**Documentation of the Functions inserted and removed from Agora**

1. File :- RegleTAO.cpp

Function :- Executer (CDocument \*pDoc)

This function is used when user wants to export the results of Agora in the step 4 of the whole process. Within this function the main function which is responsible to do further detailed processing is

((CMyAppDoc\*)pDoc)->atelier->GenererDicoEtXML(type\_zone,seuil\_dico);

So this function takes 2 input that are

1) zone type

2) threshold

1. File :- Atelier.cpp

Function :- GenererDicoEtXML(type\_zone,seuil\_dico)

This function generates the dictionary of forms and XML files. The 2 inputs they are taking are

1)type of zone on which we need to process

2)the threshold for this zone

Within this function we have a list of the zone (liste\_de\_zones) which contains the list of all the zones within a page. So we take all the zones one after other. But we need to process only type\_zones

So we take blocks 1 after other.

CBloc \*bloc = new CBloc(z->get\_liste\_elements());

And if the bloc is not processed earlier we process the block

1. bloc->Decomposer\_en\_lignes();

After this function is completed we call the function

1. bloc->Rachit1();

After this function is completed we call the function

1. bloc->Rachit2();

After this function is completed we call the function

1. bloc->Creer\_structure\_XML(z->getRect(), pDoc, type\_zone);

Then we finally delete the created bloc to save memory.

Functions that are removed from this file are :-

1. bloc->Ordonner\_lignes();
2. ligne->Supprimer\_accentuation();
3. ligne->Calculer\_espacement\_moyen();
4. ligne->Decomposer\_en\_mots();

But the functions inserted does all the required work that has to be done by these already used functions.

1. File :- Bloc.cpp

Function :- Decomposer\_en\_lignes()

This is the main function which is responsible for Line Segmentation and Detection, Character Segmentation and Detection, Class Detection ,Sorting of lines within a bloc, Sorting of characters within a line.First of all we take the list of all the connected components within a bloc (liste\_elements), We arrange the whole list in the order of the minimum left x-coordinate using a temporary bloc blocTemporaire.

**Line Segmentation and building of Histogram :-**

Now we try to travel the whole block from the top to the bottom so as to make the histogram of the pixels in each line of the page. The purpose for this is that we want to use the property of text that there is in general some gap between 2 consecutive lines.

So we store it in an array pixelCount[]. Then we try to cut the lines based on the histogram. As soon as we get a minima after maxima (that is 0 pixel line) we end the line and as the pixel count starts again start the new line. In this way we store the

line\_height[i]; height of the line

line\_start[i]; starting point of the line

line\_end[i]; ending point of the line

Now we calculate the median of the height of the lines, so that we get a rough estimation that what would be the height of the ideal line in this page (threshold\_height). Then we compare the height of this line to all the other lines, so as to know whether the line is under segmented or over segmented or rightly segmented.

int low\_cut = threshold\_height - (threshold\_height/2);

int high\_cut = threshold\_height + (threshold\_height/2);

If a line is rightly segmented we do nothing.

If a line is highly under segmented it could be the case of a noise forming a line or a dot / accent forming a line. So a very small line could be neglected treating as a noise.

If a line is over segmented then we try to find how many lines are joined together. Now we try to separate these lines. So we again consult the histogram of pixels and try to figure out points that are at 1/3 and 2/3 height of the maximum height.

int threshold\_line\_div\_up = global\_max - (global\_max/3);

int threshold\_line\_div\_down = global\_max - (2\*global\_max/3);

so we store these points in

store\_check\_point\_up[];

store\_check\_point\_down[];

and traversing this curve we find the global minima(get\_min\_break\_point) which is considered as the separation point of the 2 lines. In this way we can separate multiple lines joined into 1 line.

Once we are done with this we are sure that lines are separated properly.

Line Segmentation is over.

**Line Detection and Class detection:-**

Now we traverse the whole liste\_elements(list of elements) for the purpose of assigning a line number and a class number to each individual connected components.

**Class Number**:-

1. Normal Character
2. Problematic Character
3. Un-merged Accents
4. Punctuation
5. Broken Character
6. Merged Accents (normal character + accents)
7. Accents that got merged.

So we follow a set of rules while deciding which line/ class a particular connected component belongs.

**Line:-**

1. If a CC lies completely within a line it belongs to that line.
2. If a CC lies majorly in 1 line and slightly in other line it belongs to 1st line(either upper line or lower line)
3. If it lies majorly to more than 1 line it is a Problematic character (with lower line assignment)

**Class:-**

1. If a CC lies completely within a line it is normal character (1)
2. If it lies only in the upper half of the line it is an accents (3)
3. If it lies only in the lower half of the line it is a punctuation(4).
4. If it lies in multiple lines it is a problematic charcter(2).
5. If it lies completely in a line but satisfy the property of accents and punctuation partially it is a broken character (5)
6. Merged Accents(6) and Accents got merged(7) will come later after merging of accents are done.

