UNIVERSITY OF THE PHILIPPINES MANILA
COLLEGE OF ARTS AND SCIENCES
DEPARTMENT OF PHYSICAL SCIENCES AND MATHEMATICS

DIBUHO

A COMIC STRIP CREATOR EMPLOYING CUBIC BEZIER PATHS AND THE STANDARD VECTOR GRAPHICS FORMAT

A Special Problem in Partial Fulfillment
of the Requirements for the Degree of
Bachelor of Science in Computer Science

Submitted by:
Ivan Ignatius Francisco Eugenio
April 2007

ACCEPTANCE SHEET

The Special Problem entitled “Dibuho: A Comic Strip Creator Employing Cubic Bezier Paths and the Standard Vector Format” prepared and submitted by Ivan Ignatius Francisco Eugenio in partial fulfillment of the requirements for the degree of Bachelor of Science has been examined and is recommended for acceptance.

Gregorio B. Baes, Ph. D (Candidate)
Adviser

EXAMINERS	Approved	Disapproved
1. Avegail D. Carpio, MS (Candidate)	_____	
2. Aldrich Colin K. Co, MS (Candidate)	_____	_____
3. Ma. Sheila A. Magboo, MS	_____	_____
4. Vincent Peter C. Magboo, MD, MS	_____	
5. Philip D. Zamora, MS (Candidate)	_____	_____

_____
Date

Accepted and approved as partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

Vincent Peter C. Magboo, MD, MS
Unit Head
Mathematical and Computing Sciences Unit

Alex C. Gonzaga, Ph. D
Chair
Department of Physical Sciences and Mathematics

Department of Physical Sciences and Mathematics

Reynaldo H. Imperial, Ph. D
Dean
College of Arts and Sciences

ABSTRACT

The Dibuho Comic Strip Creator is an application written in C++ employing cubic bezier curves and the standard vector graphics format. The simple program interface makes it possible for users to create comic strips with ease, without having to learn sophisticated drawing techniques. The application has an extensible library which can be extended by securing plugins from the developer.

The library contains vector images of characters, backgrounds and props which the user can insert in the panels of the comic strip. In addition, the application allows the user to insert speech balloons for character dialogues.

Moreover, the application has a facility for character creation, allowing them to create unique characters by combining body elements available in its library. Being entirely based on vector graphics, Dibuho allows flexible editting of the comic strip including posing of characters, changing of facial expressions, and general affine transformations such as rotation, translation, and scaling.

Keywords: Comic Strip, Cartoon, Bezier Curves, Vector Graphics, Affine Transformation

TABLE OF CONTENTS

I. Introduction

A. Background of the Study

Comic strips were said to have evolved from early hieroglyphs found in caves and tombs which are usually visual narratives of juxtaposed pictures depicting a scene or an event. The existence of words were at first not mandatory, but were soon added to give more information and to increase narrative flow. The earliest comic strip was from Germany titled “Max and Moritz”, a strip about two trouble making boys and is mainly moralistic in nature. Then came other titles such as “Yellow Kid”, the first color comic which appeared in the first Sunday comic section in 1897. [1]

The surge of superheroes as characters in comic books began in Cleveland with the first appearance of “Superman” in the cover of the first issue of Action Comics in 1938, drawn by Joe Shuster who co-created the character with Jerry Siegel. [2] Soon comic strips became a form of self-expression for artists and later were used as a tool to draw more customers into reading newspaper prints. In this regard, improvements in press and printing technology contribute significantly to the development of comic strips as a medium for expression of ideas. [3]

The role of comic strips today isn’t limited to entertainment. It had expanded to being informative, or even persuasive, in nature some of which

are written to give criticisms, convey certain advocacies, argue politically, or make a stand on national and global issues. [4] Comic strips are also utilized as a teaching tool to enhance learning in the formal classroom environment, as many people have realized that comic strips have a widespread appeal to all age groups and levels of society. In fact, comic strips and comic books are one of the most read media throughout the world. [5]

B. Statement of the Problem

The expansion of the area where comic strips are being utilized also increased the demand to produce them. Although there was no shortage of illustrators with new ideas to produce hundreds of comic strips, there exists people with great ideas but do not have the talent to put them into illustrations.

Many tutorials and self-help guides on how to draw are available both online and on print to resolve this issue [6], but the problem remains that not all people have the time, or even the ability, required to learn all the skills necessary. Tools such as Adobe Photoshop, Corel Draw and other graphics softwares that aims to aid illustrators in their work have expensive licenses and not all can afford them. Even those who can buy these tools cannot easily use them, for many of its functions are quite complex and would require a separate tutorial to be understood and be used effectively.

The time-consuming process of creating comic strips has been a hindrance into its effective utilization as a tool to successfully communicate to the readers, since creating comic strips requires many manual and painstaking processes that requires time and talent. These limitations should be overcome in order to realize the full potential of comic strips not just to entertain but also to inform, to criticize, or to express ideas.

C. Objectives

To create Dibuho, an application that facilitates character and comic creation with the following functionalities:

1. A user can create and edit a Character.
    a. The user selects its gender.
    b. The user selects facial parts to define the face.
    c. The user sets body attibutes to define body.
    d. The user dresses up the Character.
2. A user can create and edit a Comic Strip.
    a. The user may input the Title of the Comic Strip
    b. The user may input the Author of the Comic Strip
    c. The user may create, edit and delete Panels. For each panel, the user may do the following:
        i. The user may resize the Panels.
        ii. The user may add or delete the Background.
        iii. The user may add or delete Elements in the Panel.
            1. Characters can be placed in the Panel.
                a. Facial parts of Characters can be configured to change facial expression.
                b. The user may dress up the Character using clothing from the library.

c. The user may pose the Character to the desired position. The clothing of the character will follow shape of the body.

2. Speech Balloons can be placed in the Panel.
    a. Content of speech balloons can be specified.
    b. Speech Balloon type can be specified.
3. Props can also be placed in the Panel.

d. A toolbar will be available with the following functionalities for modifying the attributes of an object instance inside the panel.
    i. Elements can be moved inside the panel by dragging to obtain desired position in the panel.
    ii. Elements may be rotated clockwise or counter-clockwise to a certain angle.
    iii. Elements can be scaled to the desired size.
    iv. Elements can be ordered. (Bring to front, Bring to back)
    v. Elements already placed can be deleted.
e. The user can save the Comic Strip in a Dibuho File (*.dibuho) which can later be opened again for editting.
f. The user can export the Comic Strip as a flat image in PNG format (*.png).

3. The user can open and edit an existing Dibuho File (*.dibuho).

4. The user can update the database of images by securing a Plugin from the Developer and placing it in the plugins directory.

D. Significance of the Study

The Dibuho comic strip creator removes the hassle of manually creating comic strips panel-by-panel. It makes creation of comic strips more efficient in that the user doesn't have to draw his or her characters for every frame since the user may simply add instances of already saved custom characters. With this, the user may focus its efforts on other aspects of making a comic strip such as adding humor, writing the script or story, character development and others.

Dibuho enables anyone, even non-artists, to create their own comic strips using their own ideas without being hindered by their drawing skills. In this regard, anyone who has concepts can more easily express them, without having to learn complicated drawing techniques and tools, or asking somebody else who can draw to interpret his or her ideas. It provides a user-friendly and simplified interface for comic strip development that does not require technical expertise to be used.

With the ease of its use, Dibuho can be utilized generally by everyone for producing comic strips in little amount of time and without necessarily learning how to draw. In this regard, Dibuho realizes the full potential of comic strips by removing the limiting factors that hinders its effective utilization.

E. Scope and Limitations

1. The assortment of illustrations that Dibuho can produce is dependent on the contents of the library. The creation of these new files is outside the system.
2. The program does not support comic strip script.
3. Images generated are flat, meaning they cannot be rotated in 3D.
4. The program does not guarantee duplication of existing comic strips. The variety of illustrations that the program can produce is mainly dependent on the available library.
5. The export functionality is limited to the PNG image file format.
6. There are only three tools available in modifying the elements in the panel, namely, the rotate, scale and translate tools.

F. Assumptions

1. Dibuho assumes that the illustrator and script writer is the same, for it doesn’t have a facility for separate script input. A script is assumed to have been completed already prior to comic illustration.
2. It is assumed that the user already knows the dimension of the comic strip beforehand.
3. The characters involved in the comic strip are human. The application does not produce non-human characters.

4. Production of the plugins which adds elements to the library is outside the system and is completely up to the developer.

II. **Review of Related Literature**

The Sims is a strategic life simulation computer game created by game designer Will Wright, published by Maxis, and distributed by Electronic Arts. [7] The game simulates the day-to-day life and activities of simulated electronic people, or sims as the they were called. It allows you to create a unique character by defining almost all the aspects of the character: from appearance, which includes name, age, skin color, face, and clothing; up to its behavior, or what would be the personality of the sim. The character created is in full 3D graphics, which means that after a character has been created, the player can see it going around the 3D environment of the game in almost all angles.

The first Sims has a plethora of expansion packs which provided for millions, or even billions, of unique character designs. Among these expansion packs are House Party, Living Large, Hot Date, Vacation, Superstar, Makin' Magic and Unleashed. [8] Today, Sims 2 had already spent a few years in the market also with several expansion packs available, while rumors about a Sims 3 under development are already making noise in the internet's gaming community. [9]

Ragnarok Online, often refered to as RO, is a massively multiplayer online role playing game, based on the comic book Ragnarok written by Lee Myung-Jin. [10] The game allows you to create a custom character whose

attrbutes will be decided upon by the player. The games allows the player to customize the character although it is limited, at first, to choosing the gender, the hair type and hair color of the character.

The game combines 3D graphics and flat sprites to provide the interface of the game. The game environment is in 3D, while the characters in the game simulates 3D by having many flat images shift as the camera revolve around the character. [11] The clothing of the character also changes when the character acquires or changes its job, or when the character wears certain items such as head gears or masks. The weapon that the character is holding also updates visually, as the character equipped to the character is changed or removed.

Garfield's Comic Creator is a macromedia flash-based application that allows you to create a comic strip with characters from Garfield. The user is limited between a single panel and a three panel comic strip, where the user may add characters such as Garfield, Odie, Nermal, Arlene, and Jon. For each character, there are a few choices for poses and facial expressions, and although unique, the poses remain static and will be displayed as is in the panel. The user can also add items for props and change the background. Inside the panel, the user may drag, flip, delete, or scale the objects you have already added. [12]

The Strip Generator is an online comic strip creator that requires Macromedia Flash 7 or higher. It has several pre-made characters and objects which can be dragged and positioned inside the panels. The objects are limited to what the pre-made objects are. All illustrations generated by the Strip Generator is black and white. The user may also add speech balloons and edit its contents.. It is available online, and anyone can create their own comic strip and save it there. Users may also view the gallery which contains other people's saved work. [13]

Another similar system, the Strip Creator is also a website that allows users to create and save their comic strips. The system is entirely based on javascript, and employs the GIF format for its objects. The minimum number of panels in the comic strip is one and the maximum is three. For each panel, the user is allowed to put two characters, which the user can also choose from a drop down box. The location of the character is fixed, and so is the speech balloon for them, which the user can choose to have. In addition to this, the user may also change the background and add a narration that will be placed on top of the panel. The assortment of illustrations is limited to those provided in the drop down box, although the developer accepts emails from users with attached drawings in GIF format that the developer would add manually to the drop down box should he choose to. The developer also accepts donations, and gives additional functionalities for people who donate

money. These functionalities includes the ability to edit your saved comic strips, and cast a vote on a comic strip if it is bad or good. [14]

Mojo is a 2D vector-based cartoon animation application developed by Lost Marble. It provides the users with a complete set of tools to create an animated cartoon: there are drawing and painting tools, facilities for keyframe animation, and multi-layer compositing. It also supports final output of QuickTime and AVI movie files, or even streaming flash movies. Being vector-based, it is also suitable for illustrations and behaves similar to drawing facilities in Flash. [15]

G-BEE is a stand-alone shareware program from KIQ Software. It allows you to create comic strips using pre-made characters named Bruna, Bob Eduard and Caroline. It has over three hundred different drawings of sceneries, balloons, and other objects. [16]

Inskcape is an open source vector graphics editor which supports the Standard Vector Graphics format as recommended by the W3C. The main goal of Inscape is to provide a drawing tool fully compliant with XML, SVG and CSS standards, which it is able to fulfill right now. It supports basic vector editing functionalities similar to those in proprietary software such as Illustrator, Freehand or CorelDraw. Supported SVG features are the following: shapes, paths, text, markers, clones, alpha blending, transforms, gradients,

patterns, and grouping. Apart from these, it also supports Creative Commons meta-data, node editing, layers, complex path operations, bitmap tracing, text-on-path, flowed text, direct XML editing, and more. It imports formats such as JPEG, PNG, TIFF, and others and exports PNG as well as multiple vector-based formats. [17]

Another online application that lets the user create comic strips is the flash-based Comic Creator. The user can choose from a few panel layouts, and for each panel the user can add characters, speech balloons and props. Characters are pre-made, meaning the user may not create a custom character. The application also asks the user if he or she wants to print the strip after finishing it. It doesn't let the user save the comic strip in his computer. [18]

Mostly Random Comic Generator is a web-based system that accepts 160 x 160 images files in jpg, gif and png file formats. The screening of the submitted images depends mainly on the system administrator and is independent from the system. The generated comic strips is simply a series of these pictures with captions below it, formatted in HTML. [19]

III. Theoretical Framework

A. Comic Strip

A comic strip is a drawing or series of drawings that tells a story. Today, such strips are published on a recurring basis usually daily or weekly in newspapers, or in the internet. The origin of this artform dates back to ancient egypt where tombs are often engraved or painted with symbols depicting a scene. [20]

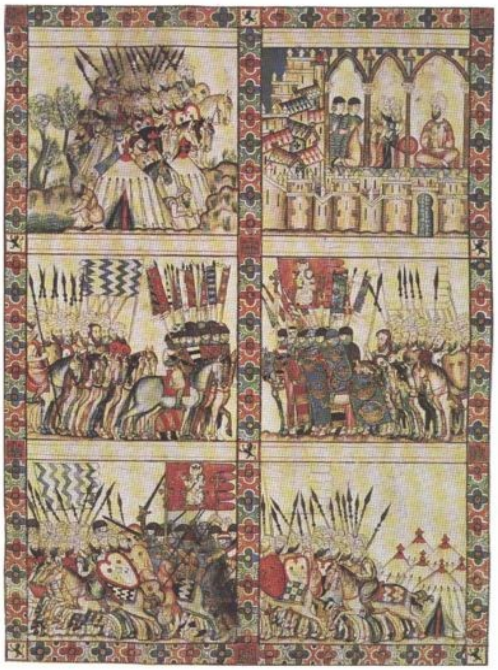


Figure 1. Proto-comic strip, taken from a Catalan manuscript

Proto-comic strips, such as the one displayed on Figure 1 dates back in the 13th century taken from a Catalan manuscript. [21] Although certain elements we are now familiar with such as speech balloons are missing, it already resembles the form of comic strips as we know it today.

The common comic strip is composed of panels, most often around one to three panels, possibly spanning multiple rows. Often the title of the comic strip can be seen atop the first panel, while the illustrator's name appears at the opposite end. Existence of words were not required, but were often added to give more information and for story-telling purposes.

B. Vector Graphics

A vector image is one of the two major graphic types, the other being bitmap. Vector graphics are made up of many individual objects, each defined by mathematical statements and has individual properties assigned to it such as color, fill, and stroke. Vector graphics are resolution independent, meaning they can be viewed with the highest quality at any scale. This is in contrast with the other graphic type, bitmap, which is raster-based and thus results in a pixelized image as the scale is increased. [22]

Software used to create vector graphics is sometimes referred to as object-based editing software. Common vector formats include AI for *Adobe Illustrator*, CDR for *CorelDRAW*, SWF for *Macromedia Flash*, DXF for *AutoCAD* and other CAD software. Rapidly gaining popularity today is the *Standard Vector Graphics* format, or simply SVG, which is an XML based format that describes a vector drawing. This format is recommended by the W3 Consortium and is now supported by major web browsers such as Mozilla Firefox and Microsoft's Internet Explorer, and can be edited in popular vector editing tools, such as *Inkscape*, an Open Source software built on Gtk+ using LibArt, or even proprietary programs such as *Adobe Illustrator* or *Macromedia Fireworks*.

Vector graphics tend to have much smaller file sizes than raster-based bitmaps since it stores only a fixed set of values independent of image

dimensions, unlike with bitmap where the file needs to store all the color information of each pixel, which means that as the image becomes bigger in dimension, the amount of information to store also increases.

For example, consider a circle of radius *r*. The main pieces of information a program needs in order to draw this circle are (1) the radius *r*, (2) the location of the center point of the circle, (3) stroke line style and color which is possibly transparent, and (4) fill style and color which is also possibly transparent.

Vector graphics has the following advantages over raster graphics:

- Minimal amount of information translates to a much smaller file size. The size of the representation doesn't depend on the dimensions of the object.
- Correspondingly, one can indefinitely zoom in on a vector image, and it remains smooth.
- A vector image may consist of many independent objects, and each object can be modified without affecting the other objects.
- The parameters of objects are stored and can be later modified. This means that moving, scaling, rotating, filling and other transformations doesn't degrade the quality of a drawing.

- It is usual to specify the dimensions in device-independent units, which results in the best possible rasterization on raster devices.

It is shown in Figure 2 that two images of different formats, vector and raster respectively, would appear differently when scaled or zoomed in. As can be observed, the magnified raster image appears pixelized, while the one in vector format remains smooth. Moreover, the zoom percentage can increase indefinitely and the vector image would still remain smooth.

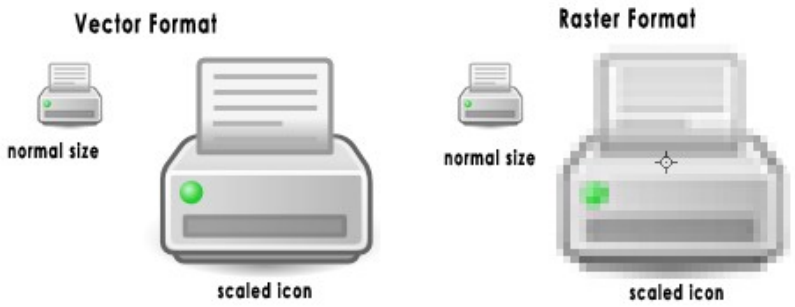


Figure 2. Scaled Vector and Raster images

C. PNG Image File Format

The Portable Network Graphics format, or PNG (pronounced “ping” []) for short, was designed to replace the older and simpler GIF format and, to some extent, the much more complex TIFF format. [20] PNG is a raster format, meaning it represents an image as a two-dimensional array of colored dots which are more commonly refered to as pixels. PNG is explicitly not a vector format, but some private extensions add vector information in

addition to the regular pixels, such as what is done by Macromedia Fireworks, but valid PNG files may never omit the pixel data. [23]

Saving, opening and re-saving a PNG image will not degrade its quality because of its lossless compression, meaning, no amount of pixel imformation is lost during the process. This is different with JPG where an amount of data is still lost even with the highest quality compression.[24]

When used in the web, PNG has the following advantages to the GIF format:

1. Alpha-channels, which means its pixels can have partial transparancy making it suitable for displaying graphics which are irregularly shaped. This feature also makes it possible to achieve effects such as drop shadows and anti-aliased edges which would not look odd against any background color. The transparency of the GIF is binary, meaning a pixel can only be completely transparent or completely opaque, which makes it a little impractical to use.
2. Gamma-correction, or cross-platform control of brightness, which results in a much better image display. The GIF format doesn’t have such feature.
3. Two-dimensional Interlacing, which is a method of progressive display, similar to progressive JPEGS, that displays “scans” of the entire image as it is being downloaded as opposed to the top to bottom downloading of web images. This allows users to have a rough preview of the image and

have an idea about what the image is even before the entire image is downloaded.

4. Better Compression, which is 5% to 25% better than that of the compression of GIF. Add the fact that its compression algorithm is not patented, unlike that of the GIF.

D. Bezier Paths

A Bézier Path is a curved line or path defined by mathematical equations, as opposed to the idea that points may only determine straight line segments. It was named after Pierre Bézier, a French mathematician and engineer who developed this method of computer drawing in the late 1960s while working for the car manufacturer Renault. [25]

The use of Bezier paths allowed car designers to experiment more and create never before perceived designs that involve curved surfaces instead of the old flat and box-like architecture. This opened new possibilities, not just for the car industry, but for architecture in general since bezier curves allowed smoothly curved lines and surfaces to be modeled in computers which weren't possible without it.

Bézier curves are widely used in computer graphics to model smooth curves. As the curve is completely contained in the convex hull of its control points, the points can be graphically displayed and used to manipulate the

curve intuitively. Affine transformations such as translation, scaling and rotation can be applied on the curve by applying the respective transform on the control points of the curve.

