a lot more to say about them of course but I shall first give the most important aspects [14] of these methods. Grammar instruction often follows the path of confrontation with a new rule followed by its application to concrete sentences. There is no direct connection with the other forms of language instruction like training in spelling, writing, speaking, and listening, but it takes place in separate courses. The main concept is learning by training exercises, not by the exchange of ideas or much discussion. They deal with rules for classification of *constituents (predicate, subject, direct object, indirect object)*, *parts of speech (noun, adjective, article)* and *sentence types (affirmative, interrogative, imperative)*. And they touch upon dealing with rules for recognition and production of *conjugations and inflections (tense, active/passive voice, 1st/2nd/3rd person, imperative)*, *adjective (comparative form and inflectional form)*, *noun (singular/plural, case)*, *analysis and punctuation marks (main/subordinate clause, point, comma, exclamation mark, interrogation-mark)* and *stylistic constructions (indirect/direct speech, clause length, choice of words, references)*. The average pupil subjected to this approach should probably be able to improve his performances in comprehensive reading and logical thinking a great deal. But there are still a number of important drawbacks to overcome. Apart from this the question rises whether traditional grammar instruction still meets the modern educational standards.

2.2 Drawbacks of traditional grammar instruction

Evers [7] mentions some major disadvantages of traditional grammar instruction like that it takes away children's natural wonder about speech and language, employs unrealistic use of language in the exercises, and provides knowledge of grammar but not directly the understanding of everyday speech. Her main conclusion is that traditional grammar instruction is too abstract and too difficult for young children so that tricks are needed, that it is insufficiently integrated in the education of reading and that it is often de-motivating for children. Kempen [11] mentions low level of motivation in pupils and teachers, little time for practising, late and fragmented feedback, insufficient *scaffolding* and the little room for explanatory learning. The cause of all the troubles that children encounter when dealing with grammar rules lies mainly in their inability to think abstractly. Roughly speaking it can be said that children aged under twelve can only perform concrete operations. It is not until the age of about eleven years that they enter the abstract phase. The main problem of traditional grammar instruction is that it prescribes rules that are not concrete enough. And a rule that is too abstract to be fully understood is not likely to mean a lot to one. No wonder that making those exercises is not regularly regarded by children as a very interesting occupation. It is simply not the natural way for young children to learn by starting with a sequence of rules. They have a completely different learning mechanism of playful and effortless discovery and challenge. They are stimulated a lot when they are given the chance to compete with others. However later on in their development this mechanism seems to be replaced more and more by explicit and logical reasoning. All this forms a strong argument that it would be better if somehow the young students were guided to discover the rules themselves. What is needed most to accomplish this is having easier exercises with the same effect with immediate result and feedback. Computer assisted language learning probably might offer just the necessary possibilities. Doing exercises with the good old-fashioned pencil and paper still might be useful of course, but this activity takes a lot of valuable time and a different kind of effort which can distract from the intended learning. And it takes a lot of the teacher's time to correct the worked out exercises. This time can be spent more efficiently when that task is taken over by some form of computer assisted learning. One of the promising things about computer assisted learning is that it allows *scaffolding*. This is a concept by which the learner is stimulated and guided to follow the desired behavioral patterns repeatedly, which hopefully results in the correct behaviour. For practical use in grammar instruction, scaffolding should be used to keep the children busy with the construction of sentences out of a set of suitable building blocks that are limited in their use only by the nature of their grammatical properties. This has the enormous advantage of immediate feedback which would normally seem impossible for a teacher with twenty pupils or more. In this project I try to exploit modern multimedia technology to support the exercises in an interactive setting as a new effort in this field. The exercises can be presented in the form of a puzzle or as a work of construction. The element of game is a very important and sometimes underestimated aspect of the children's learning mechanisms.

3 Visualizing grammar

The first thing to be done when making abstract terms more concrete is applying visualization. There are several aspects of visualization in general that are worth noting with respect to computer assisted learning. To begin with, it can be used as an analogical representation that can serve as an external memory and provide contextual clues. Such an analogical representation is often strong in combination with a direct manipulation interface [17]. Structures become more concrete because visualization builds an imaginative bridge between task and concept. Furthermore, it can be used to draw attention, especially when it looks funny and it is not disturbing or confusing. In the case of grammar instruction, the sentence structure can be unveiled at once or in steps, which hopefully improves comprehension. Next, I shall describe the sentence diagrams in which grammatical relations of sentences usually are expressed.

3.1 Sentence diagrams

The common method for visualizing the structure of a sentence is drawing a sentence diagram. Generally there are two kinds of sentence diagrams. In the first one, every single word is to be labelled with the proper part-of-speech label. In the second sentence diagram the syntactic relations between the words and word groups in the sentence are represented in a diagram as a hierarchical tree structure. This distinction is known as *part-of-speech tagging* versus *syntactic parsing* (in Dutch: *taalkundig ontleden* versus *redekundig ontleden*). An example of these sentence diagrams is illustrated for the Dutch sentence *De schildpad versloeg de haas (English:* The turtle has beaten the hare*)* in figures 1 and 2. The

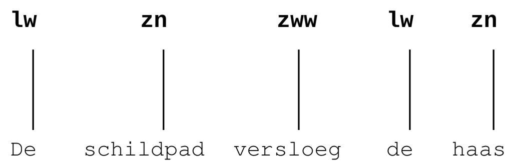

Figure 1: Part-of-speech sentence diagram of *The turtle has beaten the hare* in Dutch

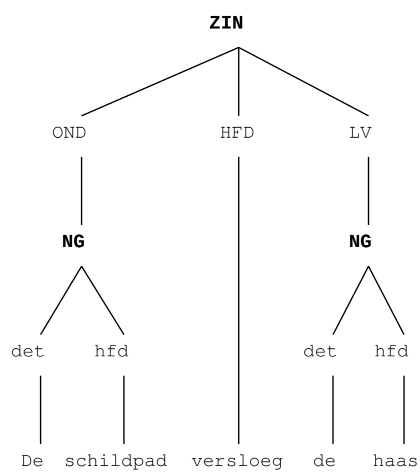

Figure 2: Syntactic parse tree of *The turtle has beaten the hare* in Dutch

labels, in Dutch, indicate the various grammatic terms [1]. Parts-of-speech labels and phrasal category labels are shown in bold font, whereas the identifiers and the syntactic function labels are printed with plain characters. I use capitals

[1]In Appendix A a list of abbreviations can be found with the English translations

only for syntactic function labels in the second row and for all phrasal category labels. As one can see the parse tree in figure 2 is slightly different than one would remember or expect to see with his own education in mind. This is a so-called *head-driven* parse tree. It is like the sentence diagram of old but now the constituents are subdivided into more detailed subtrees. The leading member of such a subtree is called *hoofd (English:* head*)*. In the sentence of figure 2, for instance, the main verb *versloeg* plays the role of head of the sentence, and both nouns *schildpad* and *haas* are *heads* in a noun phrase. The other members of a subtree also play a role in that subtree. E.g. article *de* plays the role of determiner in both noun groups. Most grammar instruction methods let the students perform some sort of part-of-speech tagging and syntactic parsing. Their task in the latter case is to analyse sentences by finding the subject, the finite verb, the direct object, the indirect object, etcetera. For the majority of the students it can be said that their skills in *part-of-speech tagging* outrun those in *syntactic parsing*. This could well be explained by the assumption that the classification of words with part-of-speech tagging is more concrete and therefore easier than the rather abstract definitions and large number of possible phrasal categories with syntactic parsing.

3.2 Examples of sentence diagrams

To introduce the part-of-speech labels I have provided two sentences, *ik vraag me af waar die is* in figure 3 and *Er wordt me door niemand verteld dat die en haar twee jonge zussen elkaar vaak besproeien in de tuin* in figure 4. Together these sentences contain most part-of-speech labels that exist. These two sentences

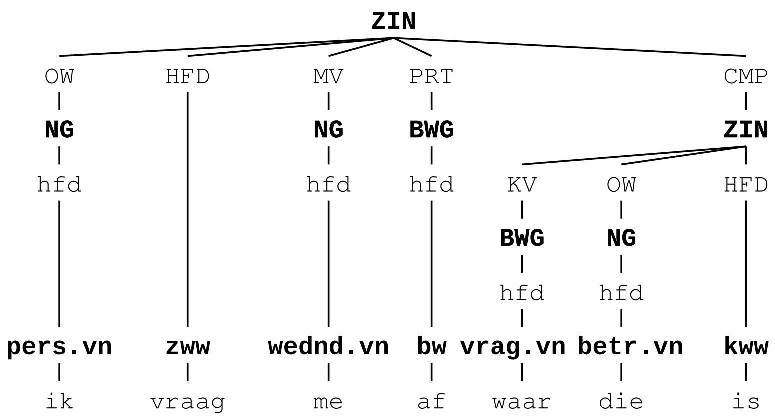

Figure 3: Example of a sentence diagram in Dutch ‘ik vraag me af waar die is’

already use a lot of segments. Figure 5 shows the most important segments that are needed for Dutch grammar.

