The Fitaly Interface

J.F. van der Straten

TU Delft

HOGESCHOOL ROTTERDAM

CONTENT

Acknowledgement 5
Introduction 7
Project Overview 9
Project Description 9
Project Goal 9
Background theories 13
Design 21
Implementation 25
Usability Evaluation 27
Conclusion 29
Future Work 31
References 33
Appendix 35

Acknowledgement

This report describes my final graduation project for the RIVIO University for higher professional education in Rotterdam. The project was carried out in the Man Machine Interaction Group, the faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology.

I would like to thank Drs. Dr. L.J.M. Rothkrantz and Siska Fitrianie for guiding me during this project.

A big thanks to Dhr. Ir. Abdelghany for his ideas, suggestions and cooperation.
Same goes for Eric de Ronde, his testing and ideas helped me a lot during this project.

Furthermore, I would like to thank the members of the MMI-RobotSIG group. The meetings were very interesting and helpfull and it was a good way to keep updated about other projects on the Delft University of Technology.

The two visits to the Philips Research Center gave me much information about the iCat Project. Therefor I want to thank Philips and Mr. van Breemen in particular.

Finally I would like to thank the people of TU Delft for providing their knowledge and facilities

Introduction

This project aims at developing an user interface combined with a speech engine and a library of pre-defined face expressions to improve the user (with communication disability)'s communication with other people. With the interface the user can select words by pointing which form a sentence and combines the output with selected face expressions. A user test is conducted to assess whether the iCat's facial expressions are correctly recognized.

During this project we discovered the iCat OPPR software was not ready yet for connecting it to our main interface. The main focus now was researching the possibilities of an user interface which can deliver fast results for text-to-speech conversion for disabled people.

Project Overview

Project Description

Due to technical progress nowadays these days more people are able to overcome their physical lacks. If someone used to miss a leg for instance, a wooden leg was put in place so the person was able to walk. Nowadays a prothesis is used. The person has more possibilities like bending, stretching and can function almost like with a normal leg. More and more this kind of solutions make it possible for people to live better with their shortcomings.

One of our students at the Hogeschool Rotterdam, Eric de Ronde, currently has problems communicating with other students because he is unable to speak, show emotions and move his hands to control a mouse for instance. For communicating he uses the program “Dasher” .

Dasher is an information-efficient text-entry interface, driven by eye pointing movement. It uses pointing for selecting the characters and predictions of a language model to determine how much of the world is devoted to each piece of the text (Figure 1). Probable pieces of text are given more space for selecting them more quickly and easily. Improbable pieces of text (text with spelling mistakes) are given less space, so they are harder to write. This language model constantly learns the user input.

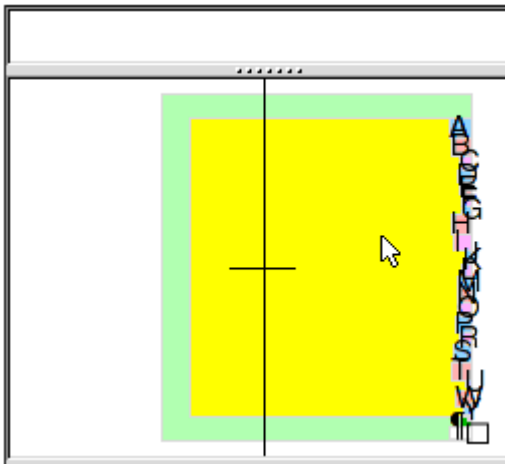


Figure 1 Screenshot of the Dasher program

Eric uses an eyetracker for controlling the zooming interface of the program. One of the disadvantages of the current system, is the awareness of the people around him. He has to make sound before he can show what he wants to say. For this purpose, the iCat can be used to speak up the output of the Dasher program loud via text-to-speech.

Another disadvantage is the lack of emotions in the output of Dasher. For the user, it would be nice to have choices of emotional expressions in combination with the output of the text interface. The iCat then could show the facial expression when speaking the words which are generated with the interface. This would make the iCat possible to be used for multimodal communication via verbal and nonverbal channels.

Project Goal

The goal of the project is to design and implement an interface between the user and the iCat, which makes it possible to have the iCat speaking and showing the expressions the user points. This interface has to deliver a quick way for selecting words which are combined to a sentence.

To pursue the goal, this project aims at exploring four directions:

1. Communication interface

Our goal is to develop interface that can receive input from eye-movement detector. For this purpose, we will adopt the interface of Dasher, i.e. a graphical user interface with zooming elements. The reason is the interface has been tested thouroughly for the target users by the

developers of Dasher (see [Das01]). We will improve the interface by adding an interface for a word prediction. The reason is that it is important that the sentences can be created in a quick way. Therefore, the interface for the user must be designed in a way the words can be combined quickly.

2. Word/text prediction

To generate a smoother dialog, we aim to exploit a word prediction to help the user to improve the input speed. This prediction could also help the user to improve the quality of syntax. It predicts which words are most likely to follow a given segment of a string that is grammatically correct. As a result, the number of character look-up processes that is necessary to generate the string will be reduced.

3. Face expression

After the user has chosen the words which form the sentence, emotional face expressions can be added to the output. It should be possible to have a different face expression in the beginning of the output than at the end of it. So the text-to-speech module must be synchronised to the face expression module. Another thing to keep in mind is the opening/closing time of the mouth of the iCat while talking. This should also be synchronised with the text-to-speech module.

4. Testing

The iCat is delivered with a library with standard face expressions. However, the face expressions must be tested on usability. Although they have been tested before, they may have to be fine-tuned and more custom expressions can be added to the library.

After testing all parts of the program with the test user, conclusions will be made what direction future work can take.