The most important Bézier curves are quadratic and cubic curves. Higher degree curves are more expensive to evaluate, making them impractical and inefficient for real time graphics manipulation. Instead, low order Bézier curves are patched together (obeying certain smoothness conditions) when more complex shapes are needed. [26]

A quadratic bezier curve involves three points: a start point, one control point, and an end point. The curve doesn't really pass through the control point; instead, it merely defines how far the line would curve before going to the direction of the end point. Mathematically, it is the path traced by the function $\mathbf{B}(t)$, given points $\mathbf{P}_0$ (the start point), $\mathbf{P}_1$ (the control point), and $\mathbf{P}_2$ (the end point). The parametric form of the curve, defined by the function $\mathbf{B}(t)$ is given by the following equation:

$$\mathbf{B}(t) = (1-t)^2\mathbf{P}_0 + 2t(1-t)\mathbf{P}_1 + t^2\mathbf{P}_2 \ , \ t \in [0,1].$$

Graphically, a quadratic bezier curve would appear as shown in Figure 3. As can be seen, the curve doesn't pass through the control point. In most vector editing programs that allows the user to draw paths, the control point

acts as the handle bar which can be dragged by the mouse allowing the user to adjust the curve. [27]

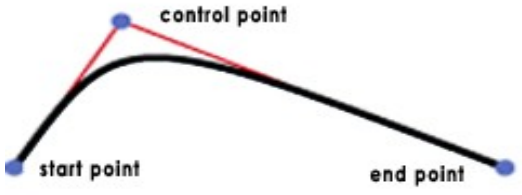


Figure 3. A Quadratic of Second Degree Bezier Curve

On the other hand, a *cubic bezier curve* is defined by four points: $\mathbf{P}_0$, $\mathbf{P}_1$, $\mathbf{P}_2$ and $\mathbf{P}_3$ in the plane or in three-dimensional space. The curve starts at P_0 going toward P_1 and arrives at P_3 coming from the direction of P_2. In general, it will not pass through P_1 or P_2; these points are only there to provide directional information. The distance between P_0 and P_1 determines "how long" the curve moves into the direction of P_2 before turning towards P_3.

The parametric form of a cubic bezier curve is given by the following equation:

$$\mathbf{B}(t) = \mathbf{P}_0(1-t)^3 + 3\mathbf{P}_1 t(1-t)^2 + 3\mathbf{P}_2 t^2(1-t) + \mathbf{P}_3 t^3 \ , \ t \in [0,1].$$

Vector software like Inkscape and Adobe Illustrator use bezier splines composed of cubic bezier curves for drawing curved shapes. Graphically, the

curve will look appear as show in Figure 4, with the two control points acting as handle bars that can be dragged with the mouse and adjust the curve. This curve can be commonly seen in the mentioned vector editing programs. [27]

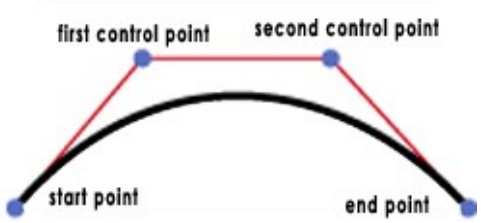


Figure 4. A Cubic of Third Degree Bezier Curve

E. Affine Transformations

An affine transformation is any transformation that preserves 1) collinearity, which means that all points lying on a line initially will still lie on a line after transformation, and 2) the ratios of distances, e.g., the midpoint of a line segment remains the midpoint after transformation. [28] In this sense, affine indicates a special class of projective transformations that do not move any objects from the affine space to the plane at infinity or conversely. An affine transformation is also called an affinity.

In general, an affine transformation is a composition of rotations, translations, dilations, and shears.[29] Geometric contraction, expansion, reflection, similarity transformations, spiral similarities are all affine transformations, as are their combinations. Figure 5 shows how these

transformations would change the appearance of an object, in this case, a square.

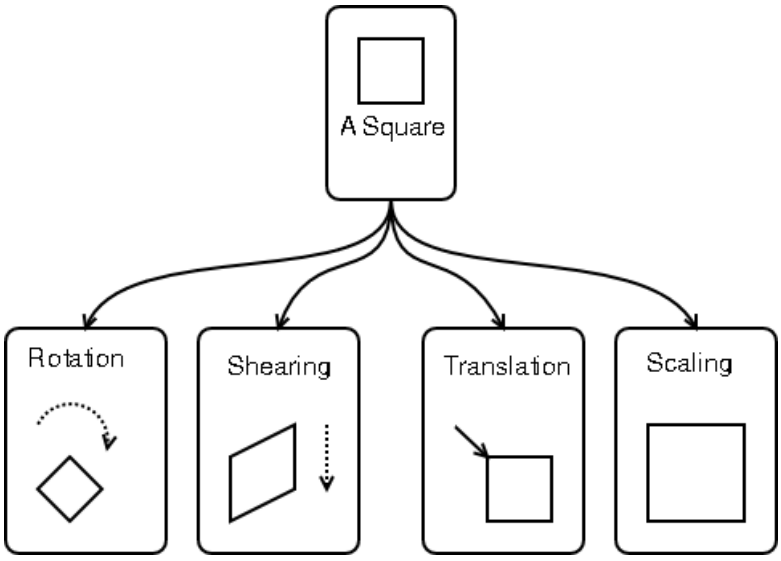


Figure 5. Affine Transformations

Affine transformations can be implemented by multiplying the set of given points with the corresponding matrices for each transformation. Figure 6 shows the matrices for the rotation, shear, scaling and translation transformations. Multiplying the point given by [x,y] with a transformation matrix with give us the resulting transformed point [x',y'] as shown in the figure. [30]

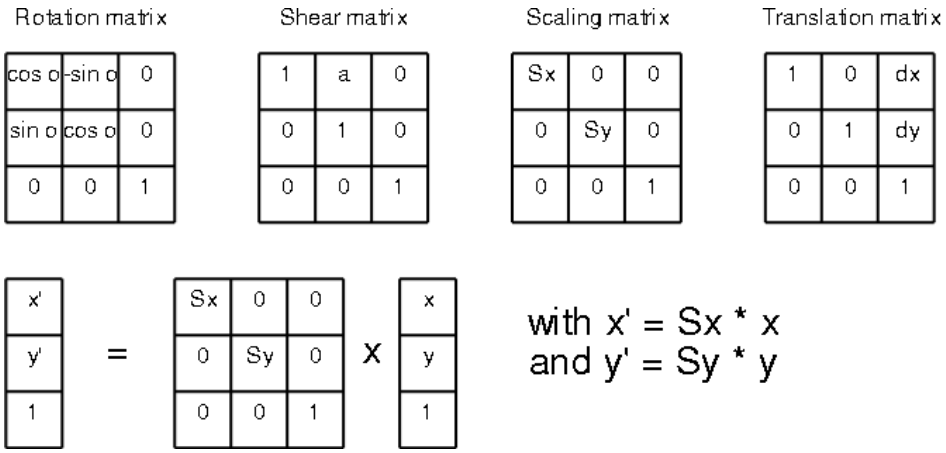


Figure 6. Affine Transformation Matrices

To achieve a combination of transformations, we can simply multiply the two matrices to achieve the required transformation matrix. A particular example combining rotation and expansion is the rotation-enlargement transformation:

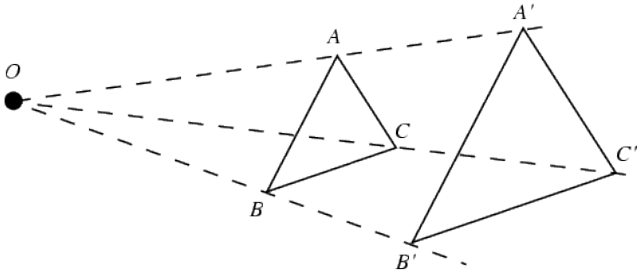
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = s \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} x - x_0 \\ y - y_0 \end{bmatrix}$$
$$= s \begin{bmatrix} \cos\alpha\,(x - x_0) + \sin\alpha\,(y - y_0) \\ -\sin\alpha\,(x - x_0) + \cos\alpha\,(y - y_0) \end{bmatrix}.$$

A *rotation* is the turning of an object or coordinate system about a fixed point by a certain angle. A rotation is an orientation-preserving orthogonal transformation. Rotations can be implemented using rotation matrices. Rotation of an object will have the effects as show in Figure 7.

Figure 7. Rotation Transformation

A *translation* is a transformation consisting of a constant offset with no rotation or distortion. In dimensional space, a translation may be specified simply as a vector giving the offset in each of the coordinates.

Figure 8. Dilation – Expansion



A *scaling*, or sometimes called dilation or expansion, is a similarity transformation which transforms each line to a parallel line whose length is a fixed multiple of the length of the original line. The simplest dilation is therefore a translation, and any dilation that is not merely a translation is called a central dilation. Two triangles related by a central dilation are said to be perspective triangles because the lines joining corresponding vertices concur. A dilation corresponds to an expansion plus a translation. Figure 8 demonstrates how scaling of a triangle is done visually.

A *shear* is a transformation in which all points along a given line L remain fixed while other points are

Figure 9. Shear

shifted parallel to L by a distance proportional to their perpendicular distance from. Shearing a plane figure does not change its area. The shear can also be generalized to three dimensions, in which planes are translated instead of lines.

IV. Design and Implementation

A. Entity Relationship Diagram

The Entity Relationship Diagram of Dibuho Comic Strip Creator is shown in Figure 10. The entities involved in the application are the: comic strip, panel, background, props, character, clothing, and facial parts. The comic strip can have one or more panels, but the minimum amount of panels is one. Each panel has a background. Each panel can contain zero or more props, and zero or more characters. Panels may also have speech balloons. A character has upper and lower garment for its clothing. A character has a set of facial parts.

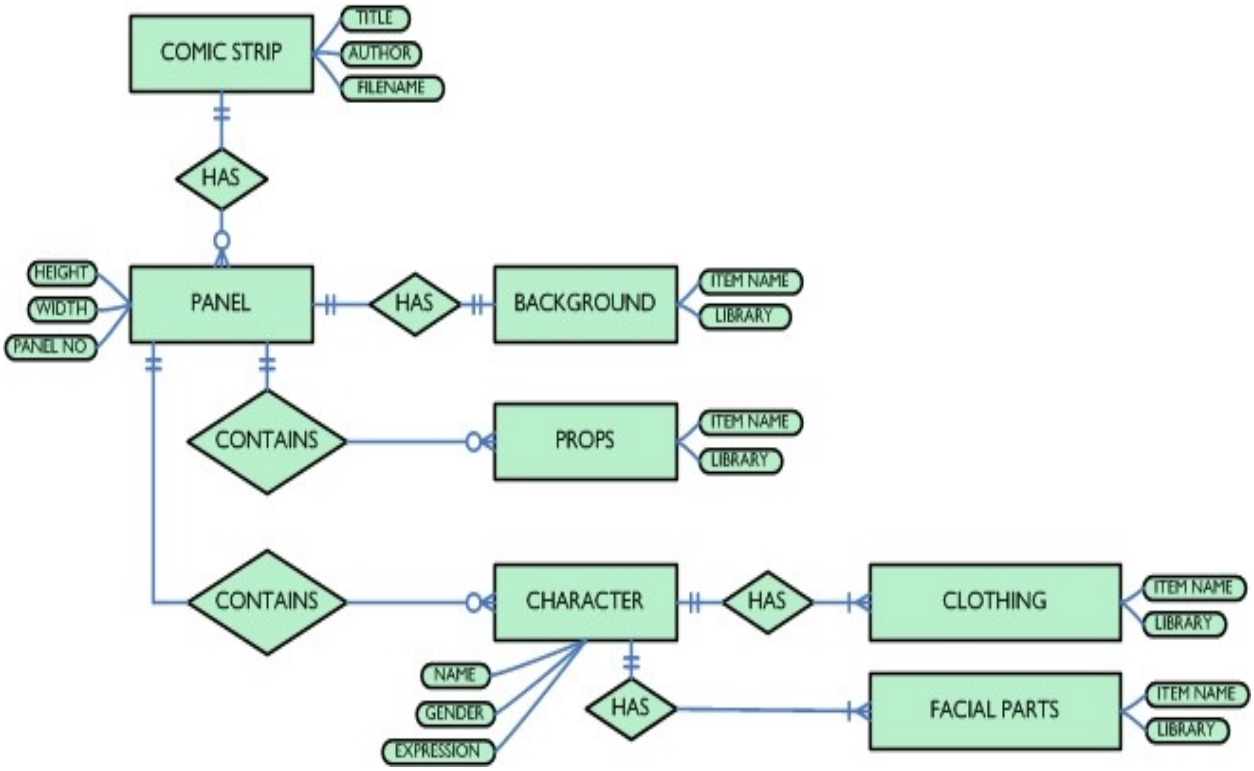


Figure 10. Entity Relationship Diagram, Dibuho Comic Strip Creator

B. Context Diagram

The Context Diagram of Dibuho Comic Strip Creator is shown in Figure 11. It shows the communication between the User and application, including participation of the Developer. Dibuho also has an Image Library which is basically a database of images. Dibuho files and exported PNG images are sent to the Hard Disk, which can later be loaded again.

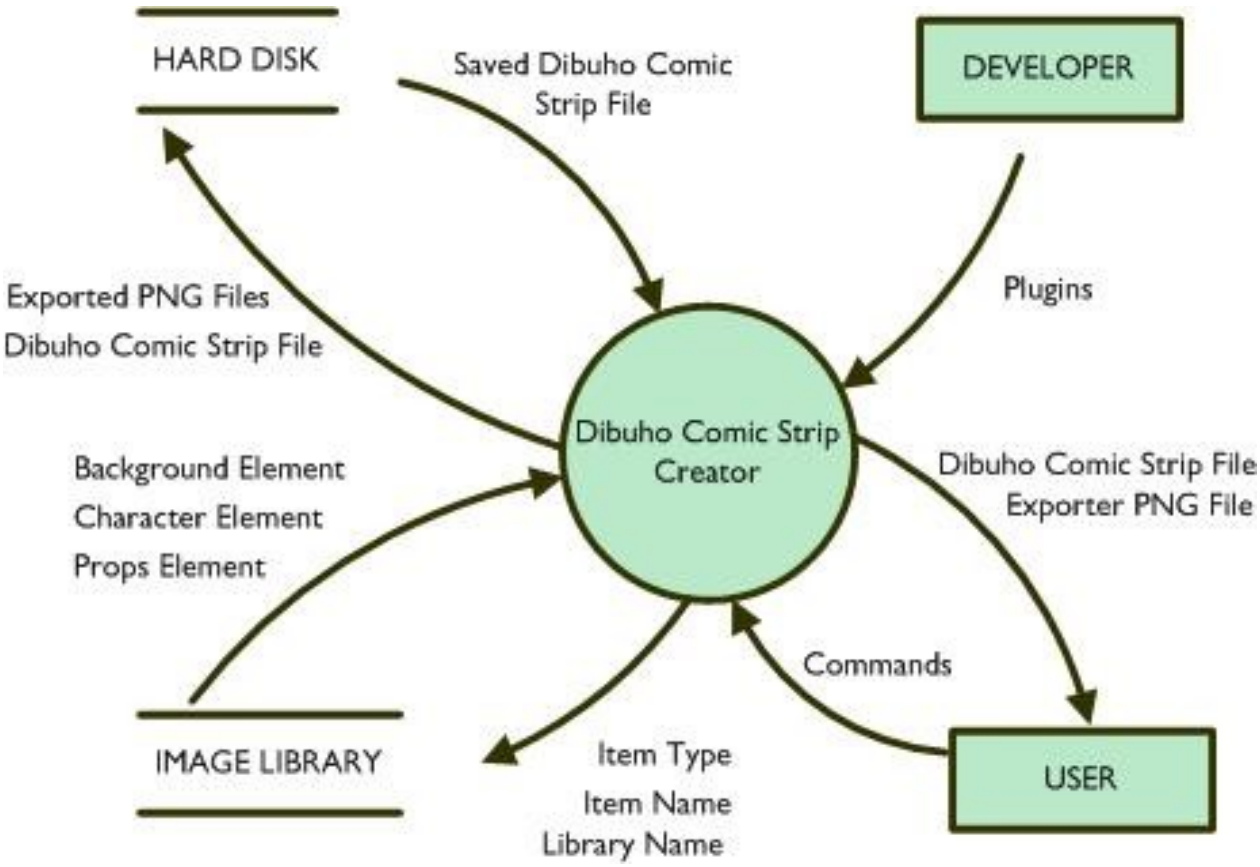


Figure 11. Context Diagram, Dibuho Comic Strip Creator

C. Data Flow Diagrams

The Top Level Data Flow Diagram of Dibuho Comic Strip Creator is shown in Figure 12. The main processes are: create character, update comic strip, load comic strip, and update image library. The user issues the commands, while the developer participates as the source of plugins. Two date stores are also involved in the processes: the Hard Disk and the Image Library.

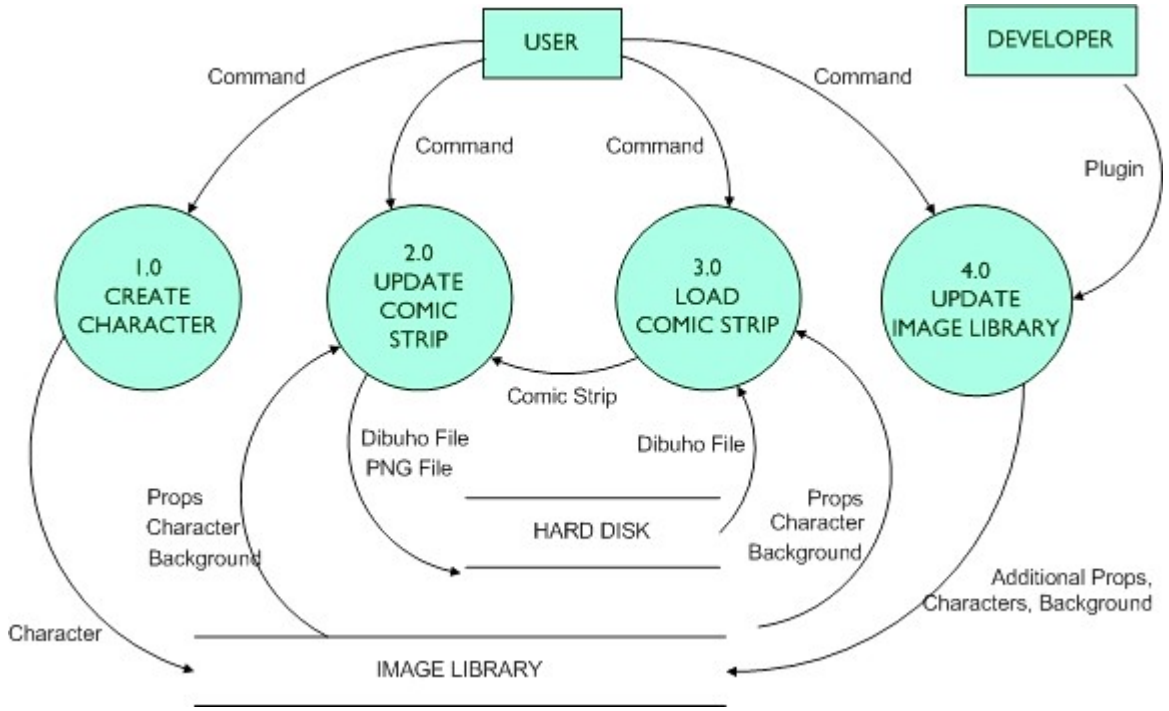


Figure 12. Top Level Data Flow Diagram, Dibuho Comic Strip Creator

Figure 13 shows the subexplosion of process 1: Create Character. The four sub-processes are: choose gender, define body attributes, select facial parts, and choose clothing. The facial parts are fetched in the image library.

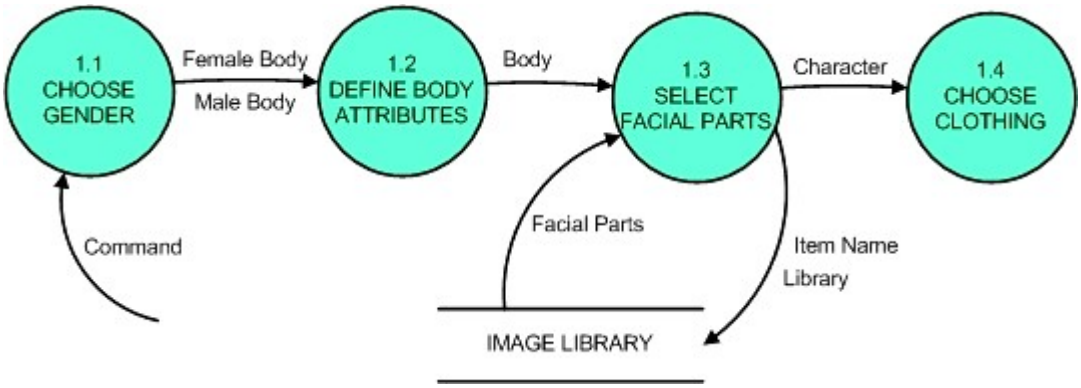


Figure 13. Subexplosion of Process 1: Create Character, Dibuho Comic Strip Creator

Figure 14 shows the subexplosion of process 2: Update Comic Strip. It has four sub-processes: create comic strip, edit comic strip, save comic strip and export comic strip. The Create Comic Strip outputs a blank comic strip which is sent to the Edit Comic Strip process. The processes loops, until the user opts to save or export the comic strip, where it enters the respective processes. The Save Comic Strip process generates a Dibuho file, while the Export Comic Strip generates a PNG file.

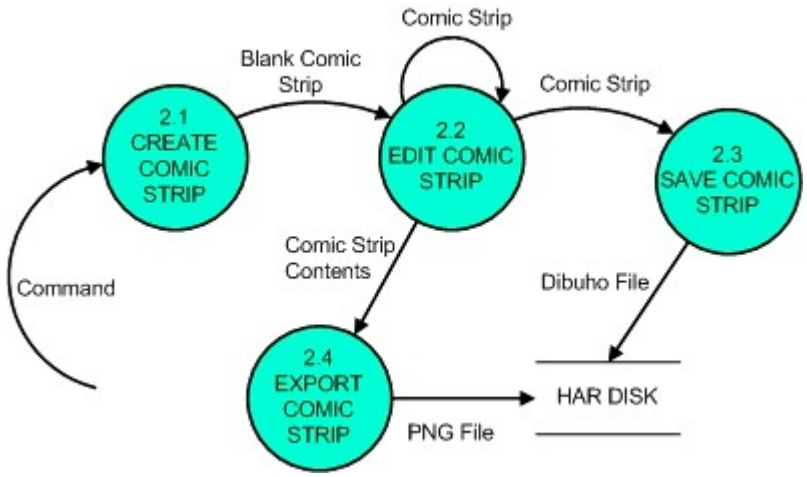


Figure 14. Subexplosion of Process 2: Update Comic Strip, Dibuho Comic Strip Creator

Figure 15 shows the subexplosion of Process 3: Load Comic Strip. The three sub-processes are : get filename, read contents of file, and generate comic strip. The file is fetched from the disk based on the filename entered by the user. Its contents are read and loaded as comic strip.