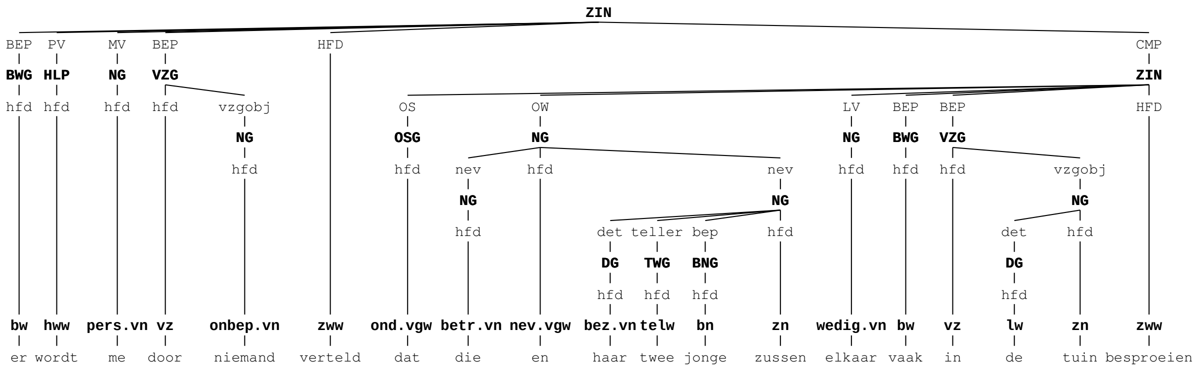

Figure 4: Example sentence diagram in Dutch

4 What goes on within the brain of a language user

Among other things, Kempen is studying the *ins* and *outs* of sentence perception and production for years now, seeking explanations for slips of the tongue and slips of the pen by language users. It shall always be the case that people create ill-formed sentences and misspell words. And some perfectly correct sentences are in some ways so nasty that one can almost predict the mistakes a reader is going to make when reading them aloud. Another example is every year's Dutch event of the 'Nationaal Dictee' where a TV-presentator reads aloud a dictation and a selected group of Dutch language users tries to write it down attempting not to miss any. It has to be said that this dictation is often bulking of uncommon and difficult words, not really every day speech.

Many cognitive models of linguistic structures in language users are being developed in an attempt to explain human language behaviour. Maybe the simplest way to look at these structures is to see them as little elements that contain conceptual, morpho-phonological or syntactic information that are activated or connected when needed. One theoretical model of brain structures at the syntactic level is based on *lexical frames* (or *word frames*). Kempen has composed these lexical frames in his earlier research [9]. They are small elementary grammatical units that are modeled after the cognitive structures that might well exist somehow in the brain of a language user. In the next section I will give a short overview of the design and the possibilities of these lexical frames.

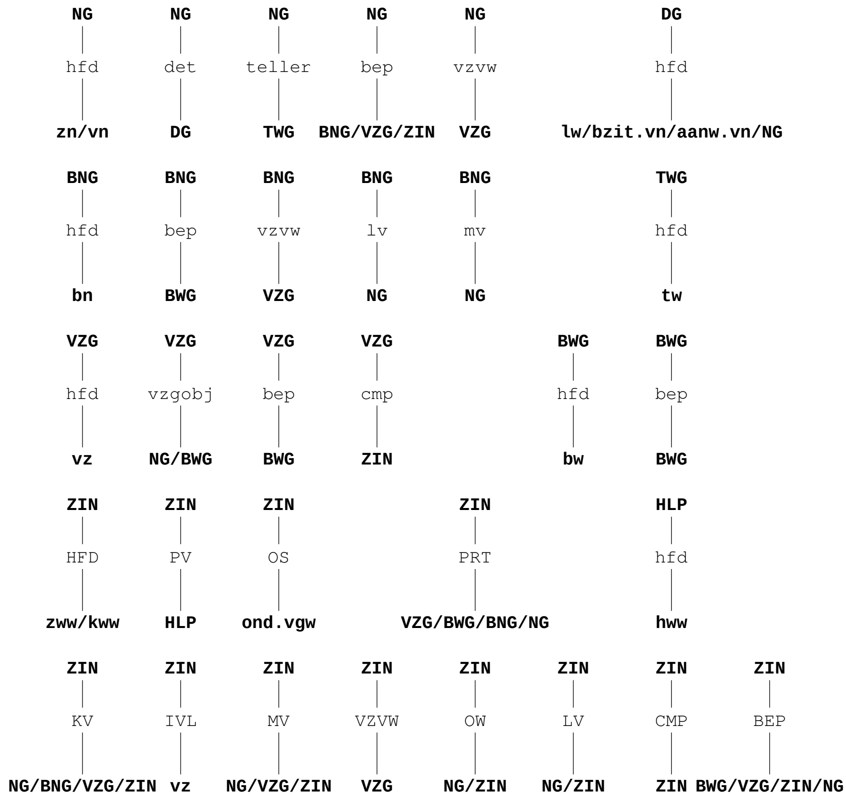

Figure 5: The most important segments for Dutch grammar

4.1 Lexical frames

Kempen utilizes the hierarchical tree structures mentioned above to visualize the grammatical concepts within a sentence. He distinguishes twenty different parts of speech, five different word groups and fifteen syntactic functions. These are not very different from those generally used in schools. Word groups, but also single parts of speech, may perform one of the syntactic functions. Furthermore, there are seventeen different syntactic properties that belong to some of the nodes. A complete table of their names and translation in English can be found in Appendix A. An important feature of the sentence diagrams proposed by Kempen is that the two sentence diagrams in figures 1 and 2 are combined into one now. The part-of-speech labels are connected to the leaf nodes of the syntactic parse tree as is shown in figure 6. The structure of this two-in-one sentence diagram shows a repeating pattern in the vertical direction. First comes a phrasal category ZIN (sentence), then the role (or syntactic function) of each

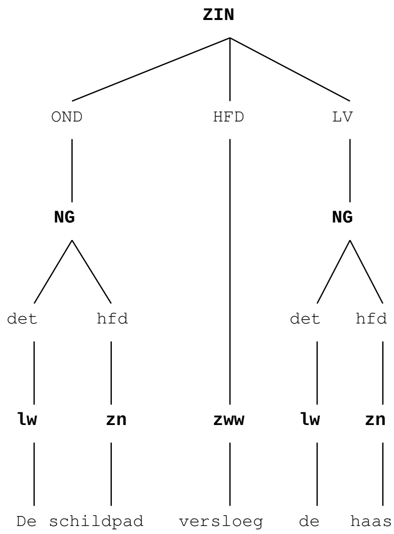

Figure 6: Combined sentence diagram of the parse tree and the part-of-speech labels

subtree, such as OND (subject) and LV (direct object), and then again another phrasal category of a subtree followed by a syntactic function again. The two nodes at the bottom of each subtree finally contain only the part-of-speech label and the word label. This pattern makes it possible to split the tree into small segments. Every sequence *phrasal category—syntactic function—phrasal category/part-of-speech label* is one segment. The top label is called the *root*, the second one *function* and the third one *foot*. Figure 7 shows the segments for the sentence *The turtle has beaten the hare*. Two or more segments with the same root label can be joined back again by a horizontal link operation as shown in figure 8. A number of segments that are joined in this way, with precisely one segment playing the role of head, forms the basis of one *lexical frame*. Every segment contains also syntactic properties like case, person, etc. that belong to that segment. These are stored in an array named the *feature matrix*. The lexical frames at the bottom of a parse tree have an additional row labeled by the words of the sentence. So, a lexical frame may have four rows with labels: a phrasal category, called the *root*, next a number of n grammatical functions, each with a part-of-speech label and a word label attached to it. When the third label is another phrasal category there are only three rows of labels. Lexical frames can be linked by connecting a root of one lexical frame to a foot node of another lexical frame. This process of linking is called *unification*. Figure 9 shows an abstract example of such an unification. Unification of two lexical frames can only be done when the root label of the first lexical frame is the same as the foot label of the second lexical frame and these labels where not connected yet. This limitation is needed for making correct sentence diagrams.

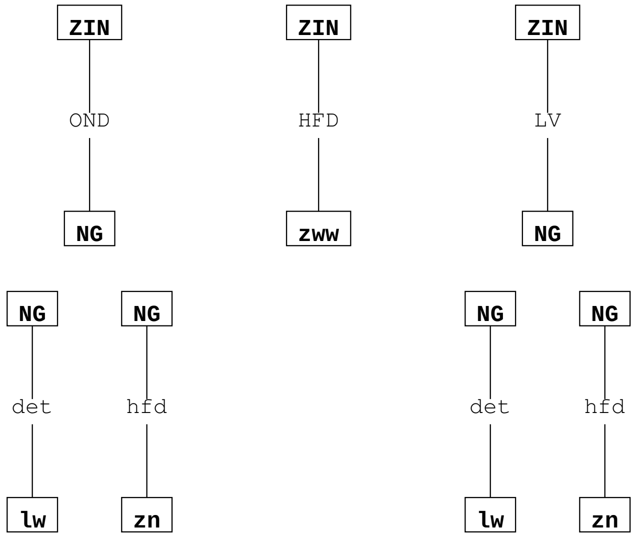

Figure 7: The segments of *The turtle has beaten the hare*

Figure 10 shows an example of some lexical frames that can be used to create a sentence diagram. Figure 11 shows how these lexical frames must be unified to obtain the sentence *'De toehoorders leunden nog meer naar voren'*. The resulting sentence diagram is shown in figure 12. To ensure that only grammatically correct sentence diagrams can be built there are yet more conditions that must be satisfied before unification is allowed. This is were the syntactic properties come into the story. For instance, the plural subject *de nachten (the nights)* cannot be combined with the singular form of the finite verb *verdwijnt (vanishes)*, and this also applies to the combination of *de nacht (the night)* and *verdwijnen (vanish)*, although the foot and root labels involved are both *NG*. But *de nachten verdwijnen* and *de nacht verdwijnt* are completely legal combinations. The same holds for the 3rd person noun *James* and first person finite verb *cook*. In general, unification can only be grammatically valid if and only if when the syntactic properties in the involved feature matrices *match*. To put it simple: matching means the intersections of the syntactic properties belonging to the to-be-unified segments do not yield the empty set.