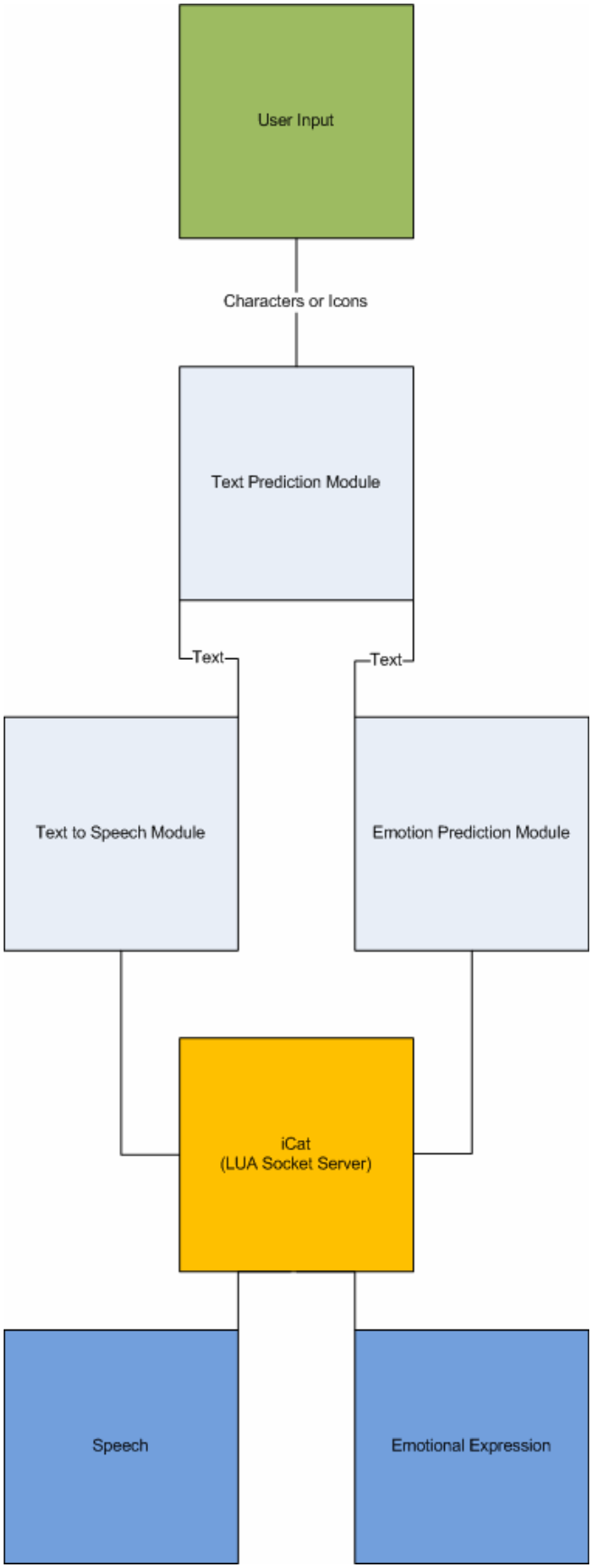
All parts of the program must be designed and implemented in a way they can communicate with each other. The OPPR software from Philips is designed to let C++ modules communicate via inputs and outputs. Because the iCat of Philips comes with no built-in software technologies in it, working together with other students and sharing components is important.

Other technologies like text to speech synthesizer must be implemented as modules for the iCat to function normally. These modules must be compiled and integrated in the Dynamic Module Library (DML) of the iCat.

Tasks

At the beginning of the project, the following tasks have been defined:

- Literature review
- User requirements study
- Documentation and evaluation of the project work


User Input
Characters or Icons
Text Prediction Module
Text
Text
Text to Speech Module
Emotion Prediction Module
iCat
(LUA Socket Server)
Speech
Emotional Expression

Background theories

n-grams

An n-gram **[6]** is a sub-sequence of n items from a given sequence. N-grams are used in various areas of statistical natural language processing and genetic sequence analysis. The items in question can be letters, words or base pairs according to the application.

Basically n-grams can be used to count the frequency of characters in a text. By counting all possible transitions and putting the results in an array, it is possible to predict the next character by looking for the highest possibility in the array.

	a	b	c	D
a	4	1	3	0
b	2	0	0	0
c	3	0	2	1
d	2	0	0	0

Above example presents the following transitions:

a -> a 4
a -> b 1
a -> c 3
…etcetera

But n-grams are not restricted to be used for characters. All kinds of object can be connected to the frequency of the transitions. For example, the following sentence can be converted to tri-grams: The man walks over the veranda.

The tri-grams would be: “the man walks”, “man walks over”, “walks over the”, “over the veranda”. N-grams are being used for example for extracting features for clustering large sets of satellite earth images (Google Earth) and for determing what part of the Earth a particular image came from.

Formula for n-grams:

$$P(W_n|W_{n-1}) = \frac{P(W_{n-1}, W_n)}{P(W_{n-1})}$$

iCat

The iCat Research Platform **[3]** is a research platform for studying human-robot interaction. It consist of 3 parts:

- The iCat robot
- The Open Platform for Personal Robotics (OPPR) software
- The iCat Community **[4]**

The OPPR platform can be compared to an operating system (like Windows XP). It integrates the individual cognitive software components that make up the iCat software system. This framework allows a component-based software setup and provides runtime flexibility, which means that during runtime the system behaviour can be probed and software modules can be added or removed. This greatly speeds up the development of new applications of the iCat.

Philips provides a special website dedicated to the iCat Research Platform. The website provides the infrastructure for supporting an online research community. The website contains information about the iCat, software updates and a discussion forum. The goal of the community is to exchange experiences with the iCat Research Platform, brainstorm on new iCat projects or modifications, track bugs and benchmark applications.

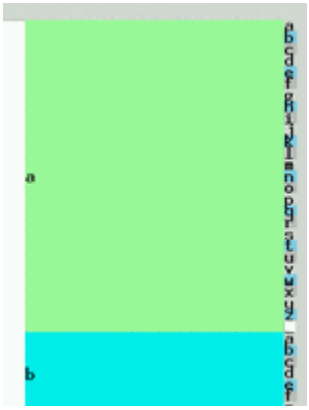
Although the iCat software would be ready during my project, it was not flexible enough to work with. The main problem was the architecture of the iCat. Although Philips declared it was .NET compatible, in the first release of the OPPR software it was not. It could be solved by building a client/server application with the LUA scripting language, but the big disadvantage was the latency between the two main applications. While the server (iCat) would be polling for each 500 ms, the client would send out 2 streams to the server:

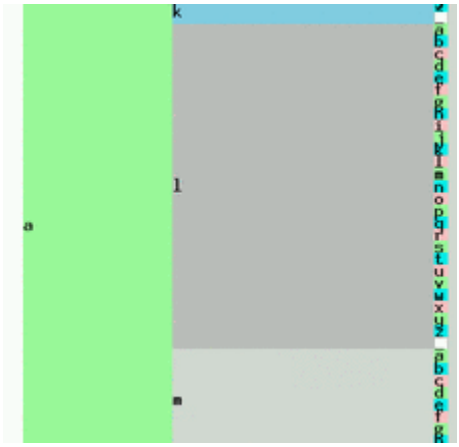
- Text-2-speech stream
- Facial expression stream

The Lip sync feature was not available yet, so while the speech was being played by the iCat soundcard, the lips should have been animated by LUA scripting. Because of the 500 ms polling latency, it never would be played synchronised.