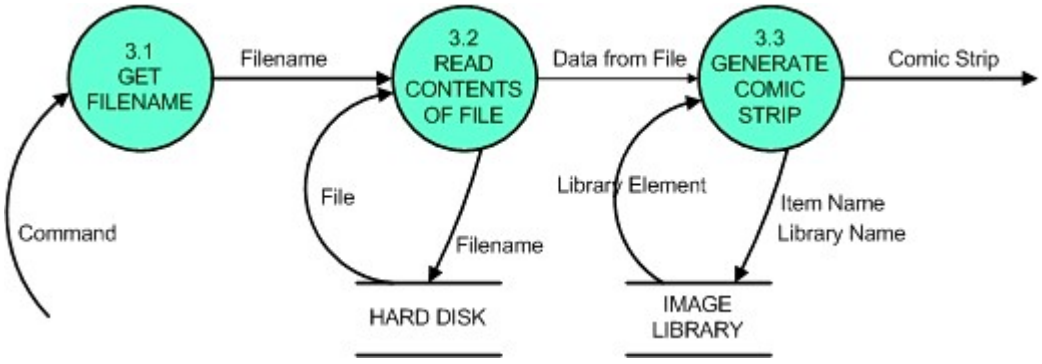


Figure 15. Subexplosion of Process 3: Load Comic Strip, Dibuho Comic Strip Creator

Figure 16 shows the subexplosion of process 4: update image library. There are two processes: restart dibuho, and load plugin. The command comes from the user to restart the application, and when he does, the plugins are loaded from the hard disk and its contents are sent to the image library.

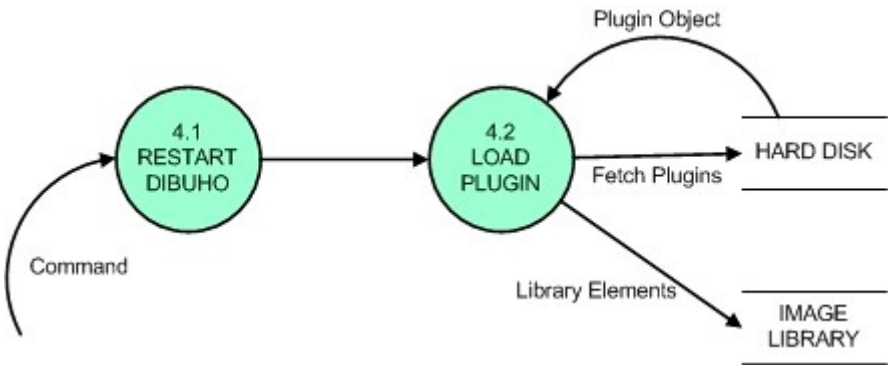


Figure 16. Subexplosion of Process 4: Update Image Library, Dibuho Comic Strip Creator

Figure 17 shows the subexplosion of process 1.2: Define Body Attributes. There are two subprocesses, Generate Body and Set Skin Color. The Generate Body process accepts gender as an input and outputs a body based on the gender. The body enters the Set Skin Color process where the body is filled with the desired color. The entire process generates a body.

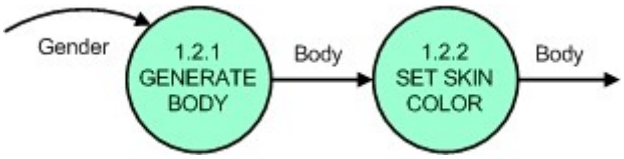


Figure 17. Subexplosion of Process 1.2: Define Body Attributes, Dibuho Comic Strip Creator

Figure 18 shows the subexplosion for process 1.3: Select Facial Parts. The input is the body generated by the previos process, and it enters the sub-processes of Select Facial Parts, namely: set eyes, set ears, set hair, set

lips, and set nose. These processes requests the required library element from the library.

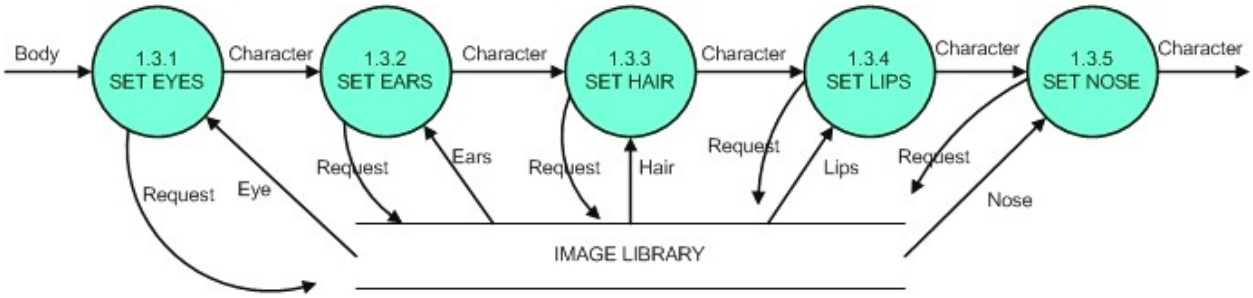


Figure 18. Subexplosion of Process 1.3: Select Facial Parts, Dibuho Comic Strip Creator

Figure 19 shows the subexplosion of Process 2.1: Create Comic Strip. The subprocesses are: input title, input author, define panels and define panel spacing. The output of these consecutive processes is a blank comic strip.

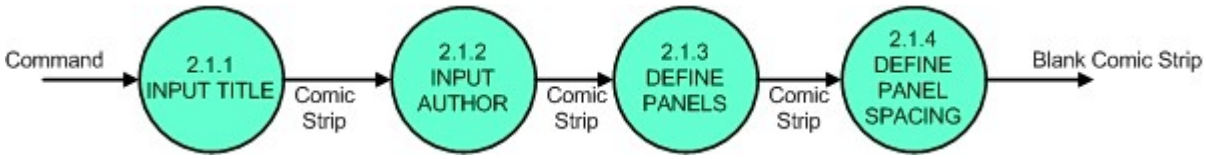


Figure 19. Subexplosion of Process 2.1: Create Comic Strip, Dibuho Comic Strip Creator

Figure 20 shows the subexplosion of Process 2.2: Edit Comic Strip. The subprocesses are the following: select panel, modify background, insert

props, insert character, and insert speech balloon. The select panel process chooses what panel to be edited, and sends that panel to the next process.

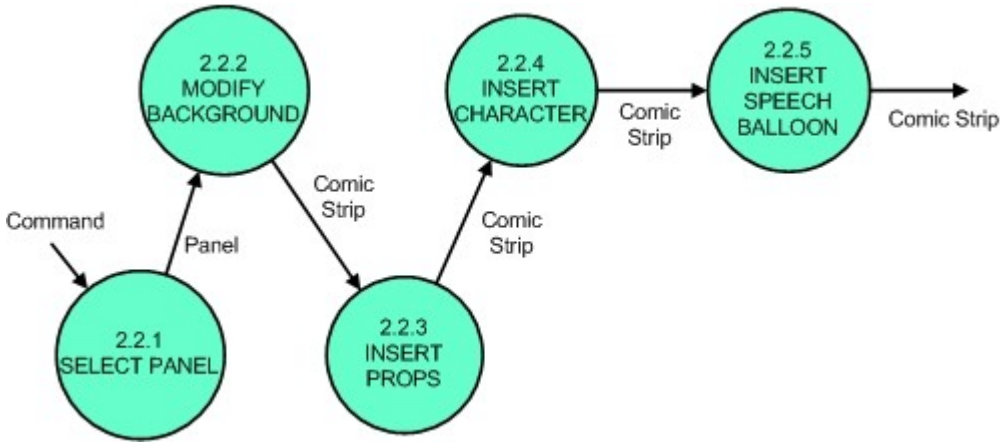


Figure 20. Subexplosion of Process 2.2: Edit Comic Strip, Dibuho Comic Strip Creator

The subexplosion of Process 2.3: Save Comic Strip is shown in Figure 21. The subprocesses are get contents, encode plaintext, and generate dibuho file. The Get Contents process outputs the contents of the comic strip and sends it in the Encode Plaintext process which generates text containing the necessary information regarding the comic strip. Finally, the Generate Dibuho File process generates the Dibuho file with the encoded text.

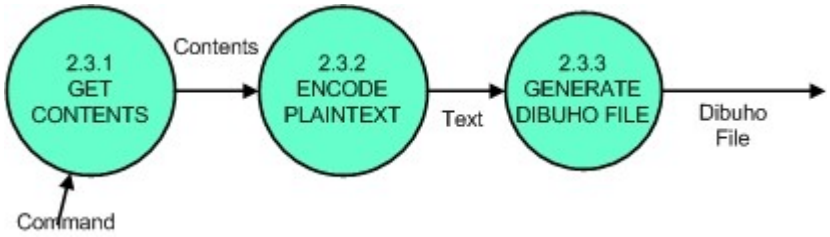


Figure 21. Subexplosion of Process 2.3: Save Comic Strip, Dibuho Comic Strip Creator

Figure 22 shows the subexplosion of process 2.4: Export Comic Strip. There are three processes involved in this explosion, namely: get contents, encode in pixels, and generate PNG File. The get contents process fetches the contents of the currently opened comic strip and sends it in the next process, Encode in Pixels, which generates data in pixels representing the comic strip. The pixel data was then sent to the Generate PNG File process where it generates the PNG file formar based on the pixel information.

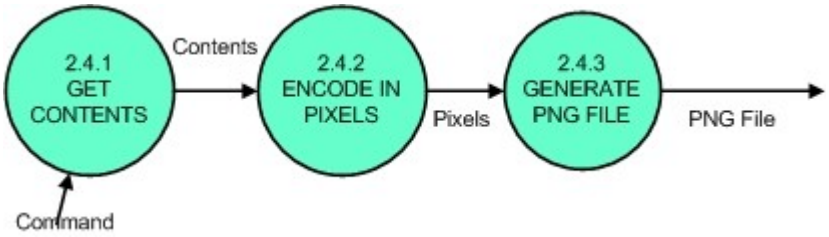


Figure 22. Subexplosion of Process 2.4: Export Comic Strip, Dibuho Comic Strip Creator

Figure 23 shows the explosion of Process 3.3: Generate Comic Strip. The explosion involves four subprocesses, namely: readline from data, analyze line, fetch item from library, and insert in panel. The Read Line from Data process reads the data from file and sends each line to Analyze Line, where the application checks on what to do. The process goes on a loop until all the necessary contents in the file are fetched.

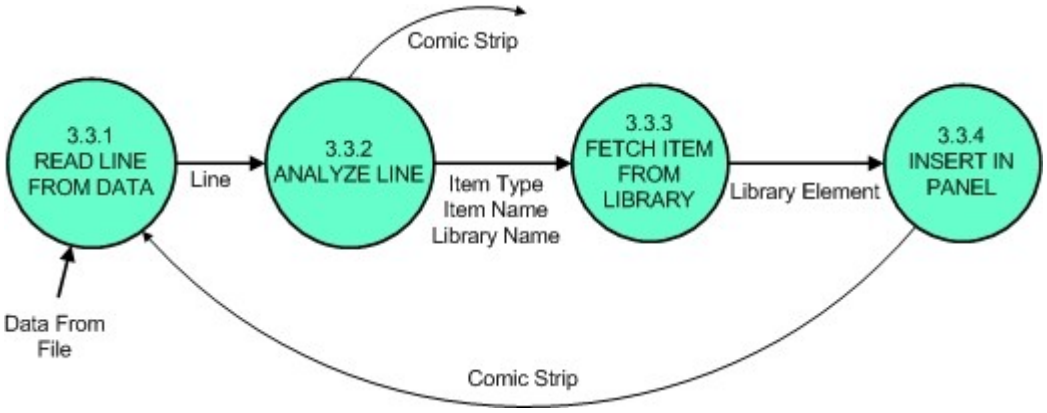


Figure 23. Subexplosion of Process 3.3: Generate Comic Strip, Dibuho Comic Strip Creator

D. Technical Architecture

Dibuho Comic Strip Creator is a stand-alone application built on the Qt GUI library. Compilation of the source codes requires Qt OpenSource 4.2.2 or higher. The Windows build of Qt OpenSource 4.2.2 requires MinGW 3.4.2 or later versions to be installed. Installer packages for specific OS are available in their website: http://www.trolltech.com/.

Operating Systems supported by the Qt includes the following:

- Unix/X11
- Windows 98 or higher
- Mac

The Qt library allows for code portability, meaning the same source code can be compiled for all the platforms. Once compiled, Dibuho will run

on the machines where the source code had been compiled. Dibuho needs these minimum requirements for it to run:

- Pentium III 800 mhz or its equivalent
- 64 MB of RAM
- 3 MB of disk space
- Any of the operating Systems mentioned above

The minimum requirements for disk space assumes that the image library contains only the BasicLib, which is the default library. Additional plugins that adds images to the library will require additional disk space.

E. Definition of Terms

1. Dibuho File Format - This pertains to the file generated by Dibuho when saving a comic strip. This contains all the necessary information about the elements contained in the comic strip including their positions and the matrix relating to the transformations for the element.
2. Panel - A Panel pertains to a single frame in a comic strip which contains the drawing, including speech balloons and narrations.
3. Elements - Elements in Dibuho pertain to the contents of the library that can either be Props, Backgrounds, Clothing, Body Parts, or Characters. These items can be placed inside the panel. The variety of

items can be expanded by getting additional plugins containing the new elements which will be loaded when Dibuho starts.

4. Props Element - These are static elements that can be placed in the panel. Although the user may still apply affine transformations such as scaling, rotation and translation, prop elements cannot be posed or articulated, and will be displayed in the panel as it is stored in the library. Examples are objects like tables and chairs.
5. Background Element - In Dibuho, this pertains to elements from the library that can be set as the background of a panel. These elements are generally rectangular and will automatically fit the dimensions of the panel when set.
6. Character Element - A character is a special type of element representing a human shape. It is fully customizable, allowing the user to create a unique character. Characters can be articulated to achieve the desired pose. Other modifications include changing of skin color and facial expression. The user may also change the clothing of characters.
7. Speech Balloon - Speech balloons are items in panels that contain either narration or character speech. Usual speech balloon types include thought bubbles, narration, normal speech and exclamation. Speech balloons may have a tail that points to the character speaking its content.

V. Results

The splash screen of the application is displayed in Figure 24. It is basically a title screen that displays the name of the application and its author while the program loads in the computer's memory. Although generally for aesthetic and presentation purposes only, it is a common practice for applications to have such.

Figure 24. Splash Screen, Dibuho Comic Strip Creator

The main interface of the application is shown in Figure 25. The main window of the application has title bar displaying the name of the application, the title and author of the currently opened comic strip, and the filename. The title bar updates as the user edits the mentioned information. The main window also has the following File, Panel, Character, Bubble, View and Help menu located at the top part of the screen.

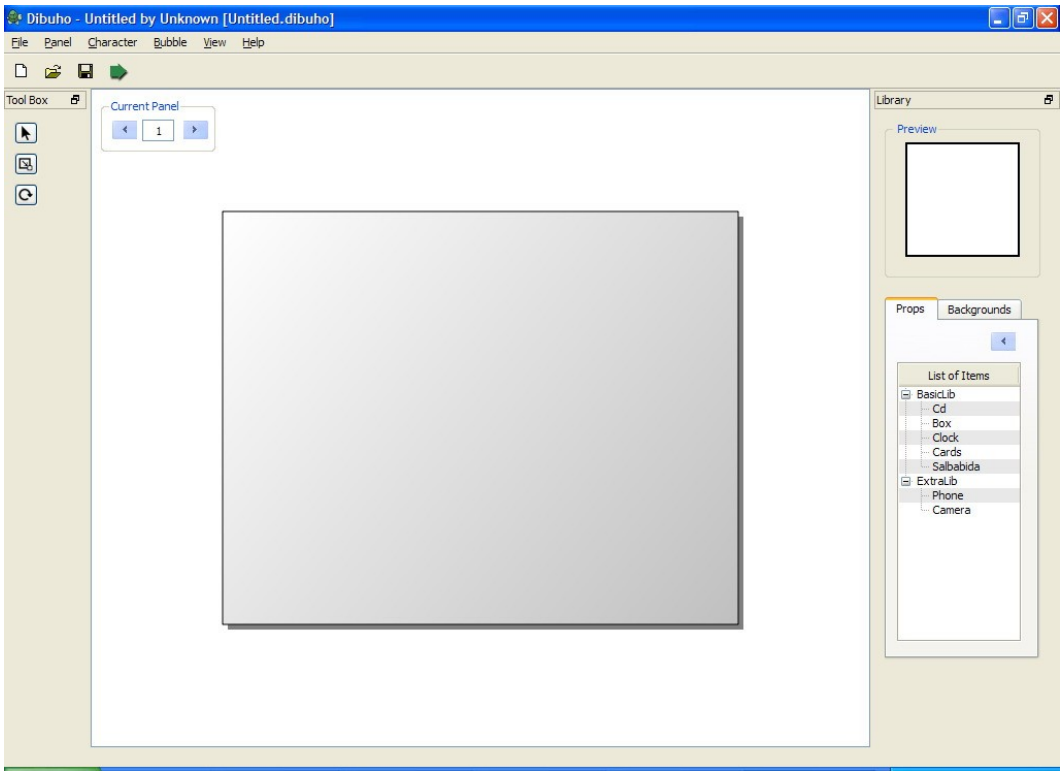


Figure 25. Main Window, Dibuho Comic Strip Creator

The main window also has a toolbar containing shortcuts to common menu commands. At the center of the main window occupying most of the screen area is the stage. The stage displays the current panel. The user may change the current panel using the navigation buttons displayed at the top left corner of the screen, which also displays the current panel. The current panel is the panel available for editing.

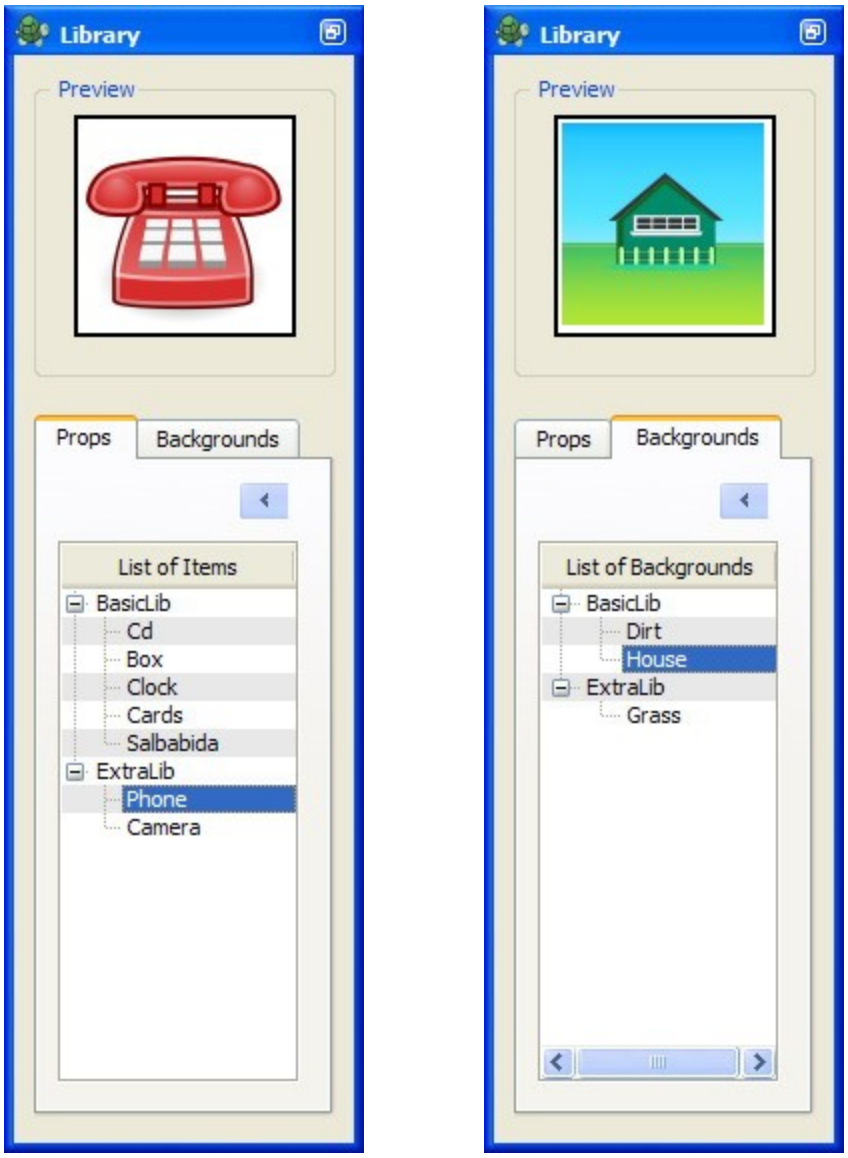


Figure 26. Props and Backgrounds Library, Dibuho Comic Strip Creator

The library window is shown in Figure 26. It has a preview window, an insert button, and two tabs: one for Props and one for Backgrounds. Each tab displays an expandable list of library elements currently available. Clicking the items will display it in the preview screen, while pressing the insert button sends the currently selected item to the stage.

The tool box window shown in Figure 27 displays three different tools that the user can use to modify the elements in the comic strip. The three tools are the following: translate, scale, and rotate, which corresponds to the affine transformations that can be applied to elements in the stage.