4.2 Using lexical frames for a new type of exercise

Grammar instruction may become more successful with a new type of exercise enabling students to perform the same task as the brain does while putting words together during the formation of any ordinary sentence. Students should exercise in constructing sentence diagrams out of elementary grammatical build-

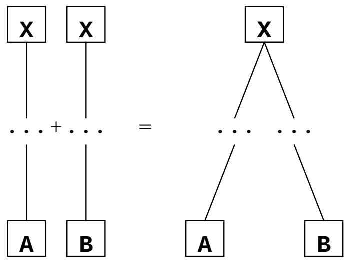

Figure 8: Illustration of the horizontal linking process of two segments

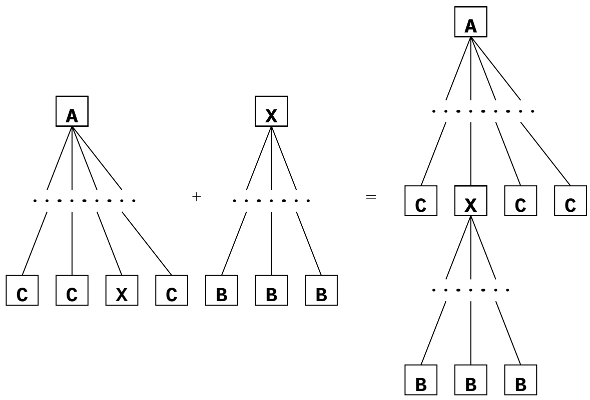

Figure 9: Illustration of the vertical linking process of two lexical frames where the root label X in the lexical frame at the right side is attached to the foot label X in the left one

ing blocks and learn this way what sentences can be built in a valid way and what sentences cannot. Although they were primarily developed to model human language behaviour, lexical frames are very appropriate to serve as building blocks. The next section describes the design and interface of a computer application that uses these lexical frames and actually allows the user to unify lexical frames together into one sentence diagram as a kind of jig-saw puzzle. It is important that this kind of building is restricted to the construction of a syntactically valid sentence diagram for scaffolding.

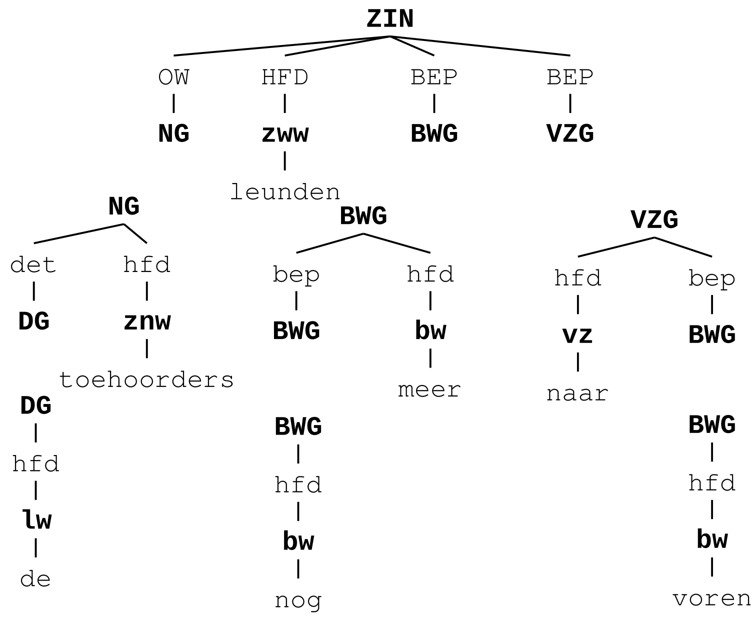

Figure 10: Example of some lexical frames

5 Applying Lexical Frames in a new kind of grammar exercises

Early experiments with the construction of sentence diagrams were done with the computer program *Palladio*. Figure 13 shows a screen-dump. The sentence diagram was shown in the shape of an ancient Greek temple with some blocks missing. The missing blocks were lying around, labeled with grammatical terms. The student's job was to fit the loose blocks so that a valid construction was made and the corresponding sentence diagram was completed. This is one possible implementation for sentence construction out of building blocks. However the interface is not up to date with modern multimedia technology any more. Furthermore it is not possible now to use bigger blocks than one label or to play the game without the labels within view. So it was decided to start from scratch again with a new design that was sketched by Nomi Olsthoorn depicted in figure 14. This time, the design is totally based on the idea of the lexical frame. Lexical frames are now shown in the form of ghost-like figures with heads and limbs. The segment that plays the role of head gets a real head and has arms representing the others grammatical functions. Great improvement in this design is also its possibility to replace the labels by shape, texture and color as illustrated in figure 15. The lexical frames can be used as flexible building blocks

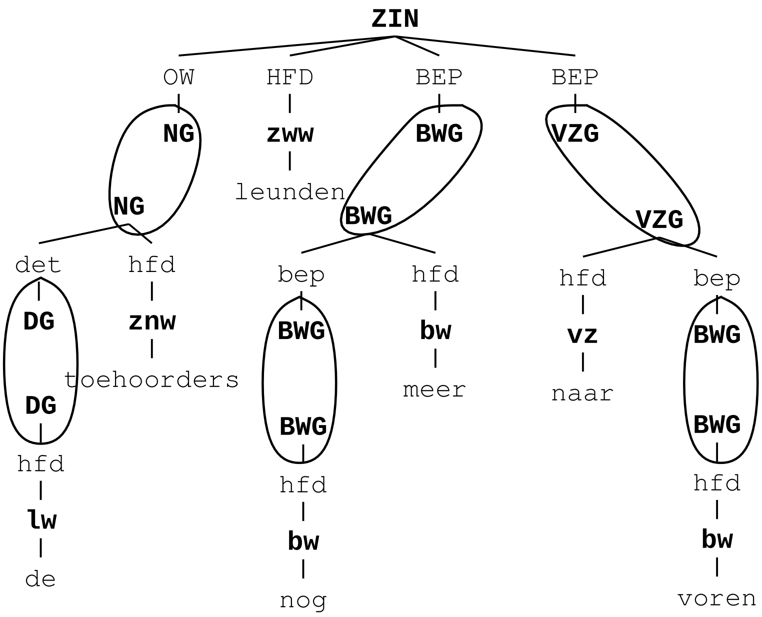

Figure 11: Ellipses indicating some possible unifications that can be applied on the lexical frames of figure 10

in the form of ghost-like shapes and unification is done by connecting the head of a lexical frame to an empty hand that fits.

Only one alteration to the combined syntactic trees is really needed: making sure that all leaf nodes are the head of a word group. Until now I used an abbreviated form of syntactic trees to keep things clear. For instance, the article *de* played the role of determiner in figure 6. It must now be made the head in a 'determiner phrase' which now gets the role of determiner in the noun phrase. The same goes for count nouns, particles and auxiliary verbs. This results in additional entries *DG*, *HLP* and *TWG* in table 4 in appendix A. Figure 16 shows the extended version of the syntactic tree of figure 6.

5.1 Design philosophy

There are several aspects of interface design that are of crucial importance for a successful educational application. Some of them apply especially for children. In the case of grammar instruction they might need, more than others, something familiar that helps them understanding the novel subject matter. Children know about puzzles and fitting shapes together. So what shapes can be found that are more flexible and capable of representing abstract concepts

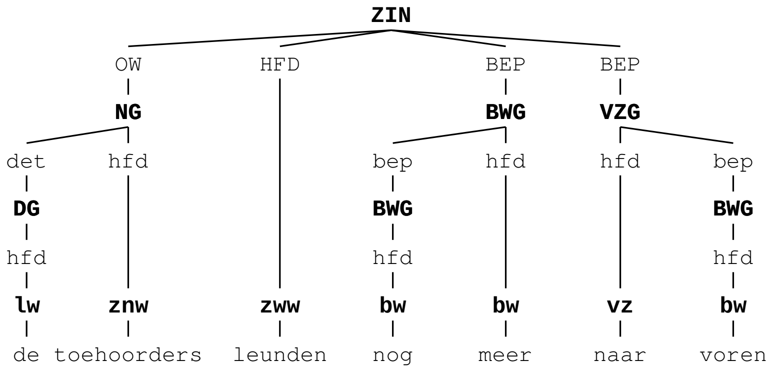

Figure 12: Result after unification

than a ghost-like shape, except an octopus perhaps. For instance, the ghost-like creatures in figures 14 and 15 can be combined in some sort of family portrait. I tried to enliven them by giving the creatures moving eyes that follow the moving mouse cursor. Something similar can easily be done too for the shape of the mouth in order to express a creature's mood: an 'ill tempered' ghost with empty hands can be made 'happier' by undergoing a successful unification. A desirable interface quality is to allow the user to reach a high mental workload of which only a minor part is taken by the complexity of interface, so that the rest can all be used for performing the task. The interface itself should be as simple as possible, with low visual complexity and all unnecessary information hidden. It can be argued that some aspects of an interface like having multiple windows are confusing [5]. For this reason I have chosen for having only one playing-field and without buttons. I believe that it is a good thing to make scaffolding an essential part of the interface. This means that the student is allowed only to make correct decisions and is warned immediately before the mistake is actually made, so that only correct patterns can be trained. An interface can become easier when it creates the illusion of manipulatable objects with reversible operations [18], and an immediate visual effect of each mouse action. Effects of animation and moving objects must not be overdone, however. They are useful when they express a reward or a penalty, but otherwise they only increase the visual complexity [16]. A learning environment for initial grammar instruction that uses sentence diagrams cannot do without an automatic layout of the sentence structure. It is very important to introduce the labels step by step by showing only relevant information, For instance a nine years old child in his first experience with grammar instruction should not be confronted with all different labels of nodes or properties at once. Later on, when 11 years old, he should gradually become capable of working his way around through these