In future Philips will release a new version of the OPPR software which features lip sync and possible .NET support.

Dasher





Dasher **[2]** is an information-efficient text-entry interface, driven by natural continous pointing gestures. Dasher is a competitive text-entry system wherever a full-size keyboard cannot be used. Dasher is a zooming interface. The world into which the user is zooming is painted with letters, so that any point you zoom in on corresponds to a piece of text.
Prediction of a language model is used to make the interface more efficient. Probable pieces of text are given more space, so they are quick and easy to select. Dasher learns from the input of the user all time. If the user writes a new word once, the next time it will be easier to write again.

Icons

Icons are used in almost every application. They can represent multiple words or sentences with one image. When combined with other icons they can

The major advantage of using icons as user input is the little time it needs to generate words. On the other hand, it is not as flexible as generating words with characters. The user can generate less possibilities because of the limited set of icons.

Word Prediction

When using dictionaries for looking up a word most of the time it will result in delay in the conversation. This was a disadvantage of the 3dphrazes application. Therefore a word/sentence prediction system was needed.

Word prediction is often used in writing support devices, for instance: the T9 system on a mobile phone. It speeds up the text input and quality of the spelling and syntax. Fewer text input from the user is needed to write complete words and sentences.

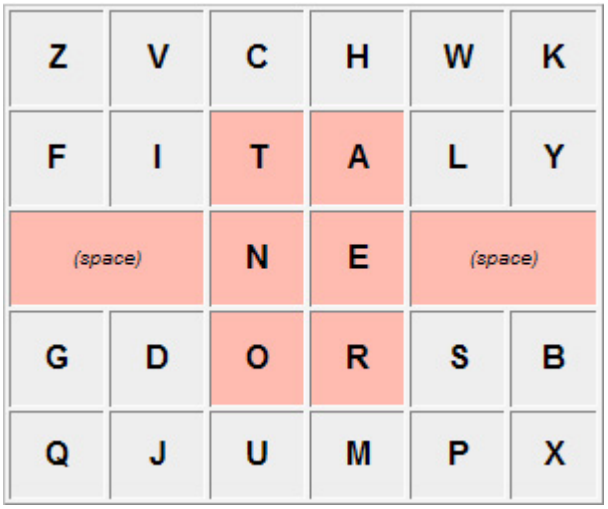
Fitaly

Z **20**	**V** **77**	**C** **230**	**H** **415**	**W** **138**	**K** **49**
F **176**	**I** **551**	**T** **701**	**A** **615**	**L** **319**	**Y** **133**
(space) **1741**		**N** **550**	**E** **976**	*(space)* **1741**	
G **147**	**D** **305**	**O** **590**	**R** **497**	**S** **497**	**B** **110**
Q **20**	**J** **20**	**U** **210**	**M** **187**	**P** **150**	**X** **20**

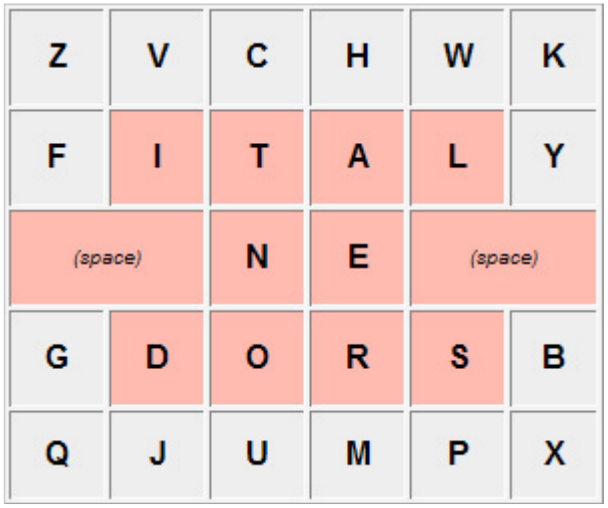
Fitaly keyboard with character frequencies

The Fitaly keyboard **[7]** was developed for minimizing pen or finger travel on a pen computer or a computer with a touch screen. It solves the ineffeciency of the QWERTY keyboard which forces the user to move his or her finger over the keyboard.

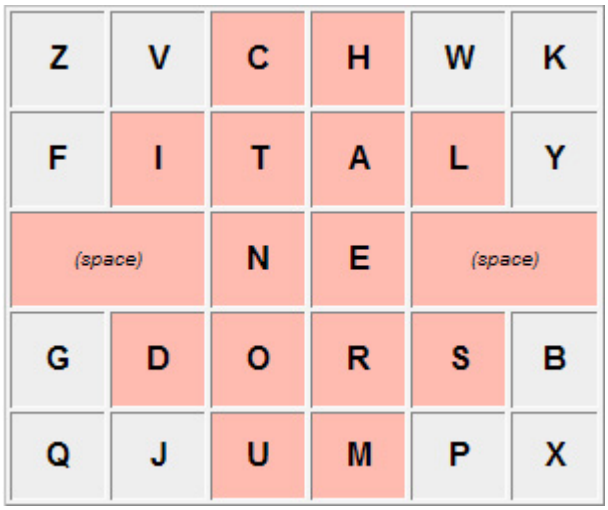
Fitaly uses characters frequencies in a circle. It starts with the character “E” which has a frequency of 976 in the English language. Next is the character “A”.



The six letters of the coloured area represent a combined frequency of **39.3%**.
Together with the space keys it represents **56,7%**.



The ten letters of the coloured area plus the space characters represent a combined frequency of **73.4%**. Close to three quarters of all keystrokes will happen in this coloured space.



With the addition of the letters **CH** and **UM** it results in a combined frequency of **84%**.