Figure 27. Tool Box, Dibuho Comic Strip Creator

When the user selects the New Comic Strip command under the File menu, the application prompts a user with a dialog asking for the following information: comic strip title, author, and number of panels. When the Ok button is clicked, the stage is set with the information entered. The New Comic Strip dialog is shown in Figure 28.

Figure 28. Create New Comic Strip Dialog, Dibuho Comic Strip Creator

When there is a currently opened comic strip file, the user is prompted if unsaved changes should be saved before continuing with the new comic strip command. The message box is displayed in Figure 29. The same is asked when the user closes the window, or when the user loads a previously saved comic strip file.

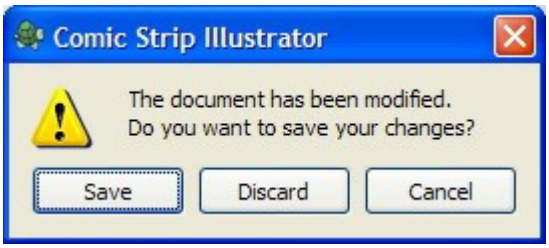


Figure 29. Save Prompt, Dibuho Comic Strip Creator

The Open command under the File menu opens a file dialog as shown in Figure 30 and lets the user select a previously saved Dibuho to be opened for editing. If the file is valid, it is loaded to the stage.

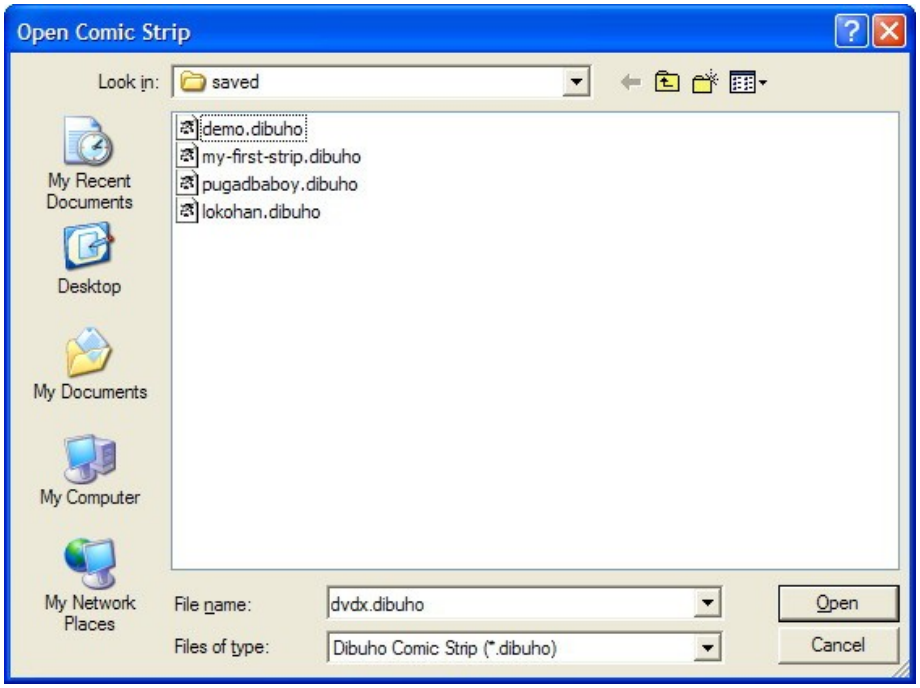


Figure 30. Open Comic Strip Dialog, Dibuho Comic Strip Creator

The Save command under the File menu also opens a file dialog as shown in Figure 31. The dialog asks for a filename, and when a valid filename is entered, the content of the Stage is then encoded in a Dibuho File for future editing. The difference between the Save and Save As command is that Save simply outputs the Stage to the a file when it is already saved initially, but calls the Save As command otherwise to ask for a filename.

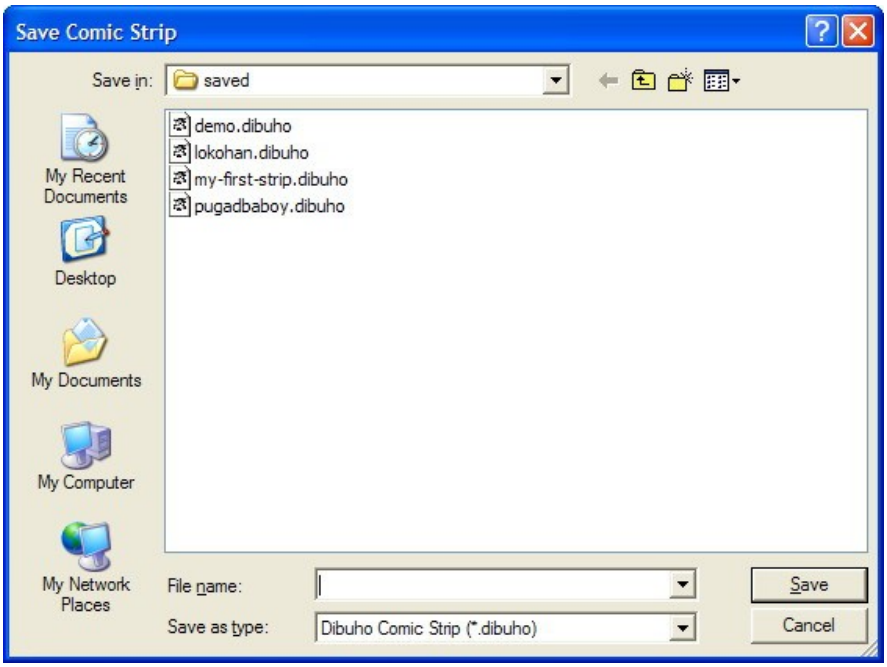


Figure 31. Save Comic Strip Dialog, Dibuho Comic Strip Creator

The Export As PNG command, which opens the dialog as shown in Figure 32, acts similar to the Save As command since it also asks for a filename but instead of encoding the Stage into the Dibuho format, it is exported as a stream of bytes and saved as a flat PNG image.

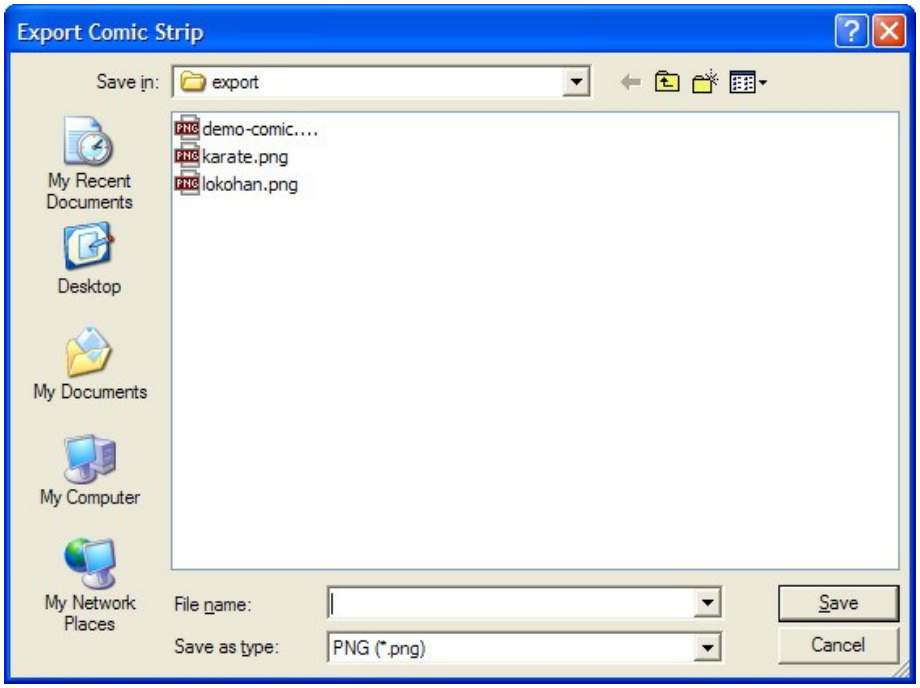


Figure 32. Export As PNG, Dibuho Comic Strip Creator

Under the Panel menu, the Insert New Panel command appends a blank panel to the currently opened comic strip. The user is allowed to add an indefinite number of panels. The Delete Panel command, on the other hand, removes the current panel from the stage along with all its contents. An error message is displayed when there is only one remaining panel, the minimum number of panels a comic strip can have.

The Clear Background command removes the Background Element in the current panel. If the Background is already empty, this command does nothing. The Preferences command displays the preferences dialog as shown in Figure 33. This allows the user to edit basic comic strip information such as author, title, and panel spacing. The Options command displays the

Options dialog shown in Figure 34, which allows the user to edit the current panel's width and the border thickness of the panel when it is exported.

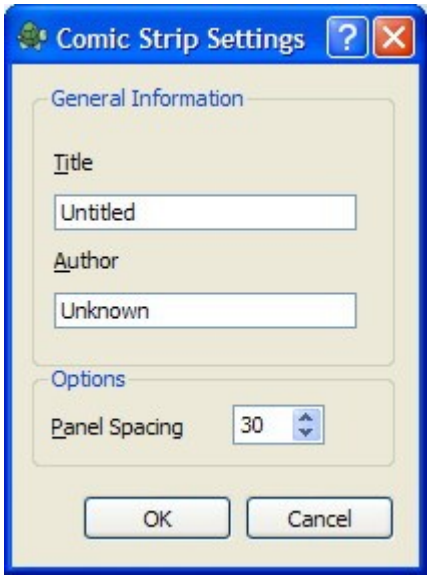


Figure 33. Preferences, Dibuho Comic Strip Creator

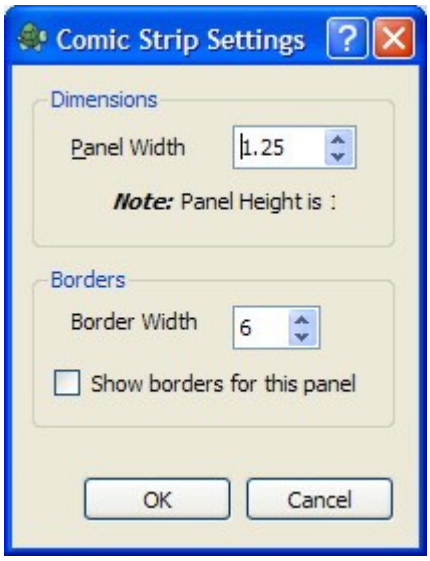


Figure 34. Panel Options, Dibuho Comic Strip Creator

Under the Character menu, the Create New Character command loads the Character dialog as shown in Figure 35. The dialog allows the user to edit character element. The user may edit its name and gender, and select the facial elements in it such as the eyes, nose, lips, ears and hair. It also allows the user to change the facial expression of the Character.

When already created, the user may choose to insert the Character to the stage by clicking the insert button. Once in the stage, the user may pose the Character and apply the desired affine transformations.

Figure 35. Character Dialog, Dibuho Comic Strip Creator

Under the Bubble menu, the Insert Bubble command displays the Bubble dialog which is shown in Figure 36. The user may input the speech bubble content in the box provided, select the bubble type, and enter a value for the speech bubble width.

Figure 36. Bubble Dialog, Dibuho Comic Strip Creator

VI. Discussion

The Dibuho Comic Strip Creator is an extensible application that allows its users to create comic strips without having to learn how to draw. As long as the user has the ideas, the comic strip can be generated through this application.

The user can utilize the available library of elements, which can be in the form of props, characters, or backgrounds. A default library is available and although this library may be limited and would appear to restrict the user, it could be extended by securing plugins from the developer. These plugins contains additional library elements that would increase the choices of the user. Plugins may contain elements with varying drawing style that would adapt to the users' needs.

Since library elements are vector graphics, it allows the users to modify them without degrading its quality and the resulting comic strip. Basic transformations can be applied to the elements. These transformations include rotations, scaling and translations allowing for flexible use of the elements in the comic strip panel.

Available digital drawing tools in the market today both free and proprietary such as GIMP, Photoshop, or Inkscape may me more flexible than Dibuho but these applications have more have complex functions that first-

time users may have trouble understanding. Learning them would require a separate tutorial, even if all the user needs is to produce a comic strip.

Since Dibuho is especially intended for the creation of comic strip, it has a simpler interface that the users would easily get used to. The character creation facility of Dibuho allows the user to design characters to suit them to their ideas. This saves them the hassle of drawing each character in every panel, since the user can just insert the created character and pose them in another position, or perhaps change the facial expression, increase the size, rotate it, or whatever the comic strip requires. This can be done even if the user has no background with drawing.

Other software similar to Dibuho that are specifically intended for the creation, such as the web-based Comic Creator doesn’t provide users with the same flexibility that Dibuho offers. [] One major difference is that it doesn’t have support for customized characters. The user may only choose from the pre-made characters, who will be placed in default spots in the fixed three-panel comic strips. The user may not pose them, change the facial expressions, or customize the clothes and their color. It also goes the same with the speech balloons and miscellaneous items. They cannot be moved, scaled, or translated inside the panel.

VII. Conclusion

The Dibuho ComicCreator is able to satisfy the objectives stated. Its user can create a character based on the available library elements. The user specifies the character's names, gender, and body attributes. This character can be inserted

The user can also create a comic strip. The user can enter the author, title, and number of panels in the comic strip, and later edit it. Elements comprised of props, backgrounds, and characters which are available in the library can be used by the user. Props and characters can be placed in the library, and later be transformed by rotating, translating, and scaling. In addition, the user can also pose the character, change the facial expressions, and modify body attributes by selecting facial parts, skin color, and its upper and lower garment. The user may also insert a speech balloon in the comic strip with the specified content and balloon type.

The user can also also save the comic strip in Dibuho format and later open it again for editing. In addition to this, the user can also export the comic strip in PNG format.

The user can also extend the library of elements by securing a plugin from the developer.

VIII. Recommendation

Although the application supports affine transformations for the elements, it is only executed through key press events from the keyboard, i.e. pressing the plus or minus keys to decrease or increase the scale, and rotate clockwise or counter clockwise. It is recommended that these functionalities be implemented with mouse events such as clicks and drags so as to make the application more user-friendly and easy to use.

It is also recommended that basic editing functions such as the copy, cut, and paste commands be applicable to the elements in the comic strip panel. This will make it more convenient for users to use the program.

The export size is of fixed size, format, and resolution. It is recommended that the resolution and size of the generated PNG image can be modified. This will make the export more flexible and will also take advantage of vector graphics on which this application is based. It is also recommended that, if possible, export formats other than PNG be made available.

IX. Bibliography

[1] Wikipedia, Comic Strip. http://en.wikipedia.org/wiki/Comic_strip

[2] History of Sequential Art. http://www.comic-art.com/history/history0.htm

[3] History of Comic Books, http://www.comic-art.com/history/history1.htm

[4] List of Comic Strips, http://en.wikipedia.org/wiki/List_of_comic_strips

[5] Comics: A Multi-dimensional Teaching Aid in Integrated-skills Classes. http://www.readwritethink.org/student_mat/student_material.asp?id=21

[6] Drawing Resources and Tutorials. http://www.artshow.com/resources/drawing.html

[7] The Sims. http://en.wikipedia.org/wiki/The_Sims

[8] The Sims Expansion Packs. http://compsimgames.about.com/od/thesimsexpansionpacks/

[9] IGN: The Sims 3 Spotted. http://pc.ign.com/articles/682/682828p1.html

[10] Ragnarok Online - Wikipedia. http://en.wikipedia.org/wiki/Ragnarok_Online#Character_customization

[11] Ragnarok Online. http://iro.ragnarokonline.com/game/

[12] Student Materials: Comic Creator. http://www.readwritethink.org/student_mat/student_material.asp?id=21

[13] StripGenerator. http://www.thirdframestudios.com/adgame/stripgen/

[14] Strip Creator. http://www.stripcreator.com/

[15] Lost Marble, Moho. http://www.lostmarble.com/moho.html

[16] G-BEE Download. http://www.freedownloadscenter.com/Games/Educational_Games/G_Bee.html

[17] Inkscape. Draw Freely. http://www.inkscape.org/

[18] Comic Creator. http://www.readwritethink.org/materials/comic/index.html

[19] Mostly Random Comic Generator. http://www.pitchingwoo.net/mostlyrandom/cg

[20] Comic Strip: History. http://www.factmonster.com/ce6/ent/A0857471.html

[21] Proto-comic strip. http://en.wikipedia.org/wiki/Image:Protocomicstrip.jpg

[22] Raster and Vector Graphics. http://www.sketchpad.net/basics1.htm

[23] W3C: PNG. http://www.w3.org/Graphics/PNG/

[24] PNG (Portable Network Graphics) Home Site. http://www.libpng.org/pub/png/

[25] Intro to PNG features. http://www.libpng.org/pub/png/pngintro.html

[26] Bezier Curves. http://www.moshplant.com/direct-or/bezier/

[27] Animated Mathematics - A curve as a Bezier path. http://www.svgopen.org/2003/papers/AnimatedMathematics/bezier.html

[28] Affine Transformation. http://en.wikipedia.org/wiki/Affine_transformation

[29] Affine Transformations. http://www.quantdec.com/GIS/affine.htm

[30] Wolfram Mathworld: Affine Transformation.

http://mathworld.wolfram.com/AffineTransformation.html

X. Appendix

Path Element Specification SVG 1.1

Command	Name	Parameters	Description
M (absolute) m (relative)	moveto	moveto (x y)+	Start a new sub-path at the given (x,y) coordinate. M (uppercase) indicates that absolute coordinates will follow; m (lowercase) indicates that relative coordinates will follow. If a relative moveto (m) appears as the first element of the path, then it is treated as a pair of absolute coordinates. If a moveto is followed by multiple pairs of coordinates, the subsequent pairs are treated as implicit lineto commands.
Z or z	closepath	None	Close the current subpath by drawing a straight line from the current point to current subpath's initial point.
L (absolute) l (relative)	lineto	(x y)+	Draw a line from the current point to the given (x,y) coordinate which becomes the new current point. L (uppercase) indicates that absolute coordinates will follow; l (lowercase) indicates that relative coordinates will follow. A number of coordinates pairs may be specified to draw a polyline. At the end of the command, the new current point is set to the final set of coordinates provided.
H (absolute) h (relative)	horizontal lineto	x+	Draws a horizontal line from the current point (cpx, cpy) to (x, cpy). H (uppercase) indicates that absolute coordinates will follow; h (lowercase) indicates that relative coordinates will follow. Multiple x values can be provided (although usually this doesn't make sense). At the end of the command, the new current point becomes (x, cpy) for the final value of x.
V (absolute) v (relative)	vertical lineto	y+	Draws a vertical line from the current point (cpx, cpy) to (cpx, y). V (uppercase) indicates that absolute coordinates will follow; v (lowercase) indicates that relative coordinates will follow. Multiple y values can be provided (although usually this

			doesn't make sense). At the end of the command, the new current point becomes (cpx, y) for the final value of y.
C (absolute) c (relative)	curveto	(x1 y1 x2 y2 x y)+	Draws a cubic Bézier curve from the current point to (x,y) using (x1,y1) as the control point at the beginning of the curve and (x2,y2) as the control point at the end of the curve. C (uppercase) indicates that absolute coordinates will follow; c (lowercase) indicates that relative coordinates will follow. Multiple sets of coordinates may be specified to draw a polybézier. At the end of the command, the new current point becomes the final (x,y) coordinate pair used in the polybézier.
S (absolute) s (relative)	shorthand/smooth curveto	(x2 y2 x y)+	Draws a cubic Bézier curve from the current point to (x,y). The first control point is assumed to be the reflection of the second control point on the previous command relative to the current point.