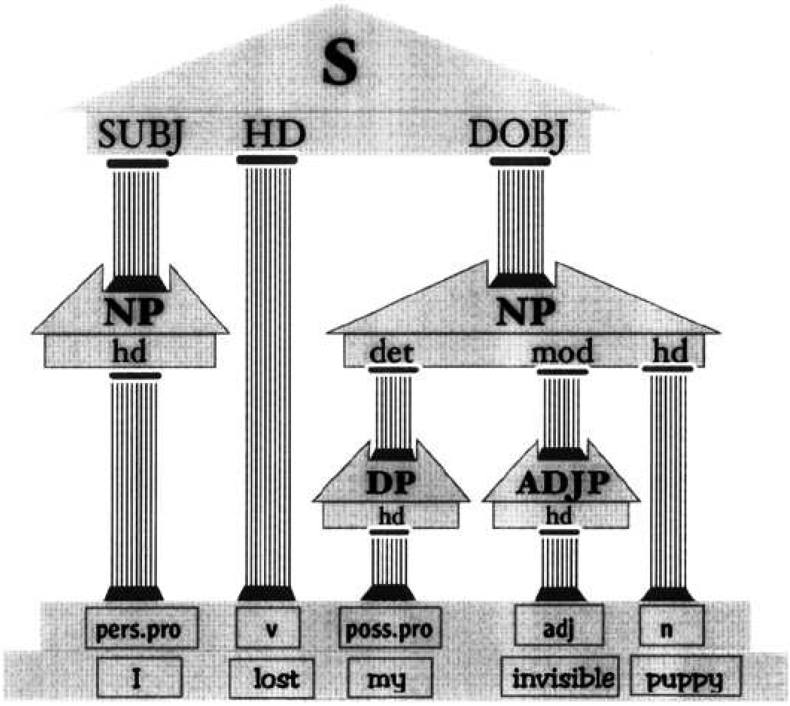

Figure 13: Finished sentence diagram in the shape of a Greek temple

labels. Eventually, the student can reach a level where also the feature matching process is visualized after a failing attempt to unify two ghosts, indicating the reason for failure. Labels must be placed at a plausible place and be hide-able so that only a few distinctive shapes remain. An important concept is *information hiding*. In the case of grammar instruction, complicated names for grammatical constructions can be hidden and replaced by a set of distinctive two-dimensional shapes. This way difficult names for things can be avoided until their meaning and role is understood. By leaving the labels out, younger children can start without knowing all the grammatical terms. When certain graphical effects are coupled to *discoveries* this can be a great stimulant to keep searching. All these considerations led to my design of a sentence diagram visualization of which the diagram in figure 17 is an example. It is my attempt to create a new attractive visualization for sentence diagrams. Each lexical frame structure is visualized by a little ghost. All segments with the function label *head* are shown with a head and a torso. The other segments are represented as limbs with a stretched hand. The ghost's torso contains the identifier. The root label is positioned on top of each head and the shape of that head is unique for that label. Only hands with the same label as foot fit onto this head. So there is a distinctive hand shape for every separate word group that is drawn instead or beneath the foot label

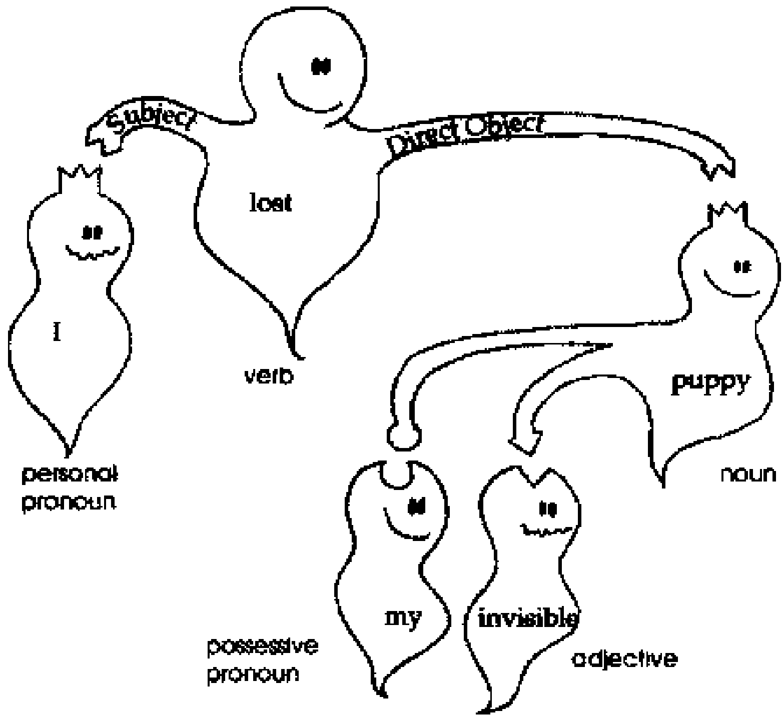

Figure 14: New sentence diagram design with ghost-shaped lexical frames

that fits exactly the head with that word group as root label. And there is a distinctive color of the hand that is drawn instead or beneath a function label for every syntactic function. I tried to use distinctive and fancy colors, like a yellow background, white ghosts and black lines and labels. During the game only the identifiers are shown to the children, as shown in figure 18 obtained from Aesop' fables [1]. To keep track of the entire sentence, the word labels are put once more on a horizontal line at the bottom of the screen, as a kind of anchor.

5.2 New exercises for grammar instruction

At this stage the user can *unify* the creatures by dragging the head and limbs onto each other. While a shape is dragged, the program tests whether this shape can unify with one of the other lexical frames on the playing-field. This is shown in color. When two segments collide, the color of their edge changes from black(default) to green when they fit, or to red if not. The reverse operation of unification is called *de*-unification and can be done by double-clicking on the limb or head involved.

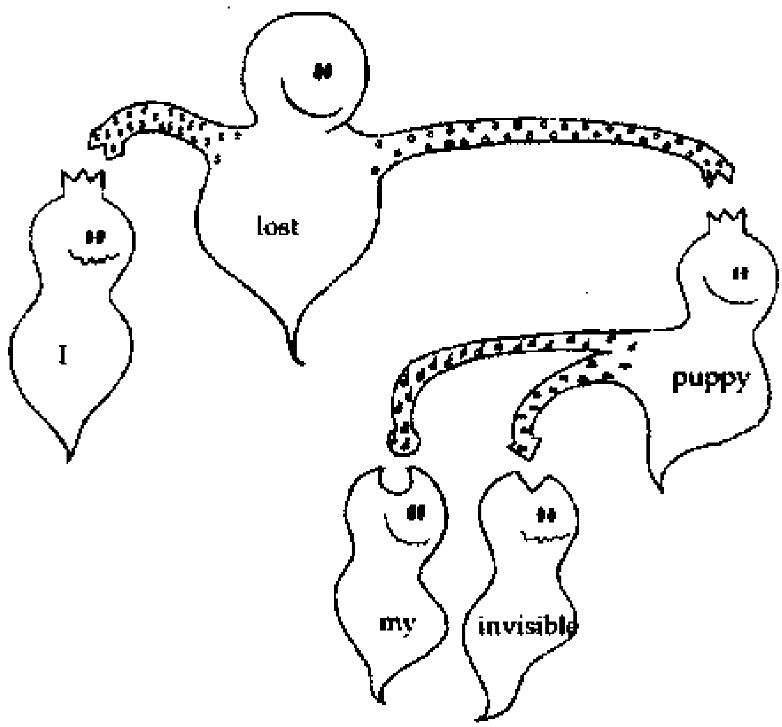

Figure 15: Sentence diagram design of figure 14 without labels

This new interface can be used in several other ways during grammar lessons. A novice user could start with completion of an almost finished sentence, or with watching a demonstration of bouncing ghosts that may try unification at a collision. After this introductory phase, a pupil may construct whole sentences out of lexical frames. In a more advanced stage the task can be pointing to a certain type of grammatical function like subject, finite verb, etcetera. At the highest level the student would have to determine whether the sentence is correct or perform a top-bottom analysis of structures from different levels of representation. Some other tasks are also useful, like determining the correct sequence of words. Still a lot of work must be done and some of the extensions described in section 7.1 must be added to make this prototype a full-fledged application. The next step would be to offer children a complete direct manipulation interface for lexical frames. This would eventually enable them to manipulate and play with grammar and its syntactic properties to help and stimulate their mental model building.

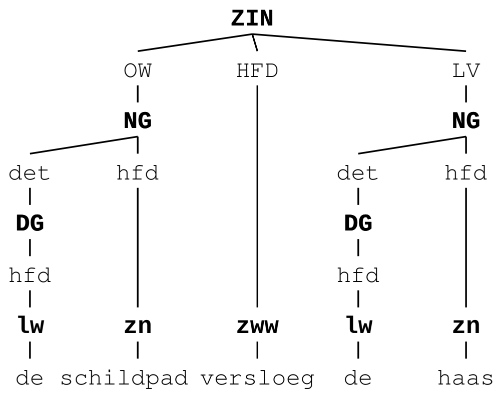

Figure 16: Extended sentence diagram

5.3 Access through internet

When developed further, the prototype's design has every potential to result in an application that can be used at schools. Before I started, the question has risen which platform and which programming language could be used best. There are schools that use Macintosh computers, but others use personal computers with Windows. There aren't many programming languages that are really platform independent. The new programming language *Java* seems the only appropriate one. The creation of applications in Java offers a great opportunity to circumvent the restrictions that all other program languages suffer. Java is the first byte-interpreted programming language that is available on all modern operating systems these days. When compiled as an *applet* it can even be accessed through any modern internet browser. Its concepts of events and threads make Java very suited for combining multimedia and interaction in one application that will run on all platforms. Not all standard multimedia support seems to be included yet but this shall improve soon. The only serious problem is the limited speed. Java source code is not compiled into machine code as with ordinary computer programs but into byte code that is executed by a local Java interpreter. Older computers might not be fast enough to be capable of a reasonable performance that is needed. So this is the price to pay for platform portability. The gain is that it is very easy to construct a website to support this project with, in addition, a demo of the game that can be played online through the internet. This site may be used to offer new sentences for the game to be downloaded from internet and might present an online version of the game along with the results and correspondence of participating schools. But the main reason for my choice to use Java lies in its platform independence. This resulted in one and the same *applet* which is tested and known to work, without recompiling, in the