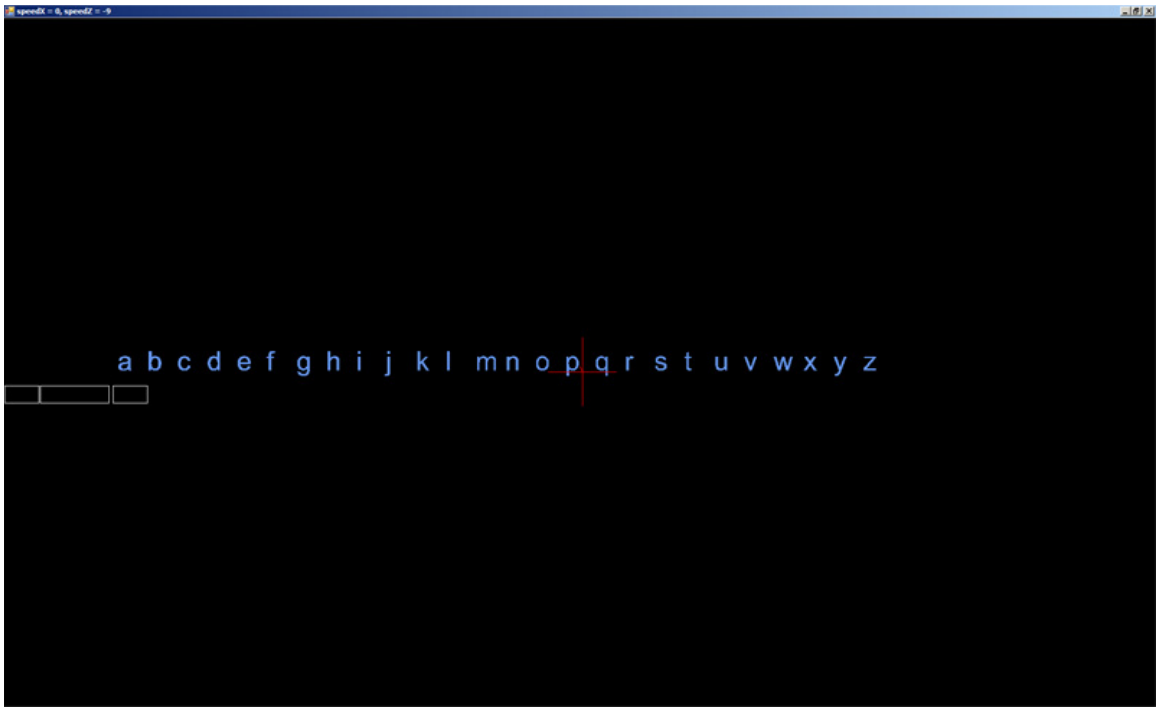
The following can be concluded:
More than half of the keys will be one away from the most used center keys N and E and 84% of all keys will be two away.

Fitaly with word prediction combination

When the fitaly keyboard will be rearranged every time the user presses a key, the most frequent transition will be placed in the middle of the keyboard. This helps the user to keep the most used keys in the center of the screen which results in time efficiency. Normal experienced users who are used to qwerty keyboards will not get good results in a short time, but for disabled people this can be a good solution besides programs like Dasher.

3d Phrazes Application

During the project a prototype of a working three dimensional dasher clone was made.
Its purpose was to find a way to optimize the possibilities of Dasher.



The 3d Phrazes application works in a different way. Movement to characters is three-dimensional instead of two-dimensional in Dasher. Most frequent characters appear in the middle of the screen when the first character is selected. The user can select characters by moving the mouse to the upper screen. This will move towards the characters and changing the focused character to red. When close enough, the character will be selected and the following character frequencies will be calculated out of the database with this query:

```
SELECT LEFT(LOWER(wordText)," + (prefix.Length + 1) + ") as woord,
SUM(wordFrequency) as freqTotal FROM `Word` WHERE LEFT(wordText," +
(prefix.Length + 1) + ") LIKE \"" + this.prefix + "%\" GROUP BY
LEFT(LOWER(wordText), " + (prefix.Length + 1) + ") ORDER BY freqTotal
DESC, woord ASC LIMIT 0, 30
```

The 3d Phrazes application works with word dictionaries. It selects word parts of most frequent used words. For example:

The user selects the character **A**.
The application will search in the database for words beginning with the character **A**.

Following words could be found:
Apple [4], After [2], Anonymous [1], Anger [0]

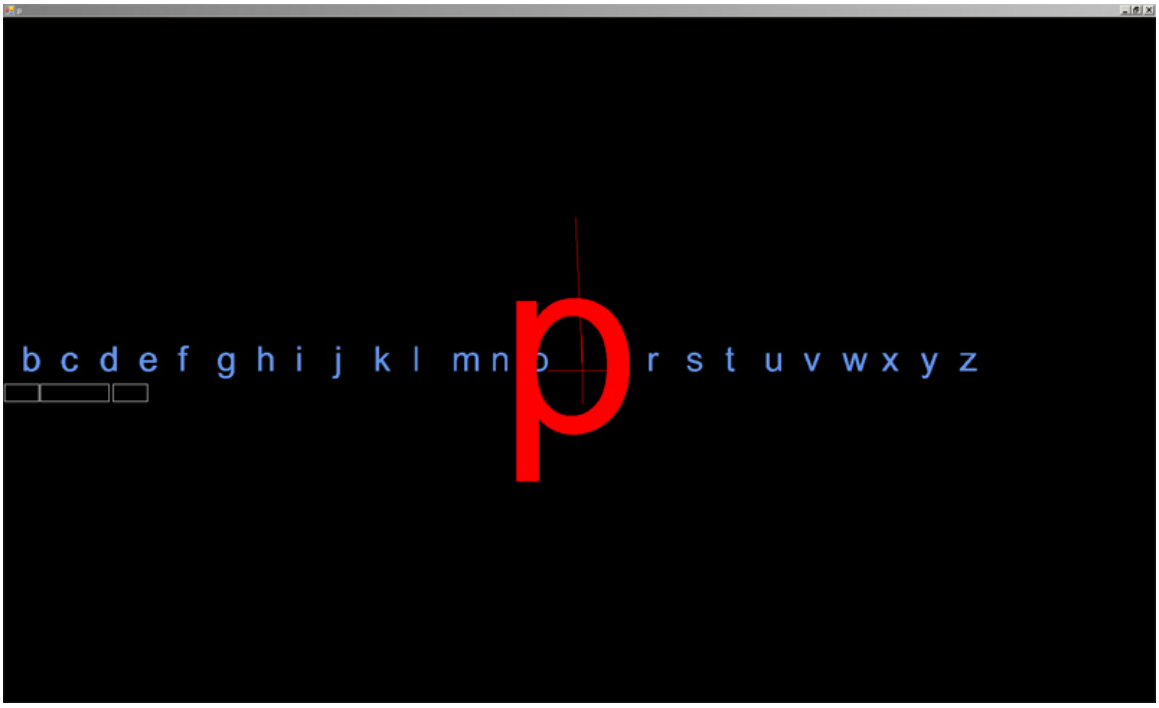
The frequency of the word *Apple* seems to be *4*, so this word part will be positioned in the center of the screen with the purpose to be easier to select for the user. This will make it faster to select most used words.