Source Codes
Class Declarations

bubble.h

```
#ifndef BUBBLE_H
#define BUBBLE_H

#include <QGraphicsItem>
#include <QFont>
#include <QString>
#include <QPainterPath>
#include <QPainter>
#include <QFontMetrics>
#include "bubbletail.h"

class Bubble : public QGraphicsItem
{

public:
   Bubble(const QString &speech, const QString &type, const
QString &dir, const int width, QGraphicsItem *parent = 0);
   QRectF boundingRect() const;
   void paint(QPainter *painter, const
QStyleOptionGraphicsItem *option, QWidget *widget = 0);

protected:
   BubbleTail *tail;
   QString content;
   QString type;
   QString direction;
   int frameWidth;
};

#endif
```

bubbletail.h

```
#ifndef BUBBLETAIL_H
#define BUBBLETAIL_H

#include <QGraphicsItem>
#include <QPainter>
#include <QPen>

class BubbleTail : public QGraphicsItem
{

public:
   BubbleTail(const QString &direction, QGraphicsItem
*parent = 0);
   QRectF boundingRect() const;
   void paint(QPainter *painter, const
QStyleOptionGraphicsItem *option, QWidget *widget = 0);

protected:
   QString dir;
};

#endif
```

dialogbubble.h

```
#ifndef DIALOGBUBBLE_H
#define DIALOGBUBBLE_H

#include "ui_dialogbubble.h"
#include <QDialog>

class DialogBubble : public QDialog, public Ui::BubbleDialog
{
   Q_OBJECT

public:
   DialogBubble(QWidget *parent = 0);

};

#endif
```

dialogcharacter.h

```
#ifndef DIALOGCHARACTER_H
#define DIALOGCHARACTER_H

#include "ui_dialogcreatecharacter.h"
#include "character.h"
#include "elements.h"
#include "dialogselectfacials.h"
#include <QtGui>

class DialogCharacter : public QDialog, public
Ui::DialogCreateCharacter
{
   Q_OBJECT

public:
   DialogCharacter(QWidget *parent = 0);

public slots:
        void updatePreviewWindow();
   void getSkinColor();
   void getUgarColor();
   void getLgarColor();
   void send();

signals:
        void sendCharacter(QGraphicsItem *item);

private:
        Character *character;
        QGraphicsScene *scene;
        QColor skin;
        QColor ugar;
        QColor lgar;

};

#endif
```

dialognew.h

```
#ifndef DIALOGNEW_H
#define DIALOGNEW_H

#include "ui_dialognew.h"
#include <QDialog>

class DialogNew : public QDialog, public Ui::NewCS
{
   Q_OBJECT

public:
   DialogNew(QWidget *parent = 0);

};

#endif
```

dialogoptions.h

```
#ifndef DIALOGOPTIONS_H
#define DIALOGOPTIONS_H

#include <QDialog>
#include "ui_dialogoptions.h"

class DialogOptions : public QDialog, public Ui::Options
{
   Q_OBJECT

public:
   DialogOptions(QWidget *parent = 0);
```

```
};

#endif
```

dialogpreferences.h

```
#ifndef DIALOGPREFERENCES_H
#define DIALOGPREFERENCES_H

#include "ui_dialogpreferences.h"
#include <QDialog>

class DialogPreferences : public QDialog, public
Ui::Preferences
{
    Q_OBJECT

public:
    DialogPreferences(const QString &title, const QString
&author, int panelSpacing, QWidget *parent = 0);

};

#endif
```

dialogpreviewstrip.h

```
#ifndef DIALOGPREVIEWSTRIP_H
#define DIALOGPREVIEWSTRIP_H

#include <QtGui>

class DialogPreviewStrip : public QDialog
{
    Q_OBJECT

public:
    DialogPreviewStrip(QImage image, QWidget *parent = 0);
    QSize sizeHint();

};

#endif
```

elements.h

```
#ifndef ELEMENTS_H
#define ELEMENTS_H

class QPoint;
class QGraphicsItem;
class QGraphicsSvgItem;
class QString;
class QStringList;

class PropElement
{
public:
    virtual ~PropElement() {}

    virtual QStringList props() const = 0;
    virtual QGraphicsItem *insertElement(const QString
&name) = 0;
};

class CharacterElement
{
public:
    virtual ~CharacterElement() {}

    virtual QStringList characters() const = 0;
    virtual QGraphicsItem *insertCharacter(const QString
&name) = 0;
};

class BackgroundElement
{
public:
    virtual ~BackgroundElement() {}

    virtual QStringList backgrounds() const = 0;
    virtual QGraphicsItem *queryBackground(const QString
&name) = 0;
};

class BodyElement
{
public:
    virtual ~BodyElement() {}

    virtual QStringList eyes() const = 0;
    virtual QStringList nose() const = 0;
    virtual QStringList ears() const = 0;
    virtual QStringList lips() const = 0;
    virtual QStringList hair() const = 0;
    virtual QGraphicsItem *getPart(const QString &name,
const QString &orientation) = 0;
};

class ClothingElement
{
public:
    virtual ~ClothingElement() {}

    virtual QStringList uppergarments() const = 0;
    virtual QStringList lowergarments() const = 0;
    virtual QStringList footwear() const = 0;
    virtual QStringList accesories() const = 0;
    virtual QGraphicsItem *getPart(const QString &name,
const QString &orientation) = 0;
};


Q_DECLARE_INTERFACE(PropElement,
"com.iieugenio.dibuho.PropElement/1.0")
Q_DECLARE_INTERFACE(CharacterElement,
"com.iieugenio.dibuho.CharacterElement/1.0")
Q_DECLARE_INTERFACE(BackgroundElement,
"com.iieugenio.dibuho.BackgroundElement/1.0")
Q_DECLARE_INTERFACE(BodyElement,
"com.iieugenio.dibuho.BodyElement/1.0")
Q_DECLARE_INTERFACE(ClothingElement,
"com.iieugenio.dibuho.ClothingElement/1.0")

#endif
```

library.h

```
#ifndef LIBRARY_H
#define LIBRARY_H

#include <QWidget>
#include <QStringList>
#include <QTreeWidgetItem>
#include <QGraphicsScene>

#include "ui_library.h"
#include "elements.h"

class Library : public QWidget, public Ui::libraryWidget
{
    Q_OBJECT

public:
    Library(QWidget *parent = 0);
    void populateLibrary(QObject *plugin, const QStringList
&props, const QStringList &backgrounds);

public slots:
        void setPreviewWindow(QTreeWidgetItem *item,
int column);
        void sendItem();
        void setBackground();
        void clearPreviewItem();

signals:
        void emitItem(QGraphicsItem *item);
        void emitBackground(QGraphicsItem
*background);
```

```
private:
    QList<QObject *> plugins;
    QStringList pluginNames;
    QGraphicsScene *scene;
    QGraphicsItem *itemInPreview;

};

#endif
```

panelnav.h

```
#ifndef PANELNAV_H
#define PANELNAV_H

#include "ui_panelnav.h"
#include <QWidget>

class PanelNav : public QWidget, public Ui::panelNav
{
    Q_OBJECT

public:
    PanelNav(QWidget *parent = 0);

};

#endif
```

stage.h

```
#ifndef STAGE_H
#define STAGE_H

#include <QGraphicsView>
#include <QKeyEvent>
#include <QList>
#include <QGraphicsScene>
#include <QTextStream>
#include "elements.h"
#include "dialogoptions.h"
#include "panelnav.h"

class Stage : public QGraphicsView
{
    Q_OBJECT

public:
    Stage();

    void setNewStage(const QString &author, const QString
&title, const int numOfPanels, const int panelSpacing);
    void encodeStage(QTextStream &content);
    QString decodeStage();
    QImage exportStage();
    bool isModified();
    void setModified(bool mod);

    QPen borders;
    QString comicStripTitle;
    QString comicStripAuthor;
    QString currentTool;
    int panelCount;
    int panelHeight;
    int panelSpacing;
    int topMargin;
    int comicStripWidth;
    int comicStripHeight;
    int currentSceneIndex;
    bool modified;

public slots:
    void insertPanel();
    void deletePanel();
    void optionsPanel();
    void getNextScene();
    void getPrevScene();
    void getTool(QString tool);
    void clearBackground();
    void drawBackground(QPainter *painter, const QRectF
&rect);
    void drawForeground(QPainter *painter, const QRectF
&rect);
    void keyPressEvent(QKeyEvent *event);
    void setupMatrix();
    void acceptItem(QGraphicsItem *element);
    void setBackground(QGraphicsItem *element);

signals:
    void valueChanged(QString value);
    void requestItem(QString lib, QString item, QMatrix
matrix, QPointF pos, int z);
    void requestBackground(QString lib, QString name);

private:
    void calculateComicStripSize();

    QMatrix matrix;
    PanelNav *nav;
    QGraphicsScene *currentScene;
    QList<QGraphicsScene *> panels;
    QList<qreal> panelW;
    QList<int> layerCounter;
    QList<int> libCounter;
    QStringList libs;
    qreal scaleValue;
};

#endif
```

toolbox.h

```
#ifndef TOOLBOX_H
#define TOOLBOX_H

#include <QWidget>
#include <QVBoxLayout>
#include <QButtonGroup>
#include <QPushButton>
#include "ui_toolbox.h"

class ToolBox : public QWidget, public Ui::ToolBox
{
    Q_OBJECT

public:
    ToolBox(QWidget *parent = 0);
    QString currentTool;

signals:
    void changeTool(QString tool);

public slots:
    void setRotateTool();
    void setScaleTool();
    void setSelectTool();
};

#endif
```

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QStringList>
#include <QDir>

#include "toolbox.h"
#include "propslib.h"
#include "bglib.h"
#include "library.h"
#include "stage.h"
#include "dialogcharacter.h"
#include "dialogbubble.h"
#include "dialognew.h"
#include "dialogpreferences.h"
```

```
#include "dialogpreviewstrip.h"

class QAction;
class QDockWidget;
class QToolBox;
class QMenu;
class QString;

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow();
    Stage *stage;

protected:
    void closeEvent(QCloseEvent *event);

private slots:
    void newCS();
    void openCS();
    bool saveCS();
    bool saveAsCS();
    bool exportCS();
    void previewCS();
    void preferences();
    void insertBubble();
    void insertCharacter();
    void newCharacter();
    void openLibrary();
    void about();

private:
    // Initialization functions
    void createActions();
    void createMenus();
    void createToolbars();
    void createStatusBar();
    void createTools();
    void createStage();
    void loadLibrary();

    // Settings functions
    void readSettings();
    void writeSettings();

    // Helper Functions
    bool maybeSave();
    void loadFile(const QString &fileName);
    bool saveFile(const QString &fileName);
    void setCurrentFile(const QString &fileName);
    void addToMenu(QObject *plugin, const QStringList
&texts, QMenu *menu, const char *member);
    void populateLibrary(QObject *plugin);

    // Stage matters
    QString currentFile;
    QDir pluginsDir;
    QStringList pluginFileNames;

    // Menus
    QMenu *fileMenu;
    QMenu *panelMenu;
    QMenu *viewMenu;
    QMenu *characterMenu;
    QMenu *libraryMenu;
    QMenu *helpMenu;

    // Toolbars
    QToolBar *fileToolbar;

    // Dock Widgets
    ToolBox *toolbox;
    PropsLib *propsLibrary;
    Library *library;
    BgLib *bgLibrary;
    QDockWidget *toolboxContainer;
    QDockWidget *propsLibraryContainer;
    QDockWidget *bgLibraryContainer;
    QDockWidget *libraryContainer;


    // Actions for the FILE Menu
    QAction *actionNewStrip;
    QAction *actionOpenStrip;
    QAction *actionCloseStrip;
    QAction *actionSave;
    QAction *actionSaveAs;
    QAction *actionExportStrip;
    QAction *actionExit;

    // Actions for the VIEW Meny
    QAction *actionPreviewStrip;

    // Actions for the PANEL Menu
    QAction *actionInsertPanel;
    QAction *actionDeletePanel;
    QAction *actionClearBackground;
    QAction *actionPanelOptions;
    QAction *actionPreferences;

    // Actions for the CHARACTER Menu
    QAction *actionNewCharacter;
    QAction *actionInsertCharacter;

    // Actions for the LIBRARY Menu
    QAction *actionOpenLibrary;
    QAction *actionBubbleDialog;

    // Actions for the HELP Menu
    QAction *actionAbout;
};

#endif
```

bubbletail.cpp

```
#include "bubbletail.h"

BubbleTail::BubbleTail(const QString &direction,
QGraphicsItem *parent)
    : QGraphicsItem(parent)
{
    setFlags(ItemIsSelectable | ItemIsMovable |
ItemIsFocusable);
    dir = direction;
}

QRectF BubbleTail::boundingRect() const
{
    return QRectF(0, 0, 20, 40);
}

void BubbleTail::paint(QPainter *painter, const
QStyleOptionGraphicsItem *option, QWidget *widget)
{
    Q_UNUSED(option);
    Q_UNUSED(widget);

    QPainterPath path;
    painter->setBrush(Qt::white);
    painter->setPen(QPen(Qt::black, 1, Qt::SolidLine,
Qt::FlatCap, Qt::RoundJoin));

    path.moveTo(0, 0);
    path.quadTo(2, 30, 20, 40);
    path.quadTo(10, 15, 20, 0);
    painter->drawPath(path);
    painter->setBrush(Qt::black);
}
```

bubble.cpp

```
#include "bubble.h"
#include "bubbletail.h"
```

```
Bubble::Bubble(const QString &speech, const QString &uri,
const QString &dir, const int width, QGraphicsItem *parent)
    : QGraphicsItem(parent), content(speech), type(uri),
direction(dir)
{
    setFlags(ItemIsSelectable | ItemIsMovable |
ItemIsFocusable);
    frameWidth = width;

    tail = new BubbleTail(direction, this);
    tail->setData(1, "bubbletail");
    tail->setZValue(1);
    tail->setPos((QRectF(boundingRect()).width())/2,
(QRectF(boundingRect()).height()/2)-1);
}

QRectF Bubble::boundingRect() const
{
    QFontMetrics meter(QFont("Verdana", 14, -1, false));
    QRectF box(meter.boundingRect(QRect(0, 0 ,frameWidth,
0), Qt::AlignCenter | Qt::TextWordWrap, content));
    box.adjust(-10, -10, 10, 10);
    return box;
}

void Bubble::paint(QPainter *painter, const
QStyleOptionGraphicsItem *option, QWidget *widget)
{
    Q_UNUSED(option);
    Q_UNUSED(widget);

    painter->setFont(QFont("Helvetica", 14, -1, false));
    painter->setBrush(Qt::white);
    painter->setPen(QPen(Qt::black, 1, Qt::SolidLine,
Qt::FlatCap, Qt::RoundJoin));

    if(type == "narration")
    {
        painter->drawRect(QRectF(boundingRect()));
    }
    else if(type == "speech")
    {
        painter->drawRoundRect(QRectF(boundingRect()), 70,
70);
    }
    else if(type == "thought")
    {
        QPainterPath path;
    }
    else if(type == "exclamation"){

    }
    painter->setBrush(Qt::black);
    painter-
>drawText(QRectF(boundingRect()).adjusted(10,10,-10,-10),
Qt::AlignCenter | Qt::TextWordWrap, content);
}
```

character.cpp

```
#include <QGraphicsSvgItem>

#include "character.h"

Character::Character(const QString &name, const QString
&gender, const QString &exp, QColor ugarment, QColor skin,
QColor lgarment, QGraphicsItem *parent)
    : QGraphicsItem(parent)
{
    Q_INIT_RESOURCE(images);

    skinColor = skin;
    upperGarmentColor = ugarment;
    lowerGarmentColor = lgarment;

    setData(1, "character");
    setData(2, name);
    setData(3, gender);
    setData(4, exp);
    setData(5, ugarment);
    setData(6, skin);
    setData(7, lgarment);

    this->name = name;

    // BODY

    QPainterPath torsoPath;
    torsoPath.moveTo(79.75,0.5);
    torsoPath.lineTo(20.5,20);
    torsoPath.lineTo(20.5,20.5);
    torsoPath.lineTo(20,20.5);
    torsoPath.cubicTo(9.193253,20.5,0.5,29.193251,0.5,40);
    torsoPath.cubicTo(0.5,50.806747,9.1932524,59.500002,2
0,59.5);
    torsoPath.lineTo(40.375,59.5);
    torsoPath.lineTo(40.375,139.625);
    torsoPath.lineTo(40.5,139.625);
    torsoPath.cubicTo(40.499561,139.71929,40.5,139.81186,
40.5,139.90625);
    torsoPath.cubicTo(40.5,172.81058,67.189425,199.5,100.0
9375,199.5);
    torsoPath.cubicTo(132.99808,199.5,159.71875,172.81058
,159.71875,139.90625);
    torsoPath.cubicTo(159.71875,138.87799,159.67653,137.8
5928,159.625,136.84375);
    torsoPath.lineTo(159.625,59.5);
    torsoPath.lineTo(180,59.5);
    torsoPath.cubicTo(190.80675,59.5,199.5,50.806745,199.5
,40);
    torsoPath.cubicTo(199.5,29.193253,190.80674,20.5,180,2
0.5);
    torsoPath.lineTo(179.5,20.5);
    torsoPath.lineTo(179.5,20);
    torsoPath.lineTo(120.25,0.5);
    torsoPath.lineTo(119.5,0.5);
    torsoPath.lineTo(100.5,0.5);
    torsoPath.lineTo(99.5,0.5);
    torsoPath.lineTo(79.75,0.5);

    torso = new QGraphicsPathItem(torsoPath, this);
    torso->setData(0, "torso");
    torso->setBrush(upperGarmentColor);

    // NECK
    QPainterPath neckPath;
    neckPath.moveTo(-19.40625,-119.59375);
    neckPath.lineTo(-19.59375,-82);
    neckPath.cubicTo(-19.52740126,-81.65725,-19.5,-
80.324685,-19.5,-80);
    neckPath.lineTo(-19.5,-20);
    neckPath.cubicTo(-19.5,-9.80505,-10.1949545,-0.5,0,-0.5);
    neckPath.cubicTo(10.805046,-0.5,19.5,-9.80505,19.5,-20);
    neckPath.lineTo(19.5,-80);
    neckPath.cubicTo(19.5,-80.324685,19.472599,-
81.65725,19.40625,-82);
    neckPath.lineTo(19.40625,-119.59375);

    neck = new QGraphicsPathItem(neckPath, torso);
    neck->setData(1, "neck");
    neck->setBrush(skinColor);

    // HIPS PATH
    QPainterPath hipsPath;
    hipsPath.moveTo(-59.4375,0.4375);
    hipsPath.lineTo(-59.4375, 89.25);
    hipsPath.lineTo(-59.6875, 89.25);
    hipsPath.cubicTo(-59.68113173,89.501191, -59.6875,
89.747284, -59.6875, 90);
    hipsPath.cubicTo(-59.6875, 106.18086,
-46.819137,119.3125,-30,119.3125);
    hipsPath.cubicTo(-29.105298,119.3125,-
29.207463,119.31361,-29.3125,119.3125);
    hipsPath.lineTo(-29.3125,119.6875);
    hipsPath.lineTo(30.09375,119.6875);
    hipsPath.lineTo(30.09375,119.3125);
    hipsPath.cubicTo(46.23111,119.26136,59.3125,106.14927
,59.3125,90);
```

```
    hipsPath.cubicTo(59.3125,89.747284,59.31887,89.501191
,59.3125,89.25);
    hipsPath.lineTo(59.5625,89.25);
    hipsPath.lineTo(59.5625,0.4375);
    hipsPath.lineTo(-59.4375,0.4375);

    hips = new QGraphicsPathItem(hipsPath, torso);
    hips->setData(1, "hips");
    hips->setBrush(lowerGarmentColor);

    // HEAD

    QPainterPath headPath;
    headPath.addEllipse(-40, -100, 80, 100);

    head = new QGraphicsPathItem(headPath, neck);
    head->setBrush(skinColor);

    // FACE
    if(gender=="Male")
            hair = new
QGraphicsSvgItem(":/bodyelements/hair-boy.svg", head);
            else
                        hair = new
QGraphicsSvgItem(":/bodyelements/hair-girl.svg", head);

            hair->setPos(-80,-120);
    eyes = new
QGraphicsSvgItem(":/bodyelements/eyes.svg", head);
    eyes->setPos(-40,-60);
    ears = new QGraphicsSvgItem(":/bodyelements/ears.svg",
head);
    ears->setPos(-50,-60);
    nose = new QGraphicsSvgItem(":/bodyparts/nose-simple-
front.svg", head);
    nose->setPos(0,0);

    if(exp=="Happy")
            lips = new
QGraphicsSvgItem(":/bodyelements/lips-happy.svg", head);
    else if(exp=="Smile")
            lips = new
QGraphicsSvgItem(":/bodyelements/lips-smile.svg", head);
    else if(exp=="Neutral")
            lips = new
QGraphicsSvgItem(":/bodyelements/lips.svg", head);
    else if(exp=="Sad")
            lips = new
QGraphicsSvgItem(":/bodyelements/lips-sad.svg", head);

    lips->setPos(-40,-30);


    //ARM PATH
    QPainterPath urarm;
    urarm.moveTo(-19.40625,0.59375);
    urarm.lineTo(-19.59375,38);
    urarm.cubicTo(-19.52740126,38.65725,-19.5,39.324685,-
19.5,40);
    urarm.lineTo(-19.5,100);
    urarm.cubicTo(-19.5,110.80505,-
10.1949545,119.5,0,119.5);
    urarm.cubicTo(10.805046,119.5,19.5,110.80505,19.5,100
);
    urarm.lineTo(19.5,40);
    urarm.cubicTo(19.5,39.324685,19.472599,38.65725,19.40
625,38);
    urarm.lineTo(19.40625,0.59375);
    QPainterPath ularm(urarm);
    QPainterPath llarm(urarm);
    QPainterPath lrarm(urarm);

    //LEG PATH
    QPainterPath urleg;
    urleg.moveTo(-29.4375,0.4375);
    urleg.lineTo(-29.4375,44.5625);
    urleg.lineTo(-29.5,44.5625);
    urleg.lineTo(-29.5,130);
    urleg.cubicTo(-29.5,146.33698,-18.66302,159.5,0,159.5);
    urleg.cubicTo(16.33698,159.5,29.499998,146.33698,29.5,
130);
    urleg.lineTo(29.5,44.5625);
    urleg.lineTo(29.5625,44.5625);
    urleg.lineTo(29.5625,0.4375);
    QPainterPath ulleg(urleg);
    QPainterPath llleg(urleg);
    QPainterPath lrleg(urleg);

    //PATH ITEMS ARM
    upperLeftArm = new QGraphicsPathItem(ularm, torso);
    upperLeftArm->setData(1, "ularm");
    upperLeftArm->setBrush(skinColor);
    upperRightArm = new QGraphicsPathItem(urarm, torso);
    upperRightArm->setData(1, "urarm");
    upperRightArm->setBrush(skinColor);
    lowerLeftArm = new QGraphicsPathItem(llarm,
upperLeftArm);
    lowerLeftArm->setBrush(skinColor);
    lowerRightArm = new QGraphicsPathItem(lrarm,
upperRightArm);
    lowerRightArm->setBrush(skinColor);

    //rightHand = new QGraphicsSvgItem(":/bodyparts/right-
hand.svg", lowerRightArm);
    //leftHand = new QGraphicsSvgItem(":/bodyparts/left-
hand.svg", lowerLeftArm);

    //PATH ITEMS LEG
    upperLeftLeg = new QGraphicsPathItem(ulleg, hips);
    upperLeftLeg->setBrush(skinColor);
    upperRightLeg = new QGraphicsPathItem(urleg, hips);
    upperRightLeg->setBrush(skinColor);
    lowerLeftLeg = new QGraphicsPathItem(llleg,
upperLeftLeg);
    lowerLeftLeg->setBrush(skinColor);
    lowerRightLeg = new QGraphicsPathItem(lrleg,
upperRightLeg);
    lowerRightLeg->setBrush(skinColor);