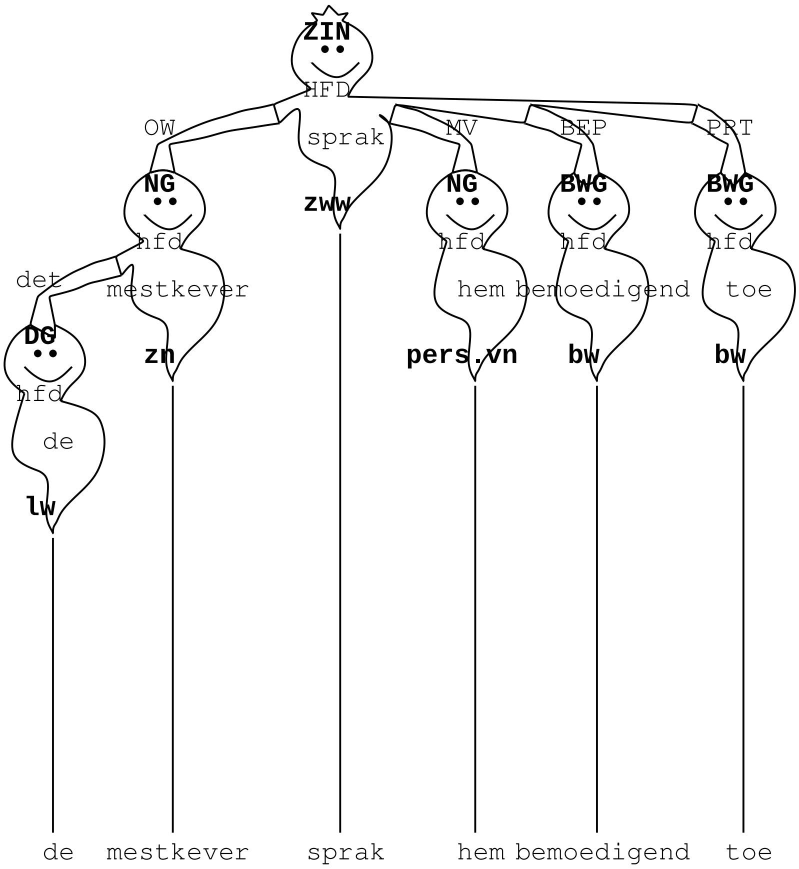

Figure 17: Sentence diagram with ghost-shaped lexical frames with labels

same manner under *Linux*, *SGI*, *Macintosh*, *Windows95*, *WindowsNT*, *HP* and *Sun* as well as the Java-enabled internet browsers Netscape and Explorer. Java forces the programmer to use all sorts of high-level programming techniques, like *Object-Oriented-Programming* for instance, i.e. subdividing program data into *classes* and into *methods* that operate on the classes. In the next section, describe the main issues of the implementation.

6 Implementation

The proposed ghost-like figures place high demands on a well-designed graphical interface that allows flexible manipulation of shapes. I chose to create *dynamic* shapes that would be able to be moved and could be filled, as opposed to *static* ones that would allow fast and fancier animation. Dynamic shapes are needed

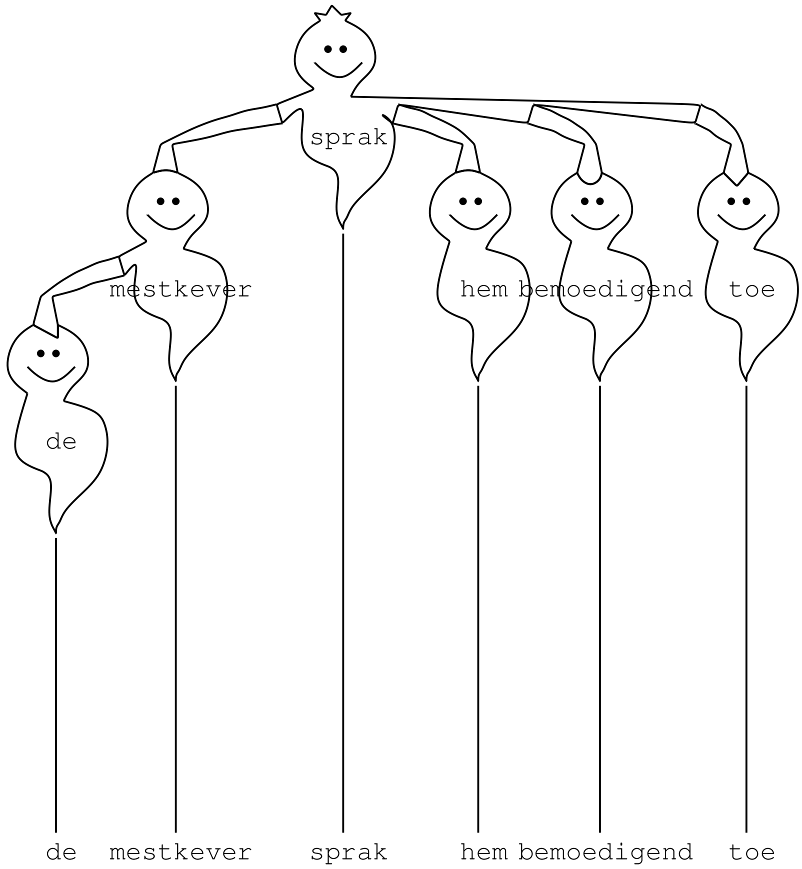

Figure 18: Sentence diagram with ghost-shaped lexical frames with hidden labels

for the automatic layout of the sentence diagrams. Due to the design it should not be difficult to create a *tweening*-effect when a shape's form is altered. When a new ghost first appears, or a new arm is added to it, or the ghost vanishes from the playing-field, this could be shown as a smooth transformation. In order to present the exercises in a game, I have designed a hierarchical graphical model that enables me to draw and manipulate lines, labels and flexible shapes. First I'll sketch its design that allows drawing the very complex shapes I wanted. An important structure that I composed to form the basis of all the graphical elements is the class *Shape*. It occupies the lowest level of my *graphical object hierarchy*. It is now time to describe this class in some detail because the graphical possibilities of this system heavily depend on the design of this class. Every instance of class *Shape* may contain a list of *children*, containing instances of *Shape* or subclasses of *Shape* with additional information added. Shapes are added to the list *children* by the methods *addShape()* and *insertShape()*. A

variable/function	description
$ancestor$	a shape can be attached to another shape, the $ancestor$ shape in which case this shape's offset depends on the $ancestor$'s offset
$children$	a linked list of attached other shapes
x	x-coordinate relative to $ancestor$
$midX()$	calculates the horizontal center, $(minX() + maxX())/2$
$minX()$	determines the left border of this shape
$maxX()$	determines the right border of this shape
$width()$	$maxX() - minX()$
$offsetX()$	absolute x-coordinate
y	y-coordinate relative to $ancestor$
$midY()$	calculates the vertical center, $(minY() + maxY())/2$
$minY()$	determines the top border of this shape
$maxY()$	determines the bottom border of this shape
$height()$	$maxY() - minY()$
$offsetY()$	absolute y-coordinate
$isOn(s)$	true, if this shape collides with shape s
$distance(x, y)$	gets the minimal distance between (x, y) and this shape
$getNearest(x, y)$	gets the nearest element to (x, y) from the $children$-list

Table 1: Shape methods and properties

Shape is drawn by the method *paint()* at a position (x, y) relative to its *ancestor*, the *offset*. This facilitates the movement of entire subtrees by changing only one set of coordinates. A shape is usually moved by function *move()*. By changing only the two variables (x, y) at the top *Shape* of a subtree, the offset of all the members of that subtree alters automatically by the same amount. Function *distance(x, y)* determines the minimum *Euclidean* distance from point (x, y) to the shape's *children*, or when there are none attached, to the shape's $offset$. This method is used by function *getNearest()* to get the nearest child of this shape to some point (x, y). There are several additional attributes for a *Shape*, like a color and a pop-up window to streamline the behaviour of the various subclasses. Finally, the class *Shape* defines the geometric properties over all of its children as described in table 1. This includes *width()*, *height()*, *midX()*, *midY()*, *minX()*, *minY()*, *maxX()*, *maxY()*, *offsetX()* and *offsetY()*. This class *Shape* forms the basis for various other *shapes* with specialized behaviour. In object-oriented design this means that these specialized shapes are all *subclasses* of *Shape* and *inherit* (=*share*) the same geometric properties and operations. For example, a shape that draws a label can be defined as a *Shape* with its center at its *offset*. The geometric boundaries are now defined by the size of the label in the picture. I defined for that purpose class *LabelShape* which contains a *string* and a font. The classes *PolygonShape* and *BSplineShape* are defined, respectively, in order to draw polygons and a sophisticated type of smooth curves, *B-splines*[2].

6.1 Drawing the ghost-like shapes

Given a complete lexical frame it is still difficult to draw the flexible parts like hands and limbs in a natural manner. The static parts of the ghosts I could fortunately draw, before I had my own B-spline algorithm (see section 6.2), by means of a program by *S. Spaans*. This resulted in the sets of *control points* for distinctive hands, heads, and torsos as shown in figure 19. There are four different torsos, one for ghosts with arms on both sides, one for ghosts with only arms on the right, one for ghosts with only arms at the left, and the last one is for ghosts without arms. Every *head*-segment gets the appropriate torso and head, with a structure on top of the head which depends on the root label of that segment. For every *non-head* attached to a head segment, extra control points are added to the head and torso of the head in order to create the dynamic arms. These are first ordered by their x-offset. The control points are then added following a sine curve from the previous arm (or the head) to the hand. The points for going backwards are put onto the stack. Each arm then gets its own shape of hand, which depends on the foot label of that segment. And now the other half of the arm can be completed by emptying the stack. To ensure a natural way of drawing, these arms are coerced horizontally by their neighbouring arms.