After selecting the character **A** the character line will contain the following words:

Anger – **A**fter – **A**pple – **A**nonymous

The mouse cursor will be located near the word **Apple**, because this is the most used word in the dictionary.

The user can go back one character by moving the mouse cursor downwards the screen.
When enough characters are selected, the text can be send to the text-to-speech engine by holding the mouse cursor above the middle button which is positioned left of the screen.
The text-to-speech engine is the same as in the Fitaly Interface application.
With the most left button, the user can load different databases with word dictionaries.

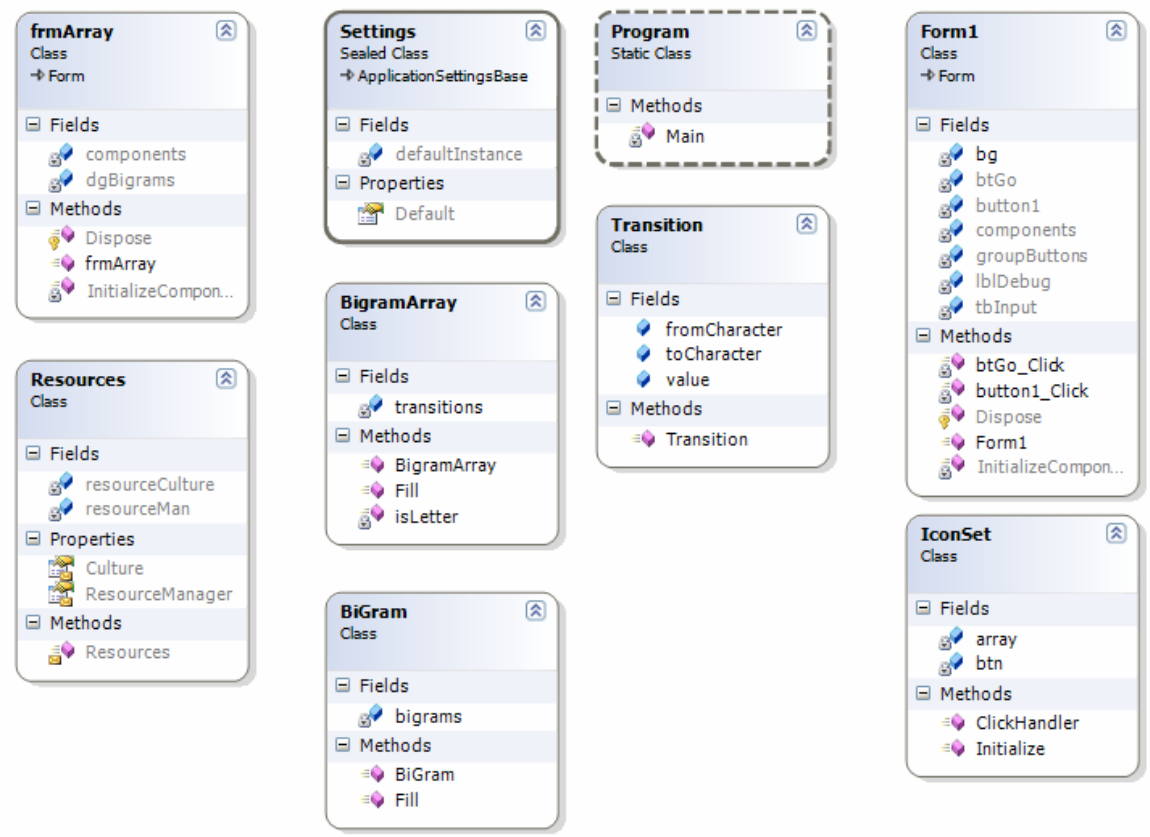


After programming this prototype it was clear it would not work as fast as the Dasher program, because for zooming in three-dimensional, more effort from the user is needed. The scrolling speed can be changed, but after some testing it was always slower than the Dasher interface.

For that reason we switched over to the Fitaly Interface in this project.

Design

Static Model

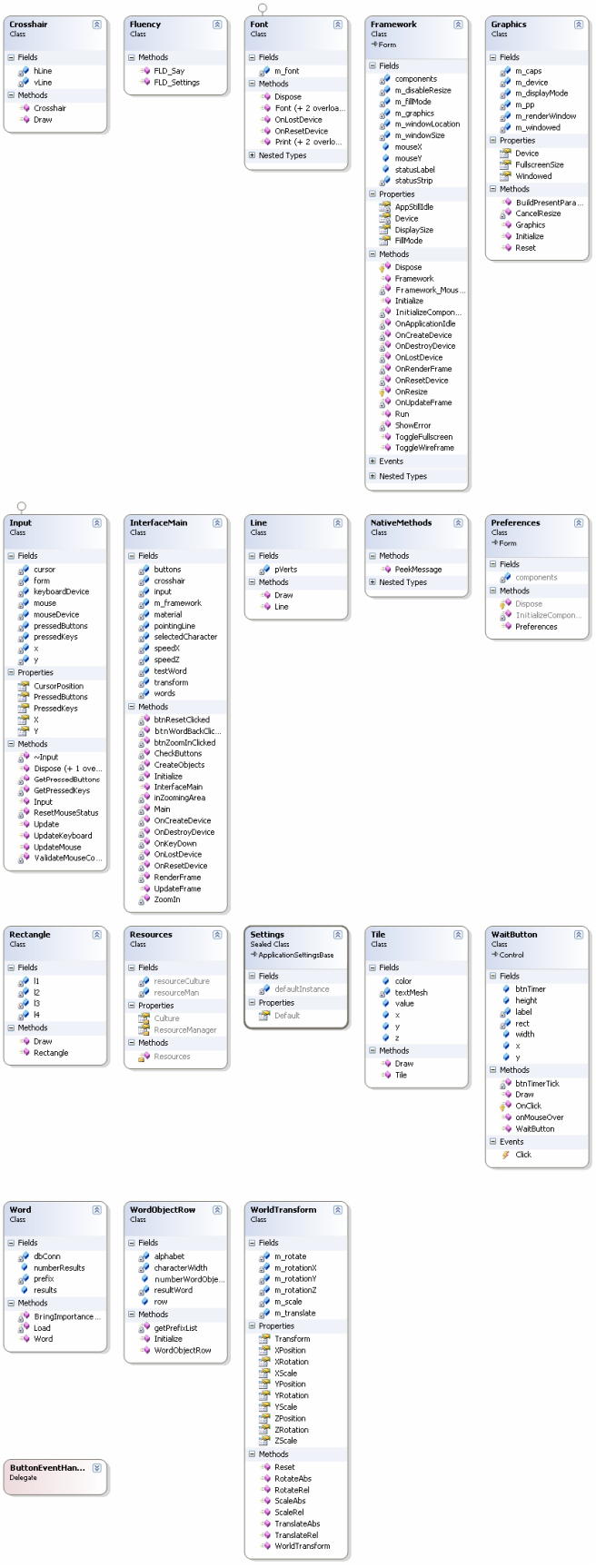


Class Diagram of Fitaly Interface **1**

The class diagram of the fitaly interface consists of:

- The main form, the main class which
- Settings class, which is used to load and save the used settings of the user
- BiGram class, a class that functions as the object of a bigram. It contains
- BiGram array. It handles the sorting and calculation of the bigrams.
- Resources class, which manages all resources used by the program
- IconSet class. This class loads up all used icons

NB: During my project the class diagrams differed constantly because of the changing destination of the project. In above class diagram the classes of the iCat functions are not visible. These classes were used to connect to the iCat via the LUA scripting language by socket connections.

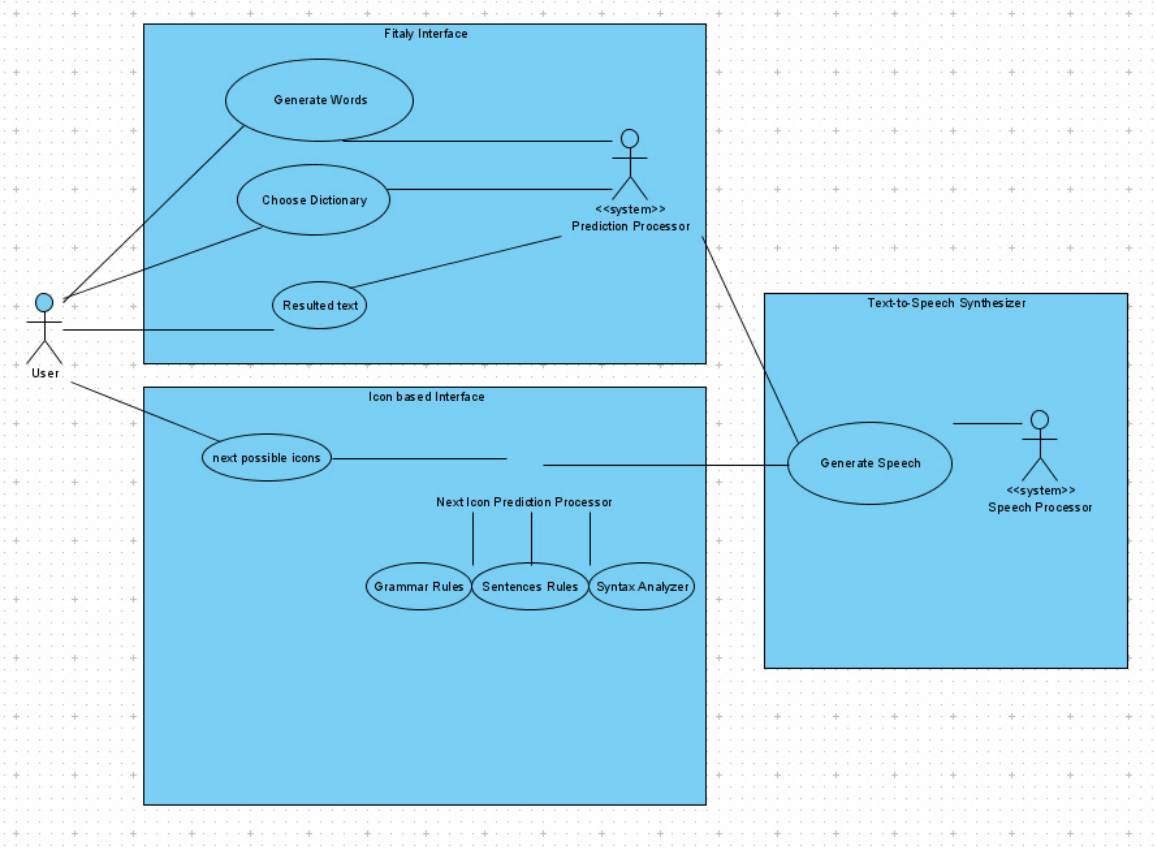

Crosshair
Class
Fields
hLine
vLine
Methods
Crosshair
Draw
Fluency
Class
Methods
FLD_Say
FLD_Settings
Font
Class
Fields
m_font
Methods
Dispose
Font (+ 2 overloa...
OnLostDevice
OnResetDevice
Print (+ 2 overlo...
Nested Types
Framework
Class
Form
Fields
components
m_disableResize
m_fillMode
m_graphics
m_windowLocation
m_windowSize
mouseX
mouseY
statusLabel
statusStrip
Properties
AppStillIdle
Device
DisplaySize
FillMode
Methods
Dispose
Framework
Framework_Mous...
Initialize
InitializeCompon...
OnApplicationIdle
OnCreateDevice
OnDestroyDevice
OnLostDevice
OnRenderFrame
OnResetDevice
OnResize
OnUpdateFrame
Run
ShowError
ToggleFullscreen
ToggleWireframe
Events
Nested Types
Graphics
Class
Fields
m_caps
m_device
m_displayMode
m_pp
m_renderWindow
m_windowed
Properties
Device
FullscreenSize
Windowed
Methods
BuildPresentPara...
CancelResize
Graphics
Initialize
Reset
Input
Class
Fields
cursor
form
keyboardDevice
mouse
mouseDevice
pressedButtons
pressedKeys
x
y
Properties
CursorPosition
PressedButtons
PressedKeys
X
Y
Methods
~Input
Dispose (+ 1 ove...
GetPressedButtons
GetPressedKeys
Input
ResetMouseStatus
Update
UpdateKeyboard
UpdateMouse
ValidateMouseCo...
InterfaceMain
Class
Fields
buttons
crosshair
input
m_framework
material
pointingLine
selectedCharacter
speedX
speedZ
testWord
transform
words
Methods
btnResetClicked
btnWordBackClic...
btnZoomInClicked
CheckButtons
CreateObjects
Initialize
InterfaceMain
inZoomingArea
Main
OnCreateDevice
OnDestroyDevice
OnKeyDown
OnLostDevice
OnResetDevice
RenderFrame
UpdateFrame
ZoomIn
Line
Class
Fields
pVerts
Methods
Draw
Line
NativeMethods
Class
Methods
PeekMessage
Nested Types
Preferences
Class
Form
Fields
components
Methods
Dispose
InitializeCompon...
Preferences
Rectangle
Class
Fields
l1
l2
l3
l4
Methods
Draw
Rectangle
Resources
Class
Fields
resourceCulture
resourceMan
Properties
Culture
ResourceManager
Methods
Resources
Settings
Sealed Class
ApplicationSettingsBase
Fields
defaultInstance
Properties
Default
Tile
Class
Fields
color
textMesh
value
x
y
z
Methods
Draw
Tile
WaitButton
Class
Control
Fields
btnTimer
height
label
rect
width
x
y
Methods
btnTimerTick
Draw
OnClick
onMouseOver
WaitButton
Events
Click
Word
Class
Fields
dbConn
numberResults
prefix
results
Methods
BringImportance...
Load
Word
WordObjectRow
Class
Fields
alphabet
characterWidth
numberWordObje...
resultWord
row
Methods
getPrefixList
Initialize
WordObjectRow
WorldTransform
Class
Fields
m_rotate
m_rotationX
m_rotationY
m_rotationZ
m_scale
m_translate
Properties
Transform
XPosition
XRotation
XScale
YPosition
YRotation
YScale
ZPosition
ZRotation
ZScale
Methods
Reset
RotateAbs
RotateRel
ScaleAbs
ScaleRel
TranslateAbs
TranslateRel
WorldTransform
ButtonEventHan...
Delegate