    //rightFoot = new QGraphicsSvgItem(":/bodyparts/right-
foot.svg", lowerRightLeg);
    //leftFoot  = new QGraphicsSvgItem(":/bodyparts/left-
foot.svg", lowerLeftLeg);

    torso->setPos(40,40);
    neck->setPos(100, 40);
    head->setPos(0, -40);
    hips->setPos(100,140);

    upperLeftArm->setPos(180,40);
    lowerLeftArm->setPos(0,100);
    upperRightArm->setPos(20,40);
    lowerRightArm->setPos(0,100);

    //rightHand->setPos(3,80);
    //leftHand->setPos(-3,80);

    //rightFoot->setPos(-3,140);
    //leftFoot->setPos(3,140);

    upperLeftLeg->setPos(30,90);
    upperRightLeg->setPos(-30,90);
    lowerLeftLeg->setPos(0,130);
    lowerRightLeg->setPos(0,130);

    torso->setZValue(1);
    neck->setZValue(1);
    head->setZValue(1);
    hips->setZValue(1);
    upperLeftArm->setZValue(1);
    lowerLeftArm->setZValue(1);
    upperRightArm->setZValue(1);
    lowerRightArm->setZValue(1);
    //rightHand->setZValue(1);
    //leftHand->setZValue(1);
    //rightFoot->setZValue(1);
    //leftFoot->setZValue(1);
    upperLeftLeg->setZValue(1);
```

```
    upperRightLeg->setZValue(1);
    lowerLeftLeg->setZValue(1);
    lowerRightLeg->setZValue(1);
    eyes->setZValue(1);
    ears->setZValue(1);
    nose->setZValue(1);
    lips->setZValue(1);
    hair->setZValue(1);

    torso->setFlags(QGraphicsItem::ItemIsMovable |
QGraphicsItem::ItemIsFocusable);
    neck->setFlags(QGraphicsItem::ItemIsFocusable);
    head->setFlags(QGraphicsItem::ItemIsFocusable);
    hips->setFlags(QGraphicsItem::ItemIsFocusable);
    upperLeftArm-
>setFlags(QGraphicsItem::ItemIsFocusable);
    lowerLeftArm->setFlags(QGraphicsItem::ItemIsFocusable);
    upperRightArm-
>setFlags(QGraphicsItem::ItemIsFocusable);
    lowerRightArm-
>setFlags(QGraphicsItem::ItemIsFocusable);
    upperLeftLeg->setFlags(QGraphicsItem::ItemIsFocusable);
    upperRightLeg-
>setFlags(QGraphicsItem::ItemIsFocusable);
    lowerLeftLeg->setFlags(QGraphicsItem::ItemIsFocusable);
    lowerRightLeg-
>setFlags(QGraphicsItem::ItemIsFocusable);
}

QRectF Character::boundingRect() const
{
    return QRectF();
}

void Character::paint(QPainter *painter,
                const QStyleOptionGraphicsItem *option,
QWidget *widget)
{
    Q_UNUSED(painter);
    Q_UNUSED(option);
    Q_UNUSED(widget);
}
```

dialogbubble.cpp

```
#include "dialogbubble.h"

DialogBubble::DialogBubble(QWidget *parent)
    : QDialog(parent, 0)
{
    setupUi(this);
}
```

dialogcharacter.cpp

```
#include <QString>
#include <QFileDialog>
#include <QMessageBox>

#include "dialogcharacter.h"

DialogCharacter::DialogCharacter(QWidget *parent)
    : QDialog(parent, 0)
{
    setupUi(this);
    scene = new QGraphicsScene(characterPreview);

    ugar = Qt::white;
    skin = QColor(230,223,202);
    lgar = Qt::red;

    skinLabel->setText(skin.name());
    skinLabel->setPalette(QPalette(skin));
    skinLabel->setAutoFillBackground(true);

            ugarLabel->setText(ugar.name());
    ugarLabel->setPalette(QPalette(ugar));
    ugarLabel->setAutoFillBackground(true);
    lgarLabel->setText(lgar.name());
    lgarLabel->setPalette(QPalette(lgar));
    lgarLabel->setAutoFillBackground(true);


    character = new Character("Stickman", "Female",
"Happy", Qt::white, QColor(230,223,202), Qt::red);
    characterPreview->setScene(scene);
    characterPreview->scene()->addItem(character);

    connect(updatePreview, SIGNAL(clicked()), this,
SLOT(updatePreviewWindow()));
    connect(skinButton, SIGNAL(clicked()), this,
SLOT(getSkinColor()));
    connect(ugarButton, SIGNAL(clicked()), this,
SLOT(getUgarColor()));
    connect(lgarButton, SIGNAL(clicked()), this,
SLOT(getLgarColor()));
    connect(insertCharacter, SIGNAL(clicked()), this,
SLOT(send()));
}

void DialogCharacter::updatePreviewWindow()
{
            character->~QGraphicsItem();
            character = new Character(charName->text(),
charGender->currentText(), charExpression->currentText(),
QColor(ugarLabel->text()), QColor(skinLabel->text()),
QColor(lgarLabel->text()));
            characterPreview->scene()->addItem(character);
}

void DialogCharacter::getSkinColor()
{
            QColor color = QColorDialog::getColor(skin, this);
    if (color.isValid()) {
        skinLabel->setText(color.name());
        skinLabel->setPalette(QPalette(color));
        skinLabel->setAutoFillBackground(true);
    }
}

void DialogCharacter::getUgarColor()
{
            QColor color = QColorDialog::getColor(ugar, this);
    if (color.isValid()) {
        ugarLabel->setText(color.name());
        ugarLabel->setPalette(QPalette(color));
        ugarLabel->setAutoFillBackground(true);
    }
}

void DialogCharacter::getLgarColor()
{
            QColor color = QColorDialog::getColor(lgar, this);
    if (color.isValid()) {
        lgarLabel->setText(color.name());
        lgarLabel->setPalette(QPalette(color));
        lgarLabel->setAutoFillBackground(true);
    }
}

void DialogCharacter::send()
{
            Character *toemit = character;
            QMatrix set;
            set.scale(.6,.6);
            toemit->children().value(0)->setMatrix(set);
    emit sendCharacter(toemit);
}
```

dialognew.cpp

```
#include "dialognew.h"

DialogNew::DialogNew(QWidget *parent)
    : QDialog(parent, 0)
{
    setupUi(this);
    connect(okButton, SIGNAL(clicked()), this,
SLOT(accept()));
```

```
    connect(cancelButton, SIGNAL(clicked()), this,
SLOT(reject()));
}
```

dialogoptions.cpp

```
#include "dialogoptions.h"

DialogOptions::DialogOptions(QWidget *parent)
    : QDialog(parent)
{
    setupUi(this);

    connect(okButton, SIGNAL(clicked()), this,
SLOT(accept()));
    connect(cancelButton, SIGNAL(clicked()), this,
SLOT(reject()));
}
```

dialogpreferences.cpp

```
#include "dialogpreferences.h"

DialogPreferences::DialogPreferences(const QString &title,
const QString &author, int spacingOfPanels, QWidget
*parent)
    : QDialog(parent)
{
    setupUi(this);
    authorText->setText(author);
    titleText->setText(title);
    panelSpacing->setValue(spacingOfPanels);

    connect(okButton, SIGNAL(clicked()), this,
SLOT(accept()));
    connect(cancelButton, SIGNAL(clicked()), this,
SLOT(reject()));
}
```

dialogpreviewstrip.cpp

```
#include "dialogpreviewstrip.h"
#include <QSizePolicy>

DialogPreviewStrip::DialogPreviewStrip(QImage image,
QWidget *parent)
    : QDialog(parent, 0)
{
            //Almost done. Resize IMAGE to fit window -- fit
width. window NOT RESIZEABLE. ZOOM function.
            setWindowTitle("Preview Comic Strip");

            QLabel *imageLabel = new QLabel;
    imageLabel->setBackgroundRole(QPalette::Base);
    imageLabel->setScaledContents(true);
    imageLabel->setPixmap(QPixmap::fromImage(image));

    QScrollArea *scrollArea = new QScrollArea(this);
    scrollArea->setBackgroundRole(QPalette::Dark);
    scrollArea->setMinimumSize(QSize(image.width(),
image.height()));
    scrollArea->setWidget(imageLabel);

    setFixedSize(QSize(image.width(), image.height()));
}
```

library.cpp

```
#include <QGraphicsItem>
#include <QMessageBox>

#include "ui_library.h"
#include "library.h"
#include "elements.h"

Library::Library(QWidget *parent)
     : QWidget(parent)
{
    setupUi(this);
    scene = new QGraphicsScene(itemPreviewWindow);
    itemPreviewWindow->setScene(scene);

    itemInPreview = 0;

    QTreeWidgetItem *headerProps = new
QTreeWidgetItem();
    headerProps->setText(0, "List of Items");
    propsList->setHeaderItem(headerProps);

    QTreeWidgetItem *headerBackgrounds = new
QTreeWidgetItem();
    headerBackgrounds->setText(0, "List of Backgrounds");
    backgroundsList->setHeaderItem(headerBackgrounds);

    connect(tabWidget, SIGNAL(currentChanged(int)), this,
SLOT(clearPreviewItem()));

    connect(insertButton, SIGNAL(clicked()), this,
SLOT(sendItem()));
    connect(setBackgroundButton, SIGNAL(clicked()), this,
SLOT(setBackground()));

    connect(propsList,
SIGNAL(itemClicked(QTreeWidgetItem*, int)), this,
SLOT(setPreviewWindow(QTreeWidgetItem*, int)));
    connect(backgroundsList,
SIGNAL(itemClicked(QTreeWidgetItem*, int)), this,
SLOT(setPreviewWindow(QTreeWidgetItem*, int)));

}

void Library::populateLibrary(QObject *plugin, const
QStringList &props, const QStringList &backgrounds)
{
    plugins.append(plugin);
    pluginNames.append(plugin->metaObject()-
>className());
    int index = plugins.size() - 1;

    if(!props.isEmpty()){
            QTreeWidgetItem *libraryName = new
QTreeWidgetItem();
            libraryName->setText(0, tr(plugin->metaObject()-
>className()));

            foreach (QString prop, props) {
                QTreeWidgetItem *newItem = new
QTreeWidgetItem(libraryName);
                newItem->setText(0, prop);
                newItem->setFlags(Qt::ItemIsSelectable |
Qt::ItemIsEnabled);
              }

               propsList->addTopLevelItem(libraryName);
            libraryName->setExpanded(true);
            libraryName->setFlags(Qt::ItemIsEnabled);
            }

            if(!backgrounds.isEmpty()){
            QTreeWidgetItem *libraryName = new
QTreeWidgetItem();
            libraryName->setText(0, tr(plugin->metaObject()-
>className()));

            foreach (QString background, backgrounds) {
                QTreeWidgetItem *newItem = new
QTreeWidgetItem(libraryName);
                newItem->setText(0, background);
                newItem->setFlags(Qt::ItemIsSelectable |
Qt::ItemIsEnabled);
              }

               backgroundsList-
>addTopLevelItem(libraryName);
            libraryName->setExpanded(true);
            libraryName->setFlags(Qt::ItemIsEnabled);
            }
}

void Library::setPreviewWindow(QTreeWidgetItem *item, int
column)
```

```
{
        if(item->parent()!=0)
        {
                QString name = item->text(column);

                int indexOfPlugin =
pluginNames.indexOf(item->parent()->text(column));


                if(tabWidget->currentIndex() == 0)
                {
                PropElement *element =
qobject_cast<PropElement *>(plugins.value(indexOfPlugin));
                itemPreviewWindow->scene()-
>removeItem(itemPreviewWindow->scene()->itemAt(0,0));
                itemInPreview = element-
>insertElement(name);

                QRectF kahon(itemInPreview-
>boundingRect());

                   qreal ratio;
                   if(kahon.height() > kahon.width())
                      ratio = (itemPreviewWindow-
>height()-10) / kahon.height();
                   else
                      ratio = (itemPreviewWindow-
>width()-10) / kahon.width();

                   itemInPreview->scale(ratio, ratio);
                }
                else
                {
                   BackgroundElement *element =
qobject_cast<BackgroundElement
*>(plugins.value(indexOfPlugin));
                   itemPreviewWindow->scene()-
>removeItem(itemPreviewWindow->scene()->itemAt(0,0));
                   itemInPreview = element-
>queryBackground(name);

                   QRectF kahon(itemInPreview-
>boundingRect());

                   qreal ratio;
                   if(kahon.height() > kahon.width())
                      ratio = (itemPreviewWindow-
>height()-10) / kahon.height();
                   else
                      ratio = (itemPreviewWindow-
>width()-10) / kahon.width();

                   itemInPreview->scale(ratio, ratio);
                }

              itemPreviewWindow->scene()-
>addItem(itemInPreview);
              itemPreviewWindow->scene()->update();
   }
}

void Library::sendItem()
{
          QTreeWidgetItem *widgetItem = propsList-
>currentItem();
   if(widgetItem){
              // 2 vital information: itemName, itemLib
      QString name    = widgetItem->text(0);
      int indexOfPlugin = pluginNames.indexOf(widgetItem-
>parent()->text(0));

      PropElement *element = qobject_cast<PropElement
*>(plugins.value(indexOfPlugin)); //get plugin based on
itemLib
      QGraphicsItem *item = element->insertElement(name);
            // queries itemLib based on itemName and returns
ITEM
      emit emitItem(item);       // emit ITEM
   }
}

void Library::setBackground()
{
          QTreeWidgetItem *widgetItem = backgroundsList-
>currentItem();
   if(widgetItem){
      QString name    = widgetItem->text(0);
      int indexOfPlugin = pluginNames.indexOf(widgetItem-
>parent()->text(0));

      BackgroundElement *element =
qobject_cast<BackgroundElement
*>(plugins.value(indexOfPlugin));
      QGraphicsItem *item = element-
>queryBackground(name);
                emit emitBackground(item);
   }
}

void Library::clearPreviewItem()
{
          if(itemInPreview)
          {
                itemInPreview->~QGraphicsItem();
                itemInPreview = 0;
                itemPreviewWindow->scene()-
>update();
          }
}
```

main.cpp

```
#include <QApplication>
#include <QSplashScreen>
#include <QIcon>

#include "mainwindow.h"

int main(int argc, char *argv[])
{
   Q_INIT_RESOURCE(images);
   QApplication app(argc, argv);
   QPixmap pixmap(":/images/splash.png");
   QPixmap icon(":/images/icon.png");
   QSplashScreen splash(pixmap);
   splash.show();

   MainWindow mainWin;
   mainWin.showMaximized();
   mainWin.setWindowIcon(QIcon(icon));
   return app.exec();
}
```

mainwindow.cpp

```
#include <QtGui>
#include <QtSvg>

#include "mainwindow.h"
#include "stage.h"

#include "toolbox.h"
#include "propslib.h"
#include "bglib.h"
#include "library.h"

#include "bubble.h"
#include "elements.h"

Q_IMPORT_PLUGIN(dibuho_basiclibrary);

MainWindow::MainWindow()
{
   createStage();
   createActions();
   createMenus();
   createStatusBar();
   createTools();
   createToolbars();
```

```
    loadLibrary();

    setCurrentFile("");
    readSettings();
}

void MainWindow::closeEvent(QCloseEvent *event)
{
    if (maybeSave()) {
        writeSettings();
        event->accept();
    } else {
        event->ignore();
    }
}

void MainWindow::newCS()
{
    DialogNew *create = new DialogNew(this);
    if(create->exec()){
        if(maybeSave()){
            int numberOfPanels = create->panelQty->value();
            int panelSpacing = create->panelSpacing->value();
            stage->setNewStage(create->titleText->text(),
create->authorText->text(), numberOfPanels,
panelSpacing);
            setCurrentFile("");
        }
    }
}

void MainWindow::openCS()
{
    if(maybeSave()){
        QString fileName = QFileDialog::getOpenFileName(this,
tr("Open Comic Strip"), "../saved", tr("Dibuho Comic Strip
(*.dibuho)"));
            if(!fileName.isEmpty())
                loadFile(fileName);
    }
}

bool MainWindow::saveCS()
{
    if (currentFile.isEmpty()) {
        return saveAsCS();
    } else {
        return saveFile(currentFile);
    }
}

bool MainWindow::saveAsCS()
{
    QString fileName = QFileDialog::getSaveFileName(this,
tr("Save Comic Strip"), "../saved", tr("Dibuho Comic Strip
(*.dibuho)"));
    if (fileName.isEmpty())
        return false;
    return saveFile(fileName);
}

bool MainWindow::maybeSave()
{
    if (stage->isModified()) {
         QMessageBox::StandardButton ret;
         ret = QMessageBox::warning(this, tr("Comic Strip
Illustrator"),
                      tr("The document has been modified.\n"
                         "Do you want to save your changes?"),
                      QMessageBox::Save | QMessageBox::Discard |
QMessageBox::Cancel);
         if (ret == QMessageBox::Save)
             return saveCS();
         else if (ret == QMessageBox::Cancel)
             return false;
    }
    return true;
}

void MainWindow::previewCS()
{
            QImage image = stage->exportStage();
            /*
            QLabel *imageLabel = new QLabel;
    imageLabel->setBackgroundRole(QPalette::Base);
    imageLabel->setScaledContents(true);
    imageLabel->setPixmap(QPixmap::fromImage(image));

    QScrollArea *scrollArea = new QScrollArea(this);
    scrollArea->setBackgroundRole(QPalette::Dark);
    //scrollArea->setFixedSize(QSize(500, 500));
    scrollArea->setWidget(imageLabel);

            setCentralWidget(scrollArea);
            */
            DialogPreviewStrip preview(image, this);
            preview.exec();
}


bool MainWindow::exportCS()
{
    QString filename = QFileDialog::getSaveFileName(this,
tr("Export Comic Strip"), "../export", tr("PNG (*.png)"));    //
Asks for FILENAME

    if (filename.isEmpty())
        return false;

    QFile file(filename);
    if (!file.open(QFile::WriteOnly | QFile::Text)) {                //
Tries to open FILENAME
        QMessageBox::warning(this, tr("Save Error"),
                             tr("Cannot write file %1:\n%2.")
                             .arg(filename)
                             .arg(file.errorString()));
        return false;
    }

    QImage image = stage->exportStage();       // Gets IMAGE
from Stage

    image.save(filename, "PNG");          // Saves IMAGE in
PNG format
    return true;
}

void MainWindow::openLibrary()
{
    // Opens Library dialog
}

void MainWindow::insertCharacter()
{

}

void MainWindow::insertBubble()
{
    DialogBubble *bubble = new DialogBubble(this);
    if(bubble->exec()){
        QString content = bubble->bubbleContent-
>toPlainText();
        QString type;
        QString direction;
        if(bubble->radioNarrate->isChecked())
            type = "narration";
        else if(bubble->radioSpeech->isChecked())
            type = "speech";

        int width = bubble->bubbleWidth->value();

        Bubble *speech = new Bubble(content, type, direction,
width, 0);
        speech->setData(1, "bubble");
        speech->setData(2, content);
        speech->setData(3, type);
        speech->setData(4, width);
```

```
        speech->setData(5, direction);
        stage->acceptItem(speech);
    }
}


void MainWindow::newCharacter()
{
    DialogCharacter *character = new DialogCharacter(this);
    connect(character,
SIGNAL(sendCharacter(QGraphicsItem*)), stage,
SLOT(acceptItem(QGraphicsItem*)));
    if(character->exec())
    {

    }
}

void MainWindow::about()
{
    QMessageBox::about(this, tr("Dibuho"),
    tr("<b>Dibuho Comic Strip Creator</b><br>Beta
Version<br><br>Written by:<br>Ivan Francisco
Eugenio<br>i.eugenio@yahoo.com"));
}

bool MainWindow::saveFile(const QString &filename)
{
    QFile file(filename);
    if (!file.open(QFile::WriteOnly | QFile::Text)) {
        QMessageBox::warning(this, tr("Save Error"),
                             tr("Cannot write file %1:\n%2.")
                             .arg(filename)
                             .arg(file.errorString()));
        return false;
    }

    QTextStream out(&file);
    QApplication::setOverrideCursor(Qt::WaitCursor);
    out << stage->decodeStage();    // Output Stage contents
in PLAINTEXT
    QApplication::restoreOverrideCursor();

    stage->setModified(false);
    setCurrentFile(filename);
    statusBar()->showMessage(tr("File saved"), 2000);
    return true;
}

void MainWindow::loadFile(const QString &filename)
{
    QFile file(filename);
    if (!file.open(QFile::ReadOnly | QFile::Text)) {
        QMessageBox::warning(this, tr("Load Error"),
                             tr("Cannot read file %1:\n%2.")
                             .arg(filename)
                             .arg(file.errorString()));
        return;
    }

    QTextStream in(&file);
    QApplication::setOverrideCursor(Qt::WaitCursor);
    stage->encodeStage(in);      // Sets the contents in the
STAGE
    QApplication::restoreOverrideCursor();

    setCurrentFile(filename);
    statusBar()->showMessage(tr("File loaded"), 2000);
    //QMessageBox::about(this, tr("About Comic Strip
Illustrator"),tr("%1").arg(stage->panelSpacing));
}

void MainWindow::setCurrentFile(const QString &fileName)
{
    currentFile = fileName;
    setWindowModified(false);

    QString shownName;
    if (currentFile.isEmpty())
        shownName = "Untitled.dibuho";
    else
        shownName = fileName;

    setWindowTitle(tr("%1[*] - %2 by %3
[%4]").arg(tr("Dibuho")).arg(stage-
>comicStripTitle).arg(stage-
>comicStripAuthor).arg(shownName));
}

void MainWindow::loadLibrary()
{
    foreach (QObject *plugin,
QPluginLoader::staticInstances())
        populateLibrary(plugin);

    pluginsDir = QDir(qApp->applicationDirPath());