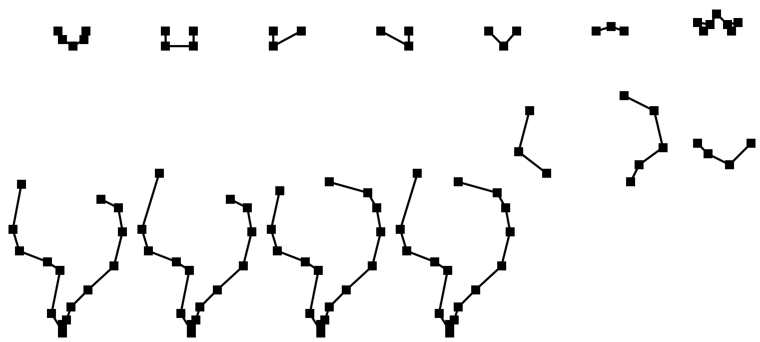

Figure 19: Control points for the static parts of the ghosts, connected by straight lines

6.2 Shape description

Most standard graphic libraries are limited to the drawing of pixels, lines, rectangles, polygons and ovals. These are very primitive and simply cannot satisfy

[2]The *B-spline algorithm* is explained later in section 6.2.

the demand for smooth flexible shapes themselves. Fortunately, a range of shape descriptions have been invented to describe a wide range of curves. In general a shape description is a mathematical formula in the form of a parametric function that is applied on a number of control points and describes some sort of curve. Once a suitable parametric function $p(t) = (x(t), y(t))$ is found, it suddenly becomes very easy to draw the corresponding curve by substituting some sequence $t_0, t_1, t_2, ...$ for t where $t_i \leq t_{i+1}$. This makes parametric functions powerful tools for generating and representing curves. Some of them have become very popular in interactive graphic design. The most important ones are spline curves. A spline curve is a blend of vectors that uses piecewise polynomial blending functions which make the curve continuous at each point. One class of splines, the Bezier curve, defines a curve over a sequence of $p_0, p_1, ..., p_n$ control points. The Bezier curve formula as a parametric function is as follows:

$$p(t) = \sum_{k=0}^{n} p_k B_k^n(t)$$

where

$$B_k^n(t) = \binom{n}{k}(1-t)^{n-k}t^k$$

are the Bernstein polynomials ($\binom{n}{k} = \frac{n!}{k!(n-k)!} \quad for\ n \geq k$). These Bernstein polynomials have the elegant property that their weighted sum $\sum_{k=0}^{n} B_k^n(t)$ is always 1, while they shift the share of the participating control points in the weighted sum $p(t) = \sum_{k=0}^{n} p_k B_k^n(t)$ gradually from p_0 to p_n as parameter t increases. Nice things about these Bezier curves are that they start at p_0 and end at p_n precisely, and whenever they need to be subjected to an affine transformation such as scaling, rotation, translation, etc., it is only necessary to transform the control points rather than every single point on the curve. However, by the way they are defined they bring a serious disadvantage for design purposes. When one control point is moved a bit, as little as it may be, the entire curve shall have to change as every point on the curve is a weighted sum of all control points. This makes local control impossible. Because it is desirable to have local control to manipulate a curve, a similar curve description that calculates the weighted sum over only a limited number of adjacent control points would be very welcome. Fortunately, such splines exist. Given any sequence of adjacent curve points, there is a set of blending functions that form the *basis* for the spline. This means that any spline whatsoever can be formed by choosing the appropriate control points. One such class of splines that defines a *basis* is the *B-spline*. Here is its parametric function:

$$p(t) = \sum_{k=0}^{n} p_k N_{k,m}(t)$$

where k is the number of control points, m is the order of the polynomial functions,

$$N_{k,m}(t) = \frac{t - t_k}{t_{k+m-1} - t_k} N_{k,m-1}(t) + \frac{t_{k+m} - t}{t_{k+m} - t_{k+1}} N_{k+1,m-1}(t)$$

and

$$N_{k,1} = \begin{cases} 1, & if \;\; t_k < t <= t_{k+1} \\ 0, & otherwise \end{cases}$$

In the case of fourth order B-splines, ($m = 4$), the spline consists of a set of curves each based on four adjacent control points: $p_i, p_{i+1}, p_{i+2}, p_{i+3},\ i = 0..n-4$. Figure 20 shows the control points sets of figure 19, but now connected by means of the B-spline algorithm.

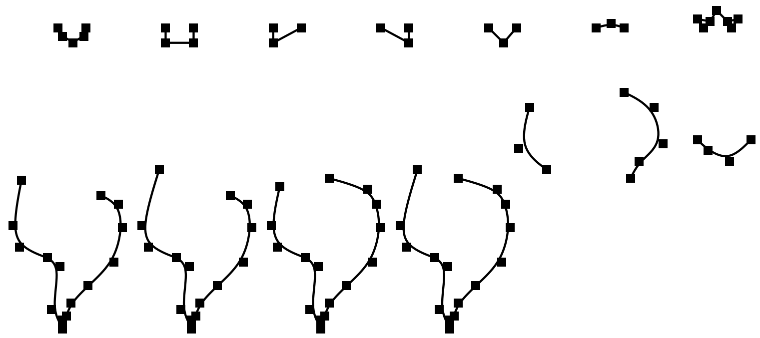

Figure 20: Control points for the static parts of the ghosts, connected by smooth B-spline curves

6.3 Living eyes

A rather funny effect it is to position the little eyes on the figures as if they are watching something, for example the mouse cursor, as depicted in figure 6.3. To do this one can draw the black oval of an eye, instead of at position (x, y), at

$$(x + l \cdot \sin(angle), y + l \cdot \cos(angle))$$

where *angle* is the direction of the line between the mouse cursor and the eye and l is the minimum of the *Euclidean* distance between the mouse cursor and the eye and the radius of the eye minus the radius of the pupil. And when in state of *sleeping*, an arc can be drawn which represents a *closed* eye. Nothing can be more simple! Class *eyeShape* is used to create this funny effect. As a subclass of *Shape* it can easily be attached to every other *Shape*. Every instance of an $eyeShape$ has defined one radius for the pupil and one for the whole eye itself. When an eye is to be painted, the position of the pupil is simply translated by some vector $\vec{v}$. The direction of the imaginary line between mouse-cursor and its position gives $\vec{v}$'s direction. Of course, the pupil may not exceed the border of the eye. This is why the length of that $\vec{v}$ is set to the minimum of the radius

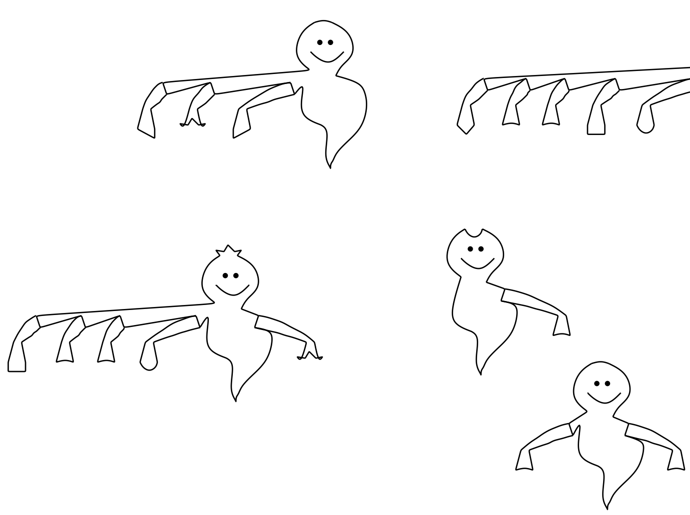

Figure 21: Example of ghosts with limbs

of the eye minus the radius of the pupil and the distance between the center of the eye and the mouse position. In formula $\vec{v}$ becomes

$$\vec{v} = \min(|\ \vec{m} - \vec{e}\ |, eye\ \ radius - pupil\ \ radius) \cdot \begin{pmatrix} \sin(angle) \\ \cos(angle) \end{pmatrix}$$

where $\vec{m}$ and $\vec{e}$ are vectors containing respectively the mouse and eye positions.

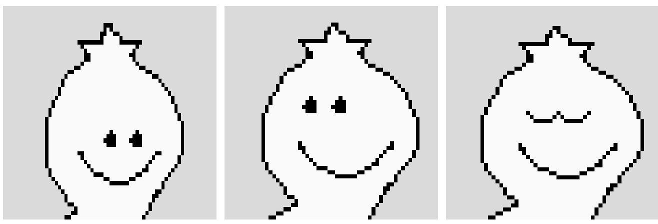

Figure 22: The eyes of the ghost-like creatures, in several states.

6.4 Bracketed string notation

The sentence diagrams are stored in a string in which every subtree is surrounded by brackets except the identifier and the parts-of-speech label which are separated by a space. The example in table 2 shows the complete string for the sentence diagram in figure 23 obtained from Aesop [1]. At the start of the program, these sentence diagrams are loaded from disk, and the segments in these sentence diagrams are collected, which can be used later, as explained in section 6.8.

```
(ZIN (OW  (NG(det(DG(hfd(LW Een))))
              (hfd(ZN adelaar))))
     (HFD (ZWW zat))
     (LV  (NG(det(DG(hfd(LW een))))
              (hfd(ZN haas))))
     (PRT (BWG(hfd(bw achterna)))))
```

Table 2: Example of a sentence diagram in *bracketed string notation*

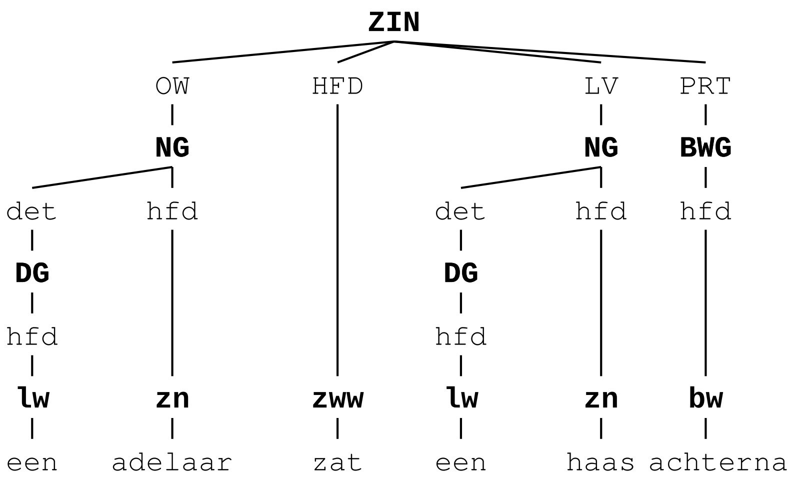

Figure 23: Sample sentence diagram

6.5 Constructing a Lexical Frame into *SegmentShapes*

As described in section 4, sentence diagrams can be divided into segments. I define a *SegmentShape* as subclass of class *Shape* that contains the labels and the lexical properties of that segment. In fact a lexical frame is put together from a number of instances of these *SegmentShapes* that are joined horizontally through a linked list of *brothers*. The *SegmentShape* can be connected at both ends to other *SegmentShapes* through variable *parent* for the root, and variable *tail* for the foot. When a non-head segment receives a call to *move*, it passes this call to the head itself, this way the entire lexical frame shall be moved. In turn, the head is moved and moves along all of its brothers and their children. In the model there is exactly one *head* for every lexical frame. Every segment contains a link called *head* to it. An array *label[]* is used to store all available labels. And the syntactic properties that come with the root and foot labels are stored in two separate *FeatureMatrices* of that segment. To create a sequence of lexical frames out of a sentence diagram, class *LabelTree* is used to convert sentences in bracketed-string notation into a tree structure. Then, for every lexical frame, the segments are cut out and linked horizontally with method *horizontalLink()*. No matter in what *SegmentShape* is started, method *getRoot()* finds always the top node of that tree structure by recursive calls to the same function in its *parent*. Geometric properties are altered in only one aspect with respect to those in class *Shape*. Functions *minX()*, *maxX()*, *minY()*, *maxY()* of a head-segment are determined not only over the children (in this case, the labels) but also over the brothers and their children.