Class diagram of 3d phrazes application 1

The 3d Phrazes has a lot more classes than the Fitaly Interface program, because the use of a three dimensional interface. A special Framework class had to be built to control the 3d world. The classes Word and WordObjectRow deliver the connection between the application and the word dictionary.

Implementation

In this chapter, we present the implementation of the prototype of our fitaly interface.
The architecture is explained in the first part of this chapter. After that more technical design is explained by UML designs.



The text-to-speech synthesizer will read aloud the output of the interface.

The Fitaly Interface is divided in three parts:

- The interface itself
- The text-to-speech synthesizer
- The library with the saved knowledge of previous sessions with the interface

The interface was implemented with Visual C# Express Edition and runs in the .NET 2.0 framework. The text-to-speech synthesizer is external software. The application uses a .NET function to import the DLL file of the text-to-speech synthesizer in realtime and makes calls to it to perform text-to-speech. The language of the text-to-speech engine is dutch because the test user in this project is dutch. Currently it is not possible to connect a different text-to-speech engine to the system, but it is not difficult to extend the program with such a library.

The main interface

The main interface draws the initial Fitaly keyboard layout:

As we have explained in chapter *Background Theories*, the next layout of the keyboard will be calculated with bigrams. Those bigrams are calculated from a source text and stored in a two-dimensional array. A different source text can be loaded from the main interface. This source text must be a plain text file. When a different file is loaded, the fitaly keyboard is reset to its initial position.

One of the advantages of the Fitaly Interface is its multi language usage. It can read for example a spanish text file and the keyboard will be rearranged in an optimal spanish keyboard.

Usability Evaluation

In this project we have tested the following applications:

- Dasher Interface
- 3d Phrazes Interface
- Fitaly Interface
- Icon Language Interface (LINGUA)

Both Dasher and LINGUA deliver the fastest way of input text. LINGUA can be faster sometimes, but it has a limited set of words. The 3d Phrazes Interface works too slow when searching in the word dictionary. Combining it with trigrams and/or icons could be a solution.

Currently the working prototype of the Fitaly Interface only works with characters and with a very limited set of icons. In future this could be combined with more icon sets, so the user has more communication possibilities. One remark: The natural language processor is something difficult to develop. Maybe it will be possible to connect this Fitaly Interface with an universal icon language. (would be something like “Esperanto”, but with icons).

Conclusion

While testing the Interface we concluded the system with the zooming interface and a word frequency list doesn't work fast enough to deliver competent results. This is a major issue in the project which has to be solved for getting good results while communicating with other people. If the user of the Interface needs to much time for choosing the words the iCat has to speak up loud, the commucation with the other user will be too slow to have a normal conversation.

Therefore the Interface must be redesigned for faster user response. After a short research on the internet an alternative method was found. For people who can use only one hand for controlling a keyboard or pda users there is a simple solution called *Fitaly keyboard*.

This fitaly keyboard is optimized in a way the most used characters are in the centre of the keyboard.

For disabled people this type of keyboard enhances their typing speed, but for all other purposes (read: normal keyboards) it would work much faster. Only problem is, therefore people need to learn their typing all over again.

Future Work

The main goal of this project is to explore the possibilities of better user interfaces of communication applications for disabled people. It will be likely these possibilities will also be used for normal devices. In future with new technology current solutions like the T9 word library on mobile cellphones are not sufficient enough. Fitaly keyboards could be a faster way for user input on mobile cellphones.

The big disadvantage is the difference for the end user. The user has to learn and practice a new way of input. Another disadvantage of an icon language interface is the usage of memory for storing the learned prediction of the icons and natural language processing.

The new iPhone is a good example of using icons, but it lacks the use of natural language processing. Text messages still have to be typed in via a normal QWERTY keyboard, although this keyboard is a touchscreen one.

Samsung delivers a new type keyboard which works with grouping of characters. Measurable time could be achieved when the layout of the keyboard would be changed to Fitaly.

Following can be concluded:

All new technologies are designed on the old qwerty keyboard. Much more typing speed could be achieved when fitaly keyboards are being used. Especially when a fitaly keyboard with bi- or trigrams is used.