    #if defined(Q_OS_WIN)
        if (pluginsDir.dirName().toLower() == "debug" ||
pluginsDir.dirName().toLower() == "release")
            pluginsDir.cdUp();
    #elif defined(Q_OS_MAC)
        if (pluginsDir.dirName() == "MacOS") {
            pluginsDir.cdUp();
            pluginsDir.cdUp();
            pluginsDir.cdUp();
        }
    #endif

    pluginsDir.cd("plugins");

    foreach (QString fileName,
pluginsDir.entryList(QDir::Files)) {
        QPluginLoader
loader(pluginsDir.absoluteFilePath(fileName));
        QObject *plugin = loader.instance();
        if (plugin) {
            populateLibrary(plugin);
            pluginFileNames += fileName;
        }
    }
}

void MainWindow::populateLibrary(QObject *plugin)
{
    PropElement *element = qobject_cast<PropElement
*>(plugin);
    if (element){
        propsLibrary->addToLibrary(plugin, element->props());
    }

    BackgroundElement *bg =
qobject_cast<BackgroundElement *>(plugin);
    if (bg){
        bgLibrary->addToLibrary(plugin, bg->backgrounds());
    }

    library->populateLibrary(plugin, element->props(), bg-
>backgrounds());

}

void MainWindow::createMenus()
{
    fileMenu = menuBar()->addMenu(tr("&File"));
    fileMenu->addAction(actionNewStrip);
    fileMenu->addAction(actionOpenStrip);
    fileMenu->addSeparator();
    fileMenu->addAction(actionSave);
    fileMenu->addAction(actionSaveAs);
    fileMenu->addAction(actionExportStrip);
    fileMenu->addSeparator();
    fileMenu->addAction(actionExit);

    panelMenu = menuBar()->addMenu(tr("&Panel"));
    panelMenu->addAction(actionInsertPanel);
    panelMenu->addAction(actionDeletePanel);
    panelMenu->addSeparator();
```

```
    panelMenu->addAction(actionClearBackground);
    panelMenu->addSeparator();
    panelMenu->addAction(actionPreferences);
    panelMenu->addAction(actionPanelOptions);

    characterMenu = menuBar()->addMenu(tr("&Character"));
    characterMenu->addAction(actionNewCharacter);

    libraryMenu = menuBar()->addMenu(tr("&Bubble"));
    libraryMenu->addAction(actionBubbleDialog);
    libraryMenu->addSeparator();

    viewMenu = menuBar()->addMenu(tr("&View"));
    viewMenu->addAction(actionPreviewStrip);
    viewMenu->addSeparator();


    helpMenu = menuBar()->addMenu(tr("&Help"));
    helpMenu->addAction(actionAbout);
}

void MainWindow::createActions()
{

    // FILE actions
    actionNewStrip = new QAction(tr("&New Comic Strip"),
this);
    actionNewStrip->setShortcut(tr("Ctrl+N"));
    actionNewStrip->setIcon(QIcon(":/images/new.png"));
    actionNewStrip->setStatusTip(tr("Create a new comic
strip"));
    connect(actionNewStrip, SIGNAL(triggered()), this,
SLOT(newCS()));

    actionOpenStrip = new QAction(tr("&Open..."), this);
    actionOpenStrip->setShortcut(tr("Ctrl+O"));
    actionOpenStrip->setIcon(QIcon(":/images/open.png"));
    actionOpenStrip->setStatusTip(tr("Open an existing comic
strip"));
    connect(actionOpenStrip, SIGNAL(triggered()), this,
SLOT(openCS()));

    actionSave = new QAction(tr("&Save"), this);
    actionSave->setShortcut(tr("Ctrl+S"));
    actionSave->setIcon(QIcon(":/images/save.png"));
    actionSave->setStatusTip(tr("Save the current comic
strip"));
    connect(actionSave, SIGNAL(triggered()), this,
SLOT(saveCS()));

    actionSaveAs = new QAction(tr("Save &As..."), this);
    actionSaveAs->setShortcut(tr("Ctrl+Alt+S"));
    actionSaveAs->setStatusTip(tr("Save the current comic
strip with a file name"));
    connect(actionSaveAs, SIGNAL(triggered()), this,
SLOT(saveAsCS()));

    actionExportStrip = new QAction(tr("&Export as PNG"),
this);
    actionExportStrip->setShortcut(tr("Ctrl+E"));
    actionExportStrip->setIcon(QIcon(":/images/export.png"));
    actionExportStrip->setStatusTip(tr("Export comic strip in
PNG format"));
    connect(actionExportStrip, SIGNAL(triggered()), this,
SLOT(exportCS()));

    actionExit = new QAction(tr("E&xit"), this);
    actionExit->setShortcut(tr("Ctrl+Q"));
    actionExit->setStatusTip(tr("Exit the application"));
    connect(actionExit, SIGNAL(triggered()), this,
SLOT(close()));

    // PANEL actions
    actionInsertPanel = new QAction(tr("Insert New Panel"),
this);
    actionInsertPanel->setShortcut(tr("Ctrl+M"));
    actionInsertPanel->setStatusTip(tr("Insert a blank panel to
the Comic Strip"));
    connect(actionInsertPanel, SIGNAL(triggered()), stage,
SLOT(insertPanel()));

    actionDeletePanel = new QAction(tr("Delete Panel"), this);
    actionDeletePanel->setShortcut(tr("Ctrl+K"));
    actionDeletePanel->setStatusTip(tr("Delete the panel
along with all of its contents"));
    connect(actionDeletePanel, SIGNAL(triggered()), stage,
SLOT(deletePanel()));

    actionClearBackground = new QAction(tr("Clear
Background"), this);
    actionClearBackground->setShortcut(tr("Ctrl+R"));
    actionClearBackground->setStatusTip(tr("Clears the
background of the current panel"));
    connect(actionClearBackground, SIGNAL(triggered()),
stage, SLOT(clearBackground()));

    actionPanelOptions = new QAction(tr("Options"), this);
    actionPanelOptions->setShortcut(tr("Ctrl+B"));
    actionPanelOptions->setStatusTip(tr("Opens dialog for
panel options"));
    connect(actionPanelOptions, SIGNAL(triggered()), stage,
SLOT(optionsPanel()));

    actionPreferences = new QAction(tr("Preferences"), this);
    actionPreferences->setShortcut(tr("Ctrl+E"));
    actionPreferences->setStatusTip(tr("Opens dialog for
general preferences"));
    connect(actionPreferences, SIGNAL(triggered()), this,
SLOT(preferences()));

    // VIEW actions
    actionPreviewStrip = new QAction(tr("Preview"), this);
    actionPreviewStrip->setShortcut(tr("Ctrl+R"));
    actionPreviewStrip->setStatusTip(tr("Preview current
Comic Strip"));
    connect(actionPreviewStrip, SIGNAL(triggered()), this,
SLOT(previewCS()));

    // CHARACTER actions
    actionNewCharacter = new QAction(tr("New Character"),
this);
    actionNewCharacter->setShortcut(tr("Ctrl+C"));
    actionNewCharacter->setStatusTip(tr("Create new
character"));
    connect(actionNewCharacter, SIGNAL(triggered()), this,
SLOT(newCharacter()));

    actionInsertCharacter = new QAction(tr("Insert
Character"), this);
    actionInsertCharacter->setShortcut(tr("Ctrl+I"));
    actionInsertCharacter->setStatusTip(tr("Insert a character
in the Panel"));
    connect(actionInsertCharacter, SIGNAL(triggered()), this,
SLOT(insertCharacter()));

    // LIBRARY actions
    actionBubbleDialog = new QAction(tr("Insert Bubble"),
this);
    actionBubbleDialog->setShortcut(tr("Ctrl+B"));
    actionBubbleDialog->setStatusTip(tr("Insert a Speech
Bubble"));
    connect(actionBubbleDialog, SIGNAL(triggered()), this,
SLOT(insertBubble()));

    actionOpenLibrary = new QAction(tr("&Open Library"),
this);
    actionOpenLibrary->setShortcut(tr("Ctrl+L"));
    actionOpenLibrary->setStatusTip(tr("Opens the Library"));
    connect(actionOpenLibrary, SIGNAL(triggered()), this,
SLOT(openLibrary()));

    // ABOUT actions
    actionAbout = new QAction(tr("&About"), this);
    actionAbout->setStatusTip(tr("Show the application's
About box"));
    connect(actionAbout, SIGNAL(triggered()), this,
SLOT(about()));
```

```
}

void MainWindow::createStatusBar()
{
    statusBar()->showMessage(tr("Ready"));
}

void MainWindow::createTools()
{
    toolbox = new ToolBox(this);
    toolboxContainer = new QDockWidget(tr("Tool Box"),
this);
    toolboxContainer-
>setAllowedAreas(Qt::LeftDockWidgetArea);
    toolboxContainer-
>setFeatures(QDockWidget::DockWidgetFloatable |
QDockWidget::DockWidgetMovable );
    toolboxContainer->setMinimumWidth(70);
    toolboxContainer->setWidget(toolbox);
    addDockWidget(Qt::LeftDockWidgetArea,
toolboxContainer);

    propsLibrary = new PropsLib();
    propsLibraryContainer = new QDockWidget(tr("Props
Library"), this);
    propsLibraryContainer-
>setAllowedAreas(Qt::RightDockWidgetArea);
    propsLibraryContainer-
>setFeatures(QDockWidget::DockWidgetFloatable |
QDockWidget::DockWidgetMovable |
QDockWidget::DockWidgetClosable);
    propsLibraryContainer->setMinimumWidth(150);
    propsLibraryContainer->setWidget(propsLibrary);
    propsLibraryContainer->setFloating(true);
    addDockWidget(Qt::RightDockWidgetArea,
propsLibraryContainer);

    bgLibrary = new BgLib();
    bgLibraryContainer = new QDockWidget(tr("Backgrounds
Library"), this);
    bgLibraryContainer-
>setAllowedAreas(Qt::AllDockWidgetAreas);
    bgLibraryContainer-
>setFeatures(QDockWidget::DockWidgetFloatable |
QDockWidget::DockWidgetMovable |
QDockWidget::DockWidgetClosable );
    bgLibraryContainer->setMinimumWidth(150);
    bgLibraryContainer->setWidget(bgLibrary);
    bgLibraryContainer->setFloating(true);
    addDockWidget(Qt::RightDockWidgetArea,
bgLibraryContainer);

    library = new Library(this);
    libraryContainer = new QDockWidget(tr("Library"), this);
    libraryContainer-
>setAllowedAreas(Qt::RightDockWidgetArea);
    libraryContainer-
>setFeatures(QDockWidget::DockWidgetFloatable |
QDockWidget::DockWidgetMovable);
    libraryContainer->setMinimumWidth(170);
    libraryContainer->setWidget(library);
    addDockWidget(Qt::RightDockWidgetArea,
libraryContainer);

    connect(library, SIGNAL(emitItem(QGraphicsItem*)),
stage, SLOT(acceptItem(QGraphicsItem*)));
    connect(library,
SIGNAL(emitBackground(QGraphicsItem*)), stage,
SLOT(setBackground(QGraphicsItem*)));

    connect(toolbox, SIGNAL(changeTool(QString)), stage,
SLOT(getTool(QString)));
    connect(propsLibrary,
SIGNAL(sendToStage(QGraphicsItem*)), stage,
SLOT(acceptItem(QGraphicsItem*)));
    connect(bgLibrary,
SIGNAL(setPanelBackground(QGraphicsItem*)), stage,
SLOT(setBackground(QGraphicsItem*)));
    connect(stage,
SIGNAL(requestBackground(QString,QString)), bgLibrary,
SLOT(emitRequestedItem(QString,QString)));
    connect(stage,
SIGNAL(requestItem(QString,QString,QMatrix,QPointF,int)),
propsLibrary,
SLOT(emitRequestedItem(QString,QString,QMatrix,QPointF,in
t)));
}

void MainWindow::createToolbars()
{
    fileToolbar = addToolBar(tr("File"));
    fileToolbar->setMovable(false);
    fileToolbar->addAction(actionNewStrip);
    fileToolbar->addAction(actionOpenStrip);
    fileToolbar->addAction(actionSave);
    fileToolbar->addAction(actionExportStrip);
}

void MainWindow::createStage()
{
    stage = new Stage;
    stage->setRenderHints(QPainter::Antialiasing |
QPainter::SmoothPixmapTransform);
    setCentralWidget(stage);
}

void MainWindow::preferences()
{
    DialogPreferences *create = new
DialogPreferences(stage->comicStripTitle, stage-
>comicStripAuthor, stage->panelSpacing, this);
    if(create->exec())
    {
        stage->panelSpacing      = create->panelSpacing-
>value();
        stage->comicStripTitle   = create->titleText->text();
        stage->comicStripAuthor  = create->authorText-
>text();
        stage->modified = true;
        setCurrentFile(currentFile);
    }
}

void MainWindow::readSettings()
{
    QSettings settings("Dibuho", "Illustrator");
    QPoint pos = settings.value("pos", QPoint(200,
200)).toPoint();
    QSize size = settings.value("size", QSize(800,
600)).toSize();
    resize(size);
    move(pos);
}

void MainWindow::writeSettings()
{
    QSettings settings("Dibuho", "Illustrator");
    settings.setValue("pos", pos());
    settings.setValue("size", size());
}
```

panelnav.cpp

```
#include "panelnav.h"

PanelNav::PanelNav(QWidget *parent)
    : QWidget(parent)
{
    setupUi(this);
}
```

stage.cpp

```
#include <QGraphicsScene>
#include <QGraphicsSvgItem>
#include <QSpinBox>
#include <QMessageBox>
```

```
#include "stage.h"

#include "elements.h"
#include "bubble.h"
#include "dialogoptions.h"
#include "panelnav.h"

#include "character.h"

Stage::Stage()
{
   setNewStage("Untitled", "Unknown", 1, 30);
   currentScene = panels.value(0);
   setScene(currentScene);
            setRenderHints(QPainter::Antialiasing |
QPainter::SmoothPixmapTransform);

   nav = new PanelNav(this);
   nav->display->setText("1");
   connect(nav->next, SIGNAL(clicked()), this,
SLOT(getNextScene()));
   connect(nav->prev, SIGNAL(clicked()), this,
SLOT(getPrevScene()));
   connect(this, SIGNAL(valueChanged(QString)), nav-
>display, SLOT(setText(QString)));

   setMinimumSize(400, 400);
}

void Stage::setNewStage(const QString &title, const QString
&author, const int panelQty, const int panelsSpc)
{
   // Clears STAGE
   panels.clear();
   panelW.clear();
   layerCounter.clear();
   libs.clear();
   libCounter.clear();

   // Load new values for stage
   comicStripTitle     = title;
   comicStripAuthor    = author;
   panelCount          = panelQty;
   panelSpacing        = panelsSpc;
   panelHeight         = 400;
   topMargin           = 100;
   scaleValue          = 1.0;

   // Sets values for border
   borders.setColor(Qt::black);
   borders.setWidth(6);
   borders.setStyle(Qt::SolidLine);
   borders.setJoinStyle(Qt::MiterJoin);

   // Initialize Panels
   for(int i = 0; i < panelQty; i++){
      QGraphicsScene *newPanel = new
QGraphicsScene(this);
      newPanel->setSceneRect(0, 0, panelHeight * 1.25,
panelHeight);
      panels.append(newPanel);
      panelW.append(1.25);
      layerCounter.append(1);
   }

   currentScene        =   panels.value(0);
   currentSceneIndex   =   0;
   modified            =   false;
   setScene(currentScene);
}

void Stage::acceptItem(QGraphicsItem *element)
{
   QString library_used = element->data(0).toString();
   if(library_used != ""){
      if(!libs.contains(library_used)){
         libs.append(library_used);
         libCounter.append(1);
      }else{
            int index = libs.indexOf(library_used);
            libCounter.replace(index, libCounter.value(index)
+ 1);
      }
   }

   element-
>setZValue(layerCounter.value(currentSceneIndex));
   layerCounter.replace(currentSceneIndex,
layerCounter.value(currentSceneIndex) + 1);
   scene()->addItem(element);
   scene()->update();
   modified = true;
}

void Stage::setBackground(QGraphicsItem *element)
{
   QString library_used = element->data(0).toString();
   if(library_used != "")
   {
      if(!libs.contains(library_used))
      {
         libs.append(library_used);
         libCounter.append(1);
      }
      else
      {
            int index = libs.indexOf(library_used);
            libCounter.replace(index, libCounter.value(index)
+ 1);
      }
   }

   element->setZValue(0);

   qreal width  = element->boundingRect().width();
   qreal height = element->boundingRect().height();
   qreal wratio = (panelHeight *
panelW.value(currentSceneIndex)) / width;
   qreal hratio = panelHeight / height;

   QList<QGraphicsItem *> itemsList = scene()->items();

   foreach(QGraphicsItem *item, itemsList)
   {
         if(item->zValue()==0)
            item->~QGraphicsItem();
   }

   QMatrix bgMatrix;
   bgMatrix.scale(wratio, hratio);
   element->setMatrix(bgMatrix);

   scene()->addItem(element);
   scene()->update();
   modified = true;
}

void Stage::calculateComicStripSize()
{
   comicStripWidth  = (int) (panelSpacing * (panels.size() +
1));
   comicStripHeight = 0;
   for (int i = 0; i < panels.size(); ++i)
   {
      comicStripWidth += (int) panels.value(i)->width();
      if(comicStripHeight < panels.value(i)->height())
         comicStripHeight = (int) panels.value(i)->height();
   }
   comicStripHeight += topMargin;
   comicStripHeight += panelSpacing;
}

QImage Stage::exportStage()
{
   // Calculates the comic strip size, to get proper width and
height.
   calculateComicStripSize();
```

```
    // Initializes an image and a respective PAINTER for it.
    QImage image(comicStripWidth, comicStripHeight,
QImage::Format_ARGB32_Premultiplied);
    QPainter p(&image);

    p.setRenderHint(QPainter::Antialiasing);

    // Header for exported Comic Srip. Writes the TITLE and
AUTHOR to the image file.
    QPainterPath title(QPointF(0, 0));
    QFont font("Verdana", 35);
    QFontMetrics fontMetrics(font);
    int widthInPixels = fontMetrics.width(comicStripAuthor);
    title.addText(QPointF(panelSpacing, 70), font,
comicStripTitle);
    title.addText(QPointF( comicStripWidth - (widthInPixels +
panelSpacing), 70), font, comicStripAuthor);
    p.setBrush(Qt::black);
    p.drawPath(title);

    p.setBrush(Qt::NoBrush);
    p.setPen(borders);
    QPoint startPoint(panelSpacing, topMargin);
    // Loop through panels, each time drawing it in an IMAGE,
then rendering that Image into the GENERAL IMAGE.
    for (int i = 0; i < panels.size(); ++i)
    {
        QGraphicsScene *toExport = panels.value(i);
        QImage current((int)toExport->width(), (int)toExport-
>height(), QImage::Format_ARGB32_Premultiplied);
        QPainter pintor(&current);
        pintor.setRenderHint(QPainter::Antialiasing);
        toExport->render(&pintor);

        // Draws current IMAGE to GENERAL IMAGE with
BORDER.
        p.drawImage(startPoint, current);
        p.drawRect((int)startPoint.x(),(int)startPoint.y(),
(int)toExport->width(),(int)toExport->height());

        // Computes position of next PANEL
        startPoint.setX((int)(startPoint.x() + toExport->width()
+ panelSpacing));
    }

    return image;
}

QString Stage::decodeStage()
{
    QString plainText;

    // Generate HEADER for the file
    plainText.append("### Created in Dibuho © 2006 -
2007 \n\n");

    plainText.append("<header>\n");
    plainText.append(tr("\tversion=%1\n").arg("1.0"));
    plainText.append(tr("\ttitle=%1\n").arg(comicStripTitle));
    plainText.append(tr("\tauthor=
%1\n").arg(comicStripAuthor));
    plainText.append(tr("\tpanels=%1\n").arg(panelCount));
    plainText.append(tr("\tspacing=
%1\n").arg(panelSpacing));
    plainText.append("</header>\n\n");

    // Write here the list of LIBS used
    plainText.append("<libs>\n");
    for(int i = 0; i < libs.size(); i++)
    {
        plainText.append("\t" + libs.value(i) + "\n");
    }
    plainText.append("</libs>\n\n");

    // Character Details that refer to the required library items.
State, orientation, position are not detailed here. It is in the
BODY.
    plainText.append("<chars>\n");
    plainText.append("</chars>\n\n");

    // Generate PANELS
    plainText.append("<panels>\n");

    for (int i = 0; i < panels.size(); ++i)
    {
        QGraphicsScene *currentPanel = panels.value(i);
        plainText.append(tr("\npanel=%1\n").arg(i+1));
        plainText.append("[\n");
        plainText.append(tr("\twidth=
%1\n").arg(panelW.value(i)));

        QList<QGraphicsItem *> itemsList = currentPanel-
>items();
        for (int j = 0; j < itemsList.size(); ++j)
        {
            QGraphicsItem *currentItem = itemsList.value(j);
            QMatrix saveMatrix = currentItem->matrix();

            // For BACKGROUND
            if(currentItem->data(1)=="background")
            {
                plainText.append("\tbackground\n");
                plainText.append("\t[\n");
                plainText.append(tr("\t\tlib=
%1\n").arg(currentItem->data(0).toString()));
                plainText.append(tr("\t\tname=
%1\n").arg(currentItem->data(2).toString()));
                plainText.append("\t]\n");
            }
            // For PROPS
            else if(currentItem->data(1)=="prop")
            {
                plainText.append("\tprops item\n");
                plainText.append("\t[\n");
                plainText.append(tr("\t\tlib=
%1\n").arg(currentItem->data(0).toString()));
                plainText.append(tr("\t\tname=
%1\n").arg(currentItem->data(2).toString()));
                plainText.append(tr("\t\tx=%1\n").arg(currentItem-
>scenePos().x()));
                plainText.append(tr("\t\ty=%1\n").arg(currentItem-
>scenePos().y()));
                plainText.append(tr("\t\tz=%1\n").arg(currentItem-
>zValue()));
                plainText.append(tr("\t\tmatrix=%1 %2 %3 %4 %5
%6\n").arg(saveMatrix.m11()).arg(saveMatrix.m12()).arg(sav
eMatrix.m21()).arg(saveMatrix.m22()).arg(saveMatrix.dx()).a
rg(saveMatrix.dy()));
                plainText.append("\t]\n");