6.6 The syntactic properties in the Feature Matrix

The syntactic properties for every *SegmentShape* are constructed by class *FeatureMatrix*. It simply adds the syntactic properties with a choice list of all possible values to the pop-up panel of the *SegmentShape*, i.e. one segment, to which it belongs. For every label of a *SegmentShape* the appropriate properties can be found in that diagram. Every label of the *SegmentShape* may bring in a number of properties. At this moment the values for the syntactic properties are not set yet. During unification they are simply ignored. Some database must be explored to set the correct values of the properties.

6.7 Interaction on lexical frames

To start with, I made a simple *puzzle mode* in which the lexical frames in their ghost-like appearance are shown on a rectangular playing-field and can be controlled by the computer mouse. The ghosts are equipped with the limbs corresponding exactly to the edges in the sentence diagram. They can be dragged to be unified or double-clicked to be de-unified again after which the shapes are automatically re-arranged. Figure 24 shows an example. Normally, only the identifiers (in the ghosts' torsos) are shown, but here I left them to illustrate that the ghost shapes fit onto the labels quite well. In this first prototype of the grammar game mouse behaviour is quite simple:

- *mouseEnter* causes the state *awake* and the eyes of the little ghosts become opened.
- *mouseMove* highlights the nearest shape to the mouse cursor in blue.
- *mouseDown* causes variable *current* to become the nearest ghost; double-click de-unifies *current*, control-click removes the nearest limb (an entire lexical frame is removed when control-click is done near head-segment)

- *mouseDrag* moves shape *current*. If the nearest shape for *current* is close-by enough, then they both become green when they may be unified, or red otherwise

- *mouseUp* checks if *current* can be linked to an empty slot of another shape. If so, unification is performed and the layout is reshaped automatically.

- *mouseExit* returns the state to *sleeping*, and the eyes of the little ghosts close.

After a unification or a *de*-unification operation, a randomly selected audio-sample is played. I use two separate sets of samples, one set for unification and one for *de*-unification. This concludes the simple scheme of *puzzle mode*. In this stage only puzzles can be made of the set of about 50 example sentence diagrams that I have in bracketed string notation(see section 6.4).

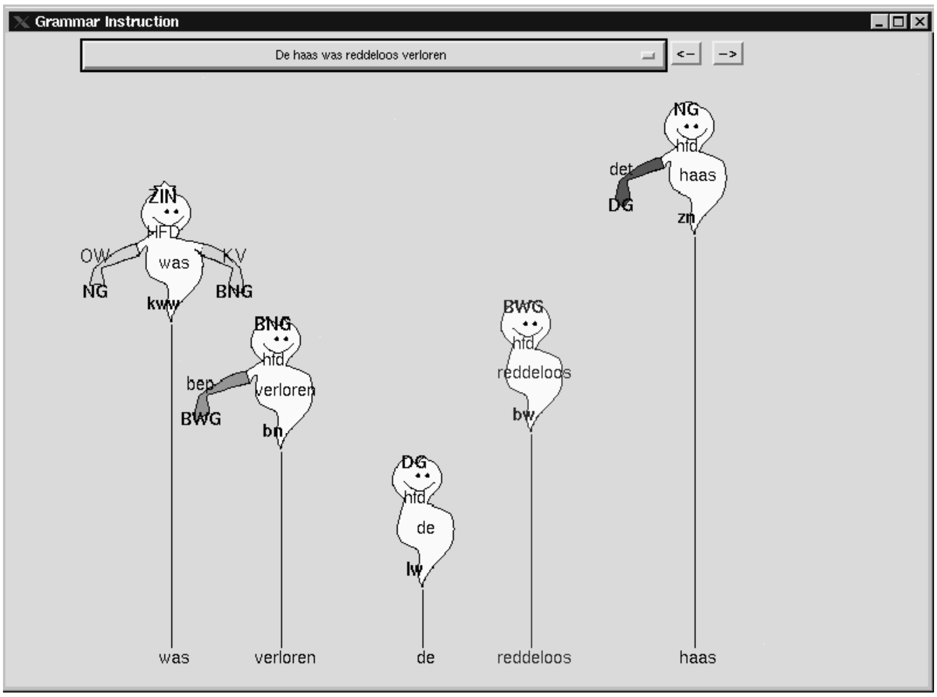

Figure 24: Sample of puzzle mode

6.8 Free-edit mode

A more advanced mode than *puzzle mode* is *free-edit mode*. Figure 25 shows an example. This mode allows the user to choose the words from a random selection

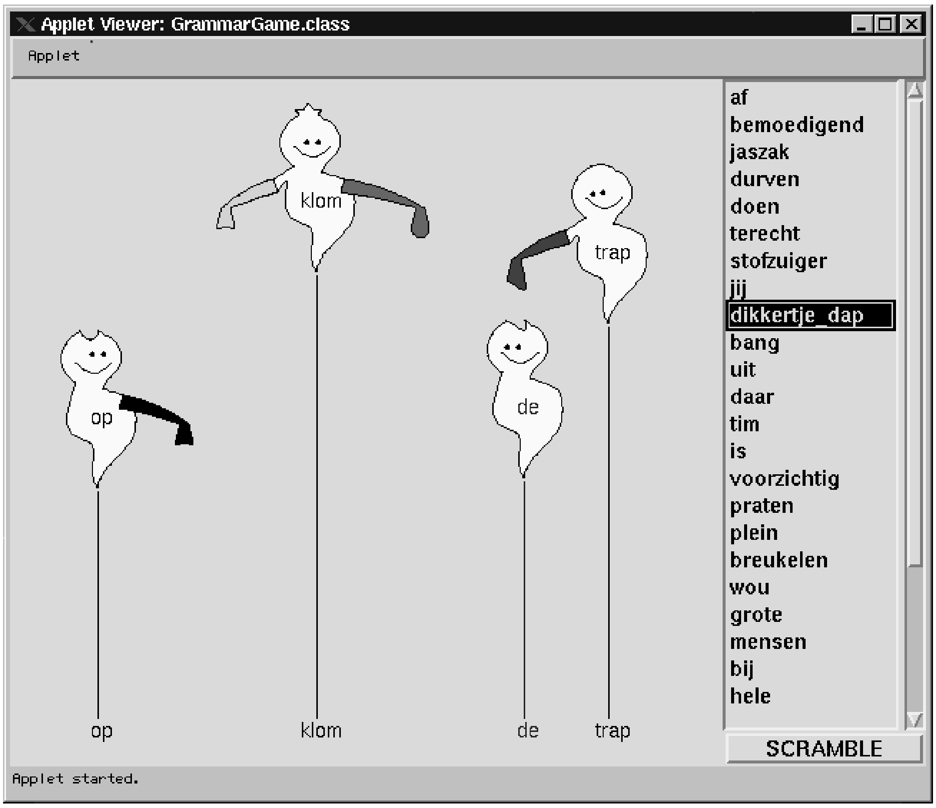

Figure 25: Screen-dump of free-edit mode

of all the words in the available sentences. These are placed in a column at the right side of the screen. By pushing button *SCRAMBLE* the user obtains a new random selection of words. Selecting a word results in the birth of a new ghost on the playing-field with the appropriate labels. And any ghost can be removed from the playing-field by *clicking* on that ghost, while the *Control*-key is pressed. The new ghost will get the same part-of-speech label as in the example sentence where it came from. And of course, as all segments with an identifier, it will be functioning as head. The only problem is that it is not always certain which *root* label should be given, as some part-of-speech labels fit into more than one type of constituent. For now, I choose the root label randomly from the list of root labels for that part-of-speech label, which I create automatically during the reading of the sentence diagrams from file. Not all grammatical constructions are now present in the existing sentence diagrams. So some new sentences diagrams with the still missing grammatical constructions are needed to complete the set of allowed segments. A newly created ghost will start on an empty location on the playing-field and without limbs. Whenever another ghost comes nearby, a new limb is formed by adding a segment to the lexical frame of the ghost and redrawing the shape. This new segment consists always of a set of labels that is valid for the lexical frame it belongs to. If the two ghosts can be unified, the limb will be equipped with the root label of the other ghost, and with a proper function. This choice of function is not deterministic however, e.g. an *NG* fits into many roles to a *ZIN*, like *OW*, *LV*, *MV*, etcetera. Whenever this occurs, it is probably best to assign a function to that segment which is not present in the lexical frame yet. Now, like in puzzle mode, a hand shape that fits onto that head can be attached to it. With free-edit mode and puzzle mode the user has all freedom to build his own sentence diagrams. These schemes shall become fully operational when I use a database to set the syntactic properties. In the last section, I shall suggest some other possible extensions that can be added to improve this program.