References

[1] An Icon-Based Communication Interface on a PDA, Siska Fitrianie, ISBN 90-444-0387-7, 2004

Links:

[2] The Dasher Project, http://www.inference.phy.cam.ac.uk/dasher/, 2006

[3] Philips Research Technologies – Robotics, http://www.research.philips.com/technologies/syst_softw/robotics/index.html, 2006

[4] iCat Research Community, http://www.hitech-projects.com/icat/, 2006

[5] Bigrams page on Wikipedia, http://en.wikipedia.org/wiki/Bigram, 2006

[6] N-grams page on Wikipedia, http://en.wikipedia.org/wiki/N-gram, 2006

[7] The Fitaly One-Finger Keyboard, http://www.fitaly.com/fitaly/fitaly.htm, 2005

[8] Romich, B.A., & Hill, K.J., *A Rate Index for Augmentative and Alternative Communication,* http://www.aacinstitute.org/Resources/MethodsandTools/2002rateindex/paper.html

Appendix A: Source Code

Following source code describes how the Fitaly Keyboard is rearranged in realtime:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace BiGramsTest
{
    class KeyboardHandler
    {
        private Button[] buttons;
        private Form mainForm;
        private TransitionArray ta;
        private String currentTransition;
        private WordSuggestionTable wordTable;
        private ListBox lbWordSuggestion;

        public KeyboardHandler(Form mainForm)
        {
            this.mainForm = mainForm;

            lbWordSuggestion =
(ListBox)mainForm.Controls["lbWordSuggestion"];
            wordTable = new WordSuggestionTable(lbWordSuggestion);

            ta = new TransitionArray();
            LoadFile("tekst.txt",
(ProgressBar)mainForm.Controls["progressBar1"]);

            currentTransition = "";

            GroupBox gb = new GroupBox();
            gb = (GroupBox)mainForm.Controls["GroupButtons"];

            buttons = new Button[29];

            int y = 24;
            int x = 16;

            int j = 1;

            for (int i = 0; i < buttons.Length; i++)
            {
                buttons[i] = new Button();

                if (i == 26)
                    buttons[i].Text = ".";
                else if (i == 27)
                    buttons[i].Text = ",";
                else if (i == 28)
```

```
                    buttons[i].Text = " ";
                else
                    buttons[i].Text = "" + (Char)(i+97);

                buttons[i].Location = new System.Drawing.Point(x, y);
                buttons[i].Size = new System.Drawing.Size(25, 25);
                buttons[i].Tag = i;

                buttons[i].Click += new
System.EventHandler(ClickHandler);

                x += 35;
                j++;

                if (gb != null)
                {
                    gb.Controls.Add(buttons[i]);
                }

                if (j > 6)
                {
                    x = 16;
                    y += 35;

                    j = 1;
                }
            }
        }

        public void LoadFile(string filename, ProgressBar progress)
        {
            try
            {
                if (ta != null)
                {
                    FileStream file = new FileStream(filename,
FileMode.Open, FileAccess.Read);
                    StreamReader sr = new StreamReader(file);
                    string s = sr.ReadToEnd();

                    ((TextBox)mainForm.Controls["tbInput"]).Text = s;
                    ta.Fill(s, progress);
                    wordTable.text = s;
                    wordTable.Init();
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        public void ClickHandler(Object sender, System.EventArgs e)
        {
            string input = ((Button)sender).Text.ToString();

            if (input.Equals(" "))
```

```
                input = "|";

            TextBox tbOutput =
(TextBox)mainForm.Controls["tbKarakter"];
            TextBox tbSuggestie =
(TextBox)mainForm.Controls["tbSuggestie"];

            currentTransition = ta.getTransition(input[0]);
            tbSuggestie.Text = currentTransition;

            if (tbOutput != null)
            {
                tbOutput.Text += ((Button)sender).Text.ToString()[0];
            }

            RearrangeButtons();

            lbWordSuggestion.Items.Clear();
            wordTable.getWordSuggestion("test");
        }

        private void RearrangeButtons()
        {
            string karakter = "";

            try
            {
                string[] karakters = currentTransition.Split('-');

                for (int i = 0; i < buttons.Length; i++)
                {
                    if (karakters[i] != null)
                    {
                        if (karakters[i].Equals("|"))
                            karakter = " ";
                        else if (karakters[i].Equals("{"))
                            karakter = ".";
                        else if (karakters[i].Equals("}"))
                            karakter = ",";
                        else
                            karakter = karakters[i];
                    }

                    buttons[i].Text = karakter;
                }
            }
            catch (Exception ex)
            {
            }
        }
    }
}
```

Following source code describes the initializing of the bigram by reading a plain text file:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows.Forms;

namespace BiGramsTest
{
    class TransitionArray
    {
        private biGram[] array = new biGram[29 * 29];

        public TransitionArray()
        {
            int i = 0;
            int fromChar = 97;

            while (i < array.Length)
            {
                for (int j = 0; j < 29; j++)
                {
                    string transition = "" + (Char)(fromChar) +
(Char)(j + 97);
                    array[i] = new biGram(transition, 0);
                    i++;
                }

                fromChar++;
            }
        }
        private void addTransition(string transition)
        {
            for (int i = 0; i < array.Length; i++)
            {
                if (array[i].transition.Equals(transition))
                    array[i].weight += 1;
            }
        }

        public string getTransition(char fromCharacter)
        {
            string ret = "";

            biGram[] tmpArray = new biGram[29];

            int offset = (((int)fromCharacter) - 97) * 29;

            for (int i = 0; i < 29; i++)
            {
                tmpArray[i] = array[offset + i];
            }

            Array.Sort(tmpArray);   // sorteer frequenties

            for (int i = 0; i < tmpArray.Length; i++)
            {
```

```
            ret += "" + tmpArray[i].transition[1] + "-";
        }

        return ret;
    }

    public void Fill(string text, ProgressBar progress)
    {
        text = text.ToLower().Trim();

        progress.Minimum = 0;
        progress.Maximum = text.Length;

        if (progress.Value >= progress.Maximum)
            progress.Value = 0;

        // count transitions
        for (int i = 0; i < text.Length - 1; i++)
        {
            if (isLetter(text[i]) && isLetter(text[i + 1]))
            {
                addTransition(text.Substring(i, 2));
            }

            if (progress.Value < progress.Maximum)
                progress.Value++;
        }

        /*
        // calculate the probability of each transition
        for (int i = 0; i < array.GetLength(0); i++)
        {
            float sum = 0;

            for (int j = 0; j < array[i].GetLength(0); j++)
            {
                sum += array[i][j];
            }

            if (sum > 0)
            {
                for (int k = 0; k < array[i].GetLength(0); k++)
                    array[i][k] = (array[i][k] / sum);
            }
        }
        */
    }

    private static float sum(float[] array)
    {
        float sum = 0f;

        foreach (float i in array)
            sum += i;

        return sum;
    }
```

```
        private bool isLetter(char letter)
        {
            bool ret = true;

            if (Convert.ToInt16(letter) < 97 || Convert.ToInt16(letter)
> 122)
                ret = false;

            return ret;
        }
    }
}
```