So we set the line number and the class number of the elements/CC by these.

e->SetLineNumber(i+1);

e->SetClassNumber(1);

Line Detection and Class Detection is over.

**Formation and Sorting of Line (Ordonner\_lignes):-**

Now at this point we have a list of elements of a zone. Now is the time to form line out of this list. So we make list of lines (liste\_lignes). So we traverse the whole list and form lines based on the line number already assigned to them. Now we sort the lines with the help of temporary line l2

CLigne \*l2 = new CLigne () ;

So the sorting is done based on the line number of the CC. Once the sorting is done we delete this variable to save memory.

Lines are formed and are sorted in liste\_lignes.

**Sorting of Character within a line:-**

Now we have each individual lines with their elements within them. Now we need to sort these elements within the line. So based on the minimum left x coordinate we sort the elements within the line.

if(e->getRect()->left < min && e->GetCharNumber()== 0)

{

min = e->getRect()->left;

s = e;

p9 =p2;

}

Hence the characters are sorted within the line.

**Character Detection:-**

Now we need to assign the character number to each individual elements so based on the minimum left x coordinate we assign each element the character number (left to right) within the line.

s->SetCharNumber(i);

Hence the characters are assigned charcter number as well.

Now we say that the processing of a zone is done so we turn the Boolean value true.

decompositionEnLigneEffectue = true;

Hence we came out of the function bloc->Decomposer\_en\_lignes() back to the function GenererDicoEtXML(type\_zone,seuil\_dico) in atelier.cpp. Now the next function called is Rachit1() in bloc.cpp.

1. File :- Bloc.cpp

Function :- Rachit1()

This function is responsible for the merging of accents. Now we are with liste\_lignes which contains the list of the lines. So we travel the whole list from line 1 to last line and whenever we find an element with class number 3 (accents) we try to know whether it belongs to any of its nearby characters. Merging of accents is not an easy tasks and requires a lot of rules, hence after a lot of analysis we came up with quite a lot of rules for merging.

**Merging of Accents:-**

There are many rules for the merging of Accents but major ones are:-

1. If it lies completely over 1 character and within the width of that character they are merged.
2. If it lies completely over 1 character and partially within the width of that charcter but not over any other character they are merged.
3. If it lies completely over 1 character and within the width of that more than 1 character but majorly on 1 they are merged.
4. If it lies completely over 1 character and but not within the width of any of the character it is left alone(transferred to broken character).

e\_next->SetRect(lmin,ltop,rmax,rbottom);

flag = 1;

e\_next->SetClassNumber(6);

e->SetClassNumber(7);

Hence the rectangle of the character with which it is merged is updated. The class number of this character is updated to 6 (character + accents). The class number of the accent is updates to 7 (accents merged to someone).

Hence merging of accents is finished.

Now the function Rachit1() comes to an end so we came back to the function GenererDicoEtXML() in Atelier.cpp file. So the next function to be called within this function is Rachit2() which is in file bloc.cpp .

1. File :- Bloc.cpp

Function :- Rachit2()

This function is responsible for Word Segmentation and Word Detection. So now we are in the situation where we have list of lines within liste\_lignes. And each individual line is sorted within itself. So all we have to do is to find where 1 word ends and where the other word starts. So this is done using the property of the text that after each word there is a gap before the starting of the next word, but the problem is that there is gap within a word between characters.

So we try to solve this problem with the help of 2 gaps.

1. Gap between rectangular block of CC
2. Gap between the actual black pixel(letter) on the middle line of the element.

So for gap1 we need to find the average width between the characters within the line. Then threshold = x\*average ;

So we calculate the average between the line and hence we get threshold taking x = 2.

Now so as to calculate gap2 we need to find the middle line of each element then we need to travel in the image(left to right for ending point and right to left for starting point) to find the 1st black pixel and to store this black pixel as the starting and ending points.

Creation of the image.

CImage \*img = new CImage();

If we get the point we set it in element

e->SetXLM(i);

If we don’t get any black pixels we set its midpoint as the one.

e->SetXLM((e->getRect()->left + e->getRect()->right)/2);

Now we calculate average and threshold

average = (double)total /(double)count;

int threshold = (int) 1.9 \* average;

Now we calculate the Gaps

gap1 = (e\_next->getRect()->left) - (e->getRect()->right) ;

gap2 = e\_next->GetXLM() - e->GetXRM() ;

Now we start applying rules for word segmentation and detection.

Firstly we identify the punctuations in the line as we know that punctuations separates the words. So when we encounter we increases the word count.

If the gap between 2 character is less than threshold it is within 1 word otherwise it may belongs to the other words.

So we further check it with modified threshold, with gap2 as if the words are near enough they are taken in same word otherwise they are counted in separate words.

e->SetWordNumber(wordcount);

wordcount++;

In this way word segmentation and detection is done.