7 Conclusion

Grammar instruction is an important means to help children learning advanced aspects of language. However, there are not true satisfactory grammar instruction methods available given the limited available time at schools. What I believe is needed are three things, *scaffolding*, *visualization* and *immediate feedback*. This can probably best be done in combination with computers in a playful interactive setting. The prototype I designed and implemented is a considerable step towards such a system. It proves that it is possible to actually build a useful application for grammar instruction. Sentence diagrams can be built by linking *lexical frames* which are building blocks that originated from psychological models. The building blocks are to be constrained only by their grammatic properties. My prototype needs a lot of additional work yet, before it can actually be used by pupils. For practical use, some educational material must be developed first, but the advantages and possibilities of interactive grammar instruction are already clear. Direct manipulation can give pupils the feeling of having control over a system. With lots of practise they can obtain self-confidence about dealing with grammar. This stands in sharp contrast with the red ink feedback that is given by some teachers. This particular design is based on construction and visualization of sentence diagrams in which the grammatical terms can be hidden while a unique distinctive shape or color remains visible. This might work out well, as it has often been observed that visual aids to memory appear to be more effective than the abstract terms [15]. Grammar instruction is often regarded as unattractive. The best response to this probably is to present it in the form of a game, which is supposed to be attractive. Games, in general, are an important way to learn coping with situations and gaining control. The more creativity is allowed the better. I expect that the role of computers in basic education will become more and more important in the future, and many educational software packages are proving themselves already in several different areas. However, not much software seems to be available for (Dutch) grammar education yet. Interactive educational computer games like this provide schools with a new generation of modern computer tools in new areas. With these tools, results and progress of pupils can be measured and compared, while frequent mistakes in specific areas can be tracked down and hopefully resolved. My present system is in Dutch, but the implemented system needs not to be altered much for most other West-european languages. It is up to psychologists, teachers and pupils to find out whether this system provides a worthwhile improvement upon or addition to conventional grammar education. Considering the complexity of grammar education and the limited time available at schools, grammar instruction can only be successful when extensive training is combined with effective exercises. Replacing traditional grammar instruction by sentence construction can be a big step forward [8] and lead to more efficient learning [2].

7.1 Future extensions

The quality of learning processes depends on the frequency, level, quality and type of feedback [3]. Some additional dialog structures must be added to ensure this and to facilitate more efficient mental model building. *Artificial Intelligence* can be added for smart training, presenting to the student the difficult parts that are not yet comprehended, or trying to provide easier exercises to train these tasks. During playtime it may happen that a user makes the same type of mistakes. It would be an idea to extend this program with a special unit that monitors such frequent errors. Some control module could then take action and explain the things that went wrong and or present the solution to a problem by means of a windowed dialog. Something could also be done to make the challenge and difficulty level rise for experienced users. A frequency table as in [12] that contains the frequencies of used words for youth lecture under twelve could be used for this by starting with frequently used words and gradually proceed to the less frequently ones with more syllables. Another thing would be to do first some explaining by spoken text or through a dialog. I for myself prefer to present the user short dialogs during a demo of the unification of an entire sentence diagram. This to demonstrate which are the possible actions, the reversible actions and the mistakes. In a final product it would be best to include a time/point system with points and a high score list for competition, though the speed depends sometimes on the computer on which the game is played. In the prototype there are audio-effects coupled to events. This kind of immediate feedback seems a very good approach. Multi-modal learning environments get hold of the attention from both auditive and visual channels so there probably is less distraction possible. When using only visual or only auditive information, an important part of the input side of the user is neglected on which there is no control or influence. And without necessary attention of the pupils learning probably is not as effective as it could be. Audio effects can be helpful in expressing rewards or penalties. Apart from that it is an idea to send the freshly formed sentence constructions in spoken language to the loud speaker. Instructions can be given in the form of a pre-recorded message. Speech output can be disturbing however and it has been found that (adult) people can handle interfaces with textual information faster [13]. Audio effects now are used in general as a reward when unification is accomplished by the pupil, but could easily be done to warn the player that he runs out of time or when something else occurs. It shall be very difficult to circumvent all of the short-comings of traditional grammar instruction, but on the other hand this can be regarded as a challenge as well. Wouldn't it be wonderful if it were not necessary to overflow poor performing pupils with negative feedback in red ink anymore.

Acknowledgements

I would like to thank all people who brought new ideas or contributed in any other way, especially Nomi Olsthoorn for her support and great drawings of what now are nice little computer-animated ghosts. None of this would have been possible without the psycholinguistic models developed by, and support of, Gerard Kempen.

Appendix A: Tables with grammatical terms

English	**Dutch**	**abbreviation**
main verb	zelfstandig werkwoord	ZWW
auxiliary verb	hulpwerkwoord	HWW
copula verb	koppelwerkwoord	KWW
substantive, noun	zelfstandig naamwoord	ZN
article	lidwoord	LW
adjective	bijvoeglijk naamwoord	BN
numeral	telwoord	TW
preposition	voorzetsel	VZ
adverb	bijwoord	BW
coordinating conjunction	nevenschikkend voegwoord	NEG.VGW
subordinating conjunction	onderschikkend voegwoord	OND.VGW
interjection	tussenwerpsel	TUSSENW
personal pronoun	persoonlijk voornaamwoord	PERS.VN
possessive pronoun	bezittelijk voornaamwoord	BZIT.VN
demonstrative pronoun	aanwijzend voornaamwoord	AANW.VN
interrogative pronoun	vragend voornaamwoord	VRAG.VN
indefinite pronoun	onbepaald voornaamwoord	ONBEP. VN.
reflexive pronoun	wederkerend voornaamwoord	WEDND.VN
reciprocal pronoun	wederkerig voornaamwoord	WEDIG.VN
relative pronoun	betrekkelijk voornaamwoord	BETR.VN

Table 3: Parts of speech

English	**Dutch**	**abbreviation**
sentence	zin	ZIN
noun phrase	naamwoordgroep	NG
adjectival phrase	bijvoeglijk-naamwoordgroep	BNG
adverbial phrase	bijwoordgroep	BWG
prepositional phrase	voorzetselgroep	VZG
determiner phrase	determineerdergroep	DG
auxilary verb phrase	hulpwerkwoordsgroep	HLP
numeral phrase	telwoordsgroep	TWG

Table 4: Phrasal categories

English	**Dutch**	**abbreviation**
subject	onderwerp	OND
direct object	lijdend voorwerp	LV
indirect object	meewerkend voorwerp	MV
prepositional object	voorzetselvoorwerp	VZVW
predicate	koppelvoorwerp	KV
head	hoofd	HFD
auxilary	hulp	HLP
particle	partikel	PRT
subordinator	onderschikker	OS
complement	complement	CMP
complementer	complementeerder	CMPR
determiner	determineerder	DET
quantifier	teller	TEL
prepositional object	voorzetselgroepobject	VZGOBJ
modifier	bepaling	BEP

Table 5: Syntactic functions

English	**Dutch**
tense	tijd
number	getal
person	persoon
finite verb	persoonsvorm
participle	deelwoord
subjunctive mood	aanvoegende wijs
infinitive mood	onbepaalde wijs
indicative mood	aantonende wijs
imperative mood	gebiedende wijs
transitive	overgankelijk
gender	geslacht
definiteness	wijze van bepaaldheid
case	naamval
countable	telbaar
diminutive form	verkleinvorm
superlative, comparative form	overtreffende, vergelijkende trap
inflection	verbuiging
separable	scheidbaar

Table 6: Word properties

References

[1] Aesopus. *Alle fabels.* Amsterdam: De Driehoek, 1990.

[2] J.H. Boonman and W.A.M. Kok. *Kennis verwerven uit teksten.* RUU, VOU, Utrecht, 1986.

[3] S. Brown, S. Armstrong, and G. Thompson. *Motivating students.* Staff and Educational Development Systems, 1988.

[4] J.M. Caroll, J. Reitman Olson, and A. Arbor. Mental models in human computer interaction. In *M. Helander(Ed), Handbook of Human Computer Interaction.* Amsterdam: Elsevier Science Publishers, 1988.

[5] J.M. Carroll, R.L. Mack, and W.A. Kellogg. Interface metaphors and user interface design. In *M. Helander(Ed), Handbook of Human Computer Interaction.* Amsterdam: Elsevier Science Publishers, 1988.

[6] J. A. Adams D. F. Rogers. *Mathematical Elements for Computer Graphics.* McGraw-Hill International Editions, 1990.

[7] S. Evers. *Grammatica, van ontleden tot zelf zinnen bouwen en zelf ontdekken.* Samson H.D. Tjeenk Willink, Alphen aan de rijn, 1997.

[8] G. Jr. Hillocks. *Research on written for teaching, new directions for teaching.* Urbana, Illinois, NCRE/ERIC, 1986.

[9] G. Kempen. *Visuele Grammatica, een constructieve methode om zinnen te ontleden.* Department of Psychology, Leiden University, February 1996.

[10] G. Kempen. *Grammatical Performance in Human Sentence Production and Comprehension.* Department of Psychology, Leiden University, September 1997.

[11] G. Kempen. Visual grammar: Multimedia for grammar and spelling instruction in primary education. In *K. Cameron(Ed.), CALL: Media, Design and Applications.* Lisse: Swetz and Zeitlinger, 1998.

[12] G. Staphorsius, R.S.H. Krom, and K. de Geus. *Frequenties van woordvormen en letterposities in jeugdlectuur.* Cito, Arnhem, 1988.

[13] L.A. Streeter. Cgi to collect synthesis. In *M. Helander(Ed), Handbook of Human Computer Interaction.* Amsterdam: Elsevier Science Publishers, 1988.

[14] A. van Gelderen. *Taalbeschouwing, wat is dat ?* SCO, 1998.

[15] L. Verhoeven(Ed.). *Handboek lees- en schrijfdidaktiek.* Amsterdam/Lisse: Swets and Zeitlinger, 1992.

[16] W.L. Verplank. Graphic challenges in designing object-oriented userinterfaces. In *M. Helander(Ed), Handbook of Human Computer Interaction*. Amsterdam: Elsevier Science Publishers, 1988.

[17] D.D. Woods and E.M. Roth. Cognitive systems engineering. In *M. Helander(Ed), Handbook of Human Computer Interaction*. Amsterdam: Elsevier Science Publishers, 1988.

[18] J.E. Ziegler and K.-P. Fähnrich. Direct manipulation. In *M. Helander(Ed), Handbook of Human Computer Interaction*. Amsterdam: Elsevier Science Publishers, 1988.