Now the function Rachit2() comes to an end so we came back to the function GenererDicoEtXML() in Atelier.cpp file. So the next function to be called within this function is Creer\_structure\_XML() which is in file bloc.cpp .

1. File :- Bloc.cpp

Function :- Creer\_structure\_XML(z->getRect(), pDoc, type\_zone);

This function is responsible for the creation of the XML files and the creation of the various folders for different zones (image, text etc). So we create folders and files to keep the data of various zones into it.

fichier.Format ("%s\\%s\_FIN", ((CMyAppApp\*)(AfxGetApp()))->projet->getParamProj()->getM\_repResultats(), type\_zone);

\_mkdir (fichier);

Now we create the XML files for the text zone with the various details of the elements. We can even add or subtract the output data of these elements.

fprintf (fp, "\t\t\t<cc value=\"%d\">%d,%d,%d,%d,%d,%d,%d</cc>\n", fichier\_CC, e->getRect()->left, e->getRect()->top, e->getRect()->right, e->getRect()->bottom, e->GetClassNumber(), e->GetXLM(), e->GetXRM() );

So we create the XML files for the whole zone at a time. The format is left, top, right, bottom, classnumber, gaplenth1, gaplength2.

fichier\_CC = ((CMyAppApp\*)(AfxGetApp()))->projet->getLaboProj()->liste\_permanente->Similaire (e, pDoc);

Using the function Similaire we try to compare the each individual CC so as to know which cluster does it belongs to.

1. File :- MyAppView.cpp

Function :- OnMouseMove(UINT nFlags, CPoint point)

This function OnMouseMove is used to handle the events that are generated due to the motion of the mouse. Once the process ends we can put over mouse pointer over the element of the image and it will tell the various properties (linenumber, wordnumber, charnumber, classnumber coordiates) etc.

pDoc->atelier->liste\_de\_zones->CherchePoint(&cherche,ecran,m\_zoomcoeff);

It calls the function CherchePoint() which is in file ListeZones.cpp

1. File :- ListeZones.cpp

Function :- CherchePoint(CPoint \*point,CDC \*ecran, double zoomcoeff)

This function is responsible so as to display the properties of an element on the status bar over which the mouse is moved. Due to the high complexity of the search and the frequent movement of the mouse I have used a pop up to click ok until then the OnMouseMove() function is disabled.

The format in which the properties will be displayed is:-

message.Format("line %d|word %d|char %d|class %d|left %d|right %d|top %d|bottom %d|Rachit Agrawal",

e->GetLineNumber(),

e->GetWordNumber(),

e->GetCharNumber(),

e->GetClassNumber(),

e->getRect()->left,

e->getRect()->right,

e->getRect()->top,

e->getRect()->bottom);

This message is displayed on the status bar of the main page.

((CMainFrame\*)AfxGetMainWnd())->ChangeStatusText(0,message);

The message which is used as the pop up to block the OnMouseMove function is:-

AfxMessageBox("See Status Bar !");

I think that is the total amount of modification that is done so as to add new functionalities to the software, and to make it more handy and user friendly. A lot of work is done in saving the memory throughout the code (in various other functions).

**Future Modifications/Bugs in the Software:-**

1. **Removal of unwanted part in CC**:- As we know that each CC contains 1 character but during the formation of the rectangle it is seen that is also contains the unwanted part of some neighboring characters as well so we need to remove these unwanted part from area of the useful CC. Because this will effect the classification of the CC.
2. **Identification of the punctuation in parts**:- We have seen in the images that several times we come across the characters like(; : ! ?) which are separated into 2 different parts. So in general the upper part results into the accent and the lower part remains into a punctuation, so we can try to apply some logic to merge them.
3. **Comparison between two CC** :- Currently while making the cluster we are comparing the two CC by putting 1 CC over other and counting similar pixels and hence keeping it in same cluster or forming new cluster, but actually this seems to be a flawfull process as there is no point comparing 2 images or irregular size.
4. **Implementation of Word Segmentation** :- Word segmentation is one of the toughest part just because of the irregular size between the CC, so it is very tough to decide whether a CC belongs to same word of different word, which lead to merging to many words in 1 and separation of a word in many.
5. **Horizontal touching and Broken Characters** :- Due to bad fonts of the images we have seen that in a number of cases we have seen 2 CC joining together or 1 CC get broken down into 2 or 3 pieces, which while layout analysis is not a problem but is a big problem while the classification of the CC is done. So we need to implement some algorithm to separate them/ join them.
6. **Technical Problem**:- Problems regarding scenario storage and ratio (height/width ) are not seen. So if present need to be handled.

**Class Number Assignment:-**

1. Normal Character
2. Problematic Character
3. Accents (unmerged)
4. Punctuation
5. Broken Character
6. Merged Accents
7. Used Accents (now of no use)