            }

            // For BUBBLES
            else if(currentItem->data(1)=="bubble")
            {
                plainText.append("\tbubble\n");
                plainText.append("\t[\n");
                plainText.append(tr("\t\tcontent=
%1\n").arg(currentItem->data(2).toString()));
                plainText.append(tr("\t\ttype=
%1\n").arg(currentItem->data(3).toString()));
                plainText.append(tr("\t\tx=%1\n").arg(currentItem-
>scenePos().x()));
                plainText.append(tr("\t\ty=%1\n").arg(currentItem-
>scenePos().y()));
                plainText.append(tr("\t\tz=%1\n").arg(currentItem-
>zValue()));
                plainText.append(tr("\t\twidth=
%1\n").arg(currentItem->data(4).toString()));
                plainText.append(tr("\t\tdir=
%1\n").arg(currentItem->data(5).toString()));
                plainText.append("\t]\n");
            }

            // For CHARACTERS

            else if(currentItem->data(1)=="character")
```

```
        {
                QGraphicsItem *tempItem = currentItem->children().value(0);
                QMatrix saveMatrix = tempItem->matrix();


            plainText.append("\tcharacter\n");
            plainText.append("\t{\n");
            plainText.append(tr("\t\tname= %1\n").arg(currentItem->data(2).toString()));
            plainText.append(tr("\t\tgender= %1\n").arg(currentItem->data(3).toString()));
            plainText.append(tr("\t\texp= %1\n").arg(currentItem->data(4).toString()));
            plainText.append(tr("\t\tugar= %1\n").arg(currentItem->data(5).toString()));
            plainText.append(tr("\t\tskin= %1\n").arg(currentItem->data(6).toString()));
            plainText.append(tr("\t\tlgar= %1\n").arg(currentItem->data(7).toString()));

            plainText.append(tr("\t\tx=%1\n").arg(currentItem->children().value(0)->scenePos().x()));
            plainText.append(tr("\t\ty=%1\n").arg(currentItem->children().value(0)->scenePos().y()));

            plainText.append(tr("\t\tmatrix=%1 %2 %3 %4 %5 %6\n").arg(saveMatrix.m11()).arg(saveMatrix.m12()).arg(saveMatrix.m21()).arg(saveMatrix.m22()).arg(saveMatrix.dx()).arg(saveMatrix.dy()));

            foreach(QGraphicsItem *child, tempItem->children()){
                        QMatrix saveTrans = child->matrix();
                        if(child->data(1)=="neck"){
                        plainText.append(tr("\t\tneckmatrix=%1 %2 %3 %4 %5 %6\n").arg(saveTrans.m11()).arg(saveTrans.m12()).arg(saveTrans.m21()).arg(saveTrans.m22()).arg(saveTrans.dx()).arg(saveTrans.dy()));
                        }
                                                                if(child->data(1)=="hips"){
                        plainText.append(tr("\t\thipsmatrix=%1 %2 %3 %4 %5 %6\n").arg(saveTrans.m11()).arg(saveTrans.m12()).arg(saveTrans.m21()).arg(saveTrans.m22()).arg(saveTrans.dx()).arg(saveTrans.dy()));
                        }
            if(child->data(1)=="ularm"){
                        plainText.append(tr("\t\tularmmatrix= %1 %2 %3 %4 %5 %6\n").arg(saveTrans.m11()).arg(saveTrans.m12()).arg(saveTrans.m21()).arg(saveTrans.m22()).arg(saveTrans.dx()).arg(saveTrans.dy()));
                        }
            if(child->data(1)=="urarm"){
                        plainText.append(tr("\t\turarmmatrix= %1 %2 %3 %4 %5 %6\n").arg(saveTrans.m11()).arg(saveTrans.m12()).arg(saveTrans.m21()).arg(saveTrans.m22()).arg(saveTrans.dx()).arg(saveTrans.dy()));
                        }
            }



            plainText.append("\t]\n");
        }
    }
    plainText.append("]\n");
  }
  plainText.append("</panels>\n");

  return plainText;
}

void Stage::encodeStage(QTextStream &content)
{
  QString currentLine;
  while(!content.atEnd())
  {
    currentLine = content.readLine();
    if(!currentLine.startsWith("###"))
    {
      // Load HEADER
      if(currentLine.startsWith("<header>"))
      {
        while(!currentLine.startsWith("</header>"))
        {
          currentLine = content.readLine();

          if(currentLine.startsWith("\tversion"))
          {
          }
          else if(currentLine.startsWith("\ttitle"))
          {
            comicStripTitle = currentLine.section("=",1,1);
          }
          else if(currentLine.startsWith("\tauthor"))
          {
            comicStripAuthor = currentLine.section("=",1,1);
          }
          else if(currentLine.startsWith("\tpanels"))
          {
            panelCount = currentLine.section("=",1,1).toInt();
          }
          else if(currentLine.startsWith("\tspacing"))
          {
            panelSpacing = currentLine.section("=",1,1).toInt();
          }
        }
        panels.clear();
        panelW.clear();
        layerCounter.clear();
        libs.clear();
        libCounter.clear();
        currentSceneIndex = 0;
      }

      else if(currentLine.startsWith("<libs>"))
      {
        // check library compatibility code
      }

      else if(currentLine.startsWith("<chars>"))
      {
        // characters used
      }

      else if(currentLine.startsWith("<panels>"))
      {
        currentSceneIndex = 0;
        while(!currentLine.startsWith("</panels>"))
        {
          currentLine = content.readLine();
          if(currentLine.startsWith("panel"))
          {
            while(!currentLine.startsWith("]"))
            {
              currentLine = content.readLine();
              if(currentLine.startsWith("\twidth"))
              {
                qreal widthFromFile = currentLine.section("=",1,2).toDouble();
                QGraphicsScene *newPanel = new QGraphicsScene(this);
                newPanel->setSceneRect(0, 0, panelHeight * widthFromFile, panelHeight);
                panels.append(newPanel);
                panelW.append(widthFromFile);
                layerCounter.append(1);
                setScene(newPanel);
              }
```

```
                    // For BACKGROUND
                    if(currentLine.startsWith("\tbackground"))
                    {
                        QString lib;
                        QString name;
                        while(!currentLine.startsWith("\t]"))
                        {
                            currentLine = content.readLine();
                            if(currentLine.startsWith("\t\tlib"))
                                lib = currentLine.section("=",1,2);
                            if(currentLine.startsWith("\t\tname"))
                                name =
currentLine.section("=",1,2);
                        }
                        emit requestBackground(lib, name);
                    }

                    // For PROPS
                    if(currentLine.startsWith("\tprops item"))
                    {
                        QString libToRequest;
                        QString nameToRequest;
                        QString matrixValue;
                        qreal x = 0;
                        qreal y = 0;
                        qreal z = 0;
                        while(!currentLine.startsWith("\t]"))
                        {
                            currentLine = content.readLine();
                            if(currentLine.startsWith("\t\tlib"))
                                libToRequest =
currentLine.section("=",1,2);
                            if(currentLine.startsWith("\t\tname"))
                                nameToRequest =
currentLine.section("=",1,2);
                            if(currentLine.startsWith("\t\tx"))
                                x =
currentLine.section("=",1,2).toDouble();
                            if(currentLine.startsWith("\t\ty"))
                                y =
currentLine.section("=",1,2).toDouble();
                            if(currentLine.startsWith("\t\tz"))
                                z =
currentLine.section("=",1,2).toDouble();
                            if(currentLine.startsWith("\t\tmatrix"))
{
                                matrixValue =
currentLine.section("=",1,2);
                            }
                        }
                        emit requestItem(libToRequest,
nameToRequest, QMatrix((qreal)matrixValue.section("
",0,0).toDouble(),(qreal)matrixValue.section("
",1,1).toDouble(),(qreal)matrixValue.section("
",2,2).toDouble(),(qreal)matrixValue.section("
",3,3).toDouble(),(qreal)matrixValue.section("
",4,4).toDouble(),(qreal)matrixValue.section("
",5,5).toDouble()), QPointF(x,y), z);
                    }

                    // For BUBBLES
                    else if(currentLine.startsWith("\tbubble"))
                    {
                        QString bubbleContent;
                        QString type;
                        QString direction;
                        qreal x = 0;
                        qreal y = 0;
                        qreal z = 0;
                        qreal width = 0;
                        while(!currentLine.startsWith("\t]"))
                        {
                            currentLine = content.readLine();
                            if(currentLine.startsWith("\t\tcontent")
)
                                bubbleContent =
currentLine.section("=",1,2);
                            if(currentLine.startsWith("\t\ttype"))
                                type = currentLine.section("=",1,2);
                            if(currentLine.startsWith("\t\tx"))
                                x =
currentLine.section("=",1,2).toDouble();
                            if(currentLine.startsWith("\t\ty"))
                                y =
currentLine.section("=",1,2).toDouble();
                            if(currentLine.startsWith("\t\tz"))
                                z =
currentLine.section("=",1,2).toDouble();
                            if(currentLine.startsWith("\t\twidth"))
                                width =
currentLine.section("=",1,2).toDouble();
                            if(currentLine.startsWith("\t\tdir"))
                                direction =
currentLine.section("=",1,2);
                        }

                        Bubble *temp = new
Bubble(bubbleContent, type, direction, width, 0);
                        temp->setData(1, "bubble");
                        temp->setData(2, bubbleContent);
                        temp->setData(3, type);
                        temp->setData(4, width);
                        temp->setData(5, direction);
                        temp->setZValue(z);
                        temp->setPos(QPointF(x,y));
                        acceptItem(temp);
                    }

                    // For CHARACTERS
                    else
if(currentLine.startsWith("\tcharacter"))
                    {
                        QString name;
                        QString gender;
                        QString exp;
                        qreal x = 0;
                        qreal y = 0;
                        QString ugar;
                        QString skin;
                        QString lgar;
                        QString matrixValue;
                        while(!currentLine.startsWith("\t]"))
                        {
                            currentLine = content.readLine();
                            if(currentLine.startsWith("\t\tname"))
                                name =
currentLine.section("=",1,2);
                            if(currentLine.startsWith("\t\tgender"))
                                gender =
currentLine.section("=",1,2);
                            if(currentLine.startsWith("\t\texp"))
                                exp = currentLine.section("=",1,2);
                            if(currentLine.startsWith("\t\tugar"))
                                ugar =
currentLine.section("=",1,2);
                            if(currentLine.startsWith("\t\tskin"))
                                skin = currentLine.section("=",1,2);
                            if(currentLine.startsWith("\t\tlgar"))
                                lgar = currentLine.section("=",1,2);
                            if(currentLine.startsWith("\t\tx"))
                                x =
currentLine.section("=",1,2).toDouble();
                            if(currentLine.startsWith("\t\ty"))
                                y =
currentLine.section("=",1,2).toDouble();
                            if(currentLine.startsWith("\t\tmatrix"))
                                matrixValue =
currentLine.section("=",1,2);
                        }

                        Character *temp = new
Character(name, gender, exp, QColor(ugar), QColor(skin),
QColor(lgar));
                        temp->children().value(0)-
>setPos(QPointF(x,y));
                        temp->children().value(0)-
>setMatrix(QMatrix((qreal)matrixValue.section("
```

```
",0,0).toDouble(),(qreal)matrixValue.section("
",1,1).toDouble(),(qreal)matrixValue.section("
",2,2).toDouble(),(qreal)matrixValue.section("
",3,3).toDouble(),(qreal)matrixValue.section("
",4,4).toDouble(),(qreal)matrixValue.section("
",5,5).toDouble()));
                        acceptItem(temp);
                    }
                }
                currentSceneIndex++;
            }
        }
    }
}

currentScene       = panels.value(0);
currentSceneIndex  = 0;
modified           = false;
setScene(currentScene);
scene()->update();
}

bool Stage::isModified()
{
   return modified;
}

void Stage::setModified(bool mod)
{
   modified = mod;
}


void Stage::keyPressEvent(QKeyEvent *event)
{
   QGraphicsItem *currentFocus;
   switch (event->key()) {
      case Qt::Key_Plus:
         if(currentTool=="rotateTool")
         {
            currentFocus = scene()->focusItem();
            if(currentFocus)
            {
               matrix = currentFocus->matrix();
               matrix.rotate(10);
               currentFocus->setMatrix(matrix);
               modified = true;
            }
         }
         else if (currentTool=="scaleTool")
         {
            currentFocus = scene()->focusItem();
             if(currentFocus)
             {
               matrix = currentFocus->matrix();
               matrix.scale(1.1,1.1);
               currentFocus->setMatrix(matrix);
               modified = true;
             }
         }
         else
         {
            if(scaleValue < 10)
               scaleValue += 0.1;
         }
         break;
      case Qt::Key_Minus:
         if(currentTool=="rotateTool")
         {
            currentFocus = scene()->focusItem();
            if(currentFocus)
            {
               matrix = currentFocus->matrix();
               matrix.rotate(-10);
               currentFocus->setMatrix(matrix);
               modified = true;
            }
         }
         else if (currentTool=="scaleTool")
         {
            currentFocus = scene()->focusItem();
            if(currentFocus)
            {
               matrix = currentFocus->matrix();
               matrix.scale(0.9,0.9);
               currentFocus->setMatrix(matrix);
               modified = true;
            }
         }
         else
         {
            if(scaleValue > 0.2)
               scaleValue -= 0.1;
         }
         break;
      case Qt::Key_W:       //  Shear
         currentFocus = scene()->focusItem();
         if(currentFocus){
            matrix = currentFocus->matrix();
            matrix.shear(.1,.1);
            currentFocus->setMatrix(matrix);
            modified = true;
         }
         break;
      case Qt::Key_Q:       //  Shear
         currentFocus = scene()->focusItem();
         if(currentFocus){
            matrix = currentFocus->matrix();
            matrix.shear(-0.1,-0.1);
            currentFocus->setMatrix(matrix);
            modified = true;
         }
         break;
      case Qt::Key_D:       //  Scale Down
         currentFocus = scene()->focusItem();
         if(currentFocus){
            matrix = currentFocus->matrix();
            matrix.scale(0.9, 0.9);
            currentFocus->setMatrix(matrix);
            modified = true;
         }
         break;
      case Qt::Key_Delete:       //  Delete
         currentFocus = scene()->focusItem();
         if(currentFocus){
            QString library = currentFocus->data(0).toString();
            int index = libs.indexOf(library);
            if(libCounter.value(index) > 1){
               libCounter.replace(index,
libCounter.value(index)-1);
            } else{
               libCounter.removeAt(index);
               libs.removeAt(index);
            }
            if(currentFocus->data(0)=="torso")
               currentFocus->parentItem()-
>~QGraphicsItem();
            else
               currentFocus->~QGraphicsItem();
            modified = true;
         }
         break;
      case Qt::Key_PageUp:
         getPrevScene();
         break;
      case Qt::Key_PageDown:
         getNextScene();
         break;
      default:
         QGraphicsView::keyPressEvent(event);
      }
      setupMatrix();
}

void Stage::setupMatrix()
{
   QMatrix matrix;
```

```
    matrix.scale(scaleValue, scaleValue);
    setMatrix(matrix);
}

void Stage::getTool(QString tool)
{
    currentTool = tool;
}

void Stage::drawBackground(QPainter *painter, const QRectF
&rect)
{
    Q_UNUSED(rect);

    // Shadow
    QRectF sceneRect = this->sceneRect();
    QRectF rightShadow(sceneRect.right(), sceneRect.top() +
5, 5, sceneRect.height());
    QRectF bottomShadow(sceneRect.left() + 5,
sceneRect.bottom(), sceneRect.width(), 5);
    if (rightShadow.intersects(rect) ||
rightShadow.contains(rect))
        painter->fillRect(rightShadow, Qt::darkGray);
    if (bottomShadow.intersects(rect) ||
bottomShadow.contains(rect))
        painter->fillRect(bottomShadow, Qt::darkGray);

    // Fill
    QLinearGradient gradient(sceneRect.topLeft(),
sceneRect.bottomRight());
    gradient.setColorAt(0, Qt::white);
    gradient.setColorAt(1, Qt::lightGray);
    painter->fillRect(rect.intersect(sceneRect), gradient);
    painter->setBrush(Qt::NoBrush);
    painter->drawRect(sceneRect);
}

void Stage::drawForeground(QPainter *painter, const QRectF
&rect)
{
    Q_UNUSED(rect);

    // Fill
    foreach(QGraphicsItem *selected, scene()-
>selectedItems())
    {
            QRectF rect = selected->boundingRect();

            QRectF selectBox(0, 0, rect.width(), rect.height());
            painter->setMatrix(selected->sceneMatrix(), true);
            painter->setPen(QPen(Qt::blue, .5, Qt::DashLine));
            painter->drawRect(selectBox);
            }
}

void Stage::getNextScene()
{
    if(currentSceneIndex < panels.count()-1)
    {
      currentSceneIndex++;
      currentScene = panels.value(currentSceneIndex);
      setScene(currentScene);
      emit
valueChanged(tr("%1").arg(currentSceneIndex+1));
    }
    scene()->update(rect());
}

void Stage::getPrevScene()
{
    if(currentSceneIndex > 0)
    {
      currentSceneIndex--;
      currentScene = panels.value(currentSceneIndex);
      setScene(currentScene);
      emit
valueChanged(tr("%1").arg(currentSceneIndex+1));
    }
    scene()->update(rect());
}

void Stage::insertPanel()
{
    QGraphicsScene *bago = new QGraphicsScene();
    bago->setSceneRect(0, 0, panelHeight * 1.25,
panelHeight);
    panelW.append(1.25);
    panels.append(bago);
    layerCounter.append(1);
    panelCount++;

    currentSceneIndex = panels.size() - 1;
    currentScene = panels.value(currentSceneIndex);
    emit valueChanged(tr("%1").arg(currentSceneIndex+1));
    modified = true;

    setScene(currentScene);
    scene()->update(rect());
}

void Stage::deletePanel()
{
    if(panels.size()>1)
    {
        QList<QGraphicsItem *> itemsList = scene()->items();
        foreach(QGraphicsItem *item, itemsList)
        {
            QString library = item->data(0).toString();
            int index = libs.indexOf(library);
            if(libCounter.value(index) > 1){
                libCounter.replace(index,
libCounter.value(index)-1);
            } else{
                libCounter.removeAt(index);
                libs.removeAt(index);
            }
        }

        panels.removeAt(currentSceneIndex);
        panelW.removeAt(currentSceneIndex);
        layerCounter.removeAt(currentSceneIndex);
        if(currentSceneIndex!=0){
          currentSceneIndex--;
        }
        panelCount--;

        currentScene = panels.value(currentSceneIndex);
        setScene(currentScene);
        scene()->update(rect());
        modified = true;
        emit
valueChanged(tr("%1").arg(currentSceneIndex+1));
    }
    else
    {
            QMessageBox::about(this, tr("Delete Panel
Error"), tr("You cannot delete this panel.\nThere is only one
panel remaining.\nThe comic strip must have at least one
panel."));
    }
}

void Stage::clearBackground()
{
    QList<QGraphicsItem *> itemsList = scene()->items();

    foreach(QGraphicsItem *item, itemsList)
    {
        if(item->zValue()==0)
            item->~QGraphicsItem();
    }

    scene()->update();
    modified = true;
}

void Stage::optionsPanel()
```

```
{
   DialogOptions *create = new DialogOptions(this);
   create->panelWidthMult-
>setValue(panelW.value(currentSceneIndex));
   create->borderWidth->setValue(borders.width());

   if(create->exec())
   {
      panelW[currentSceneIndex] = create->panelWidthMult-
>value();
      borders.setWidth(create->borderWidth->value());
      scene()->setSceneRect(0, 0, panelHeight *
panelW[currentSceneIndex], panelHeight);
   }

   // Resize the BACKGROUND to fit new size of PANEL
   QList<QGraphicsItem *> itemsList = scene()->items();
   foreach(QGraphicsItem *item, itemsList)
   {
      if(item->zValue()==0)
      {
         item->resetMatrix();
         qreal width  = item->boundingRect().width();
         qreal height = item->boundingRect().height();
         qreal wratio = (panelHeight *
panelW.value(currentSceneIndex)) / width;
         qreal hratio = panelHeight / height;

         QMatrix bgMatrix;
         bgMatrix.scale(wratio, hratio);
         item->setMatrix(bgMatrix);
      }
   }
   scene()->update();
}
```

toolbox.cpp

```
#include "ui_toolbox.h"
#include "toolbox.h"

ToolBox::ToolBox(QWidget *parent)
   : QWidget(parent)
{
   setupUi(this);

   selectTool->setCheckable(true);
   scaleTool->setCheckable(true);
   rotateTool->setCheckable(true);

   QButtonGroup *affine = new QButtonGroup(this);
   affine->setExclusive(true);
   affine->addButton(scaleTool);
   affine->addButton(rotateTool);
   affine->addButton(selectTool);

   connect(scaleTool, SIGNAL(clicked()), this,
SLOT(setScaleTool()));
   connect(rotateTool, SIGNAL(clicked()), this,
SLOT(setRotateTool()));
   connect(selectTool, SIGNAL(clicked()), this,
SLOT(setSelectTool()));
}

void ToolBox::setScaleTool()
{
   currentTool = "scaleTool";
   emit changeTool(currentTool);
}

void ToolBox::setRotateTool()
{
   currentTool = "rotateTool";
   emit changeTool(currentTool);
}

void ToolBox::setSelectTool()
{
   currentTool = "selectTool";
   emit changeTool(currentTool);
}
```

BasicLib Plugin

basiclib..h

```
#ifndef BASICLIB_H
#define BASICLIB_H

#include <QGraphicsItem>
#include <QGraphicsSvgItem>
#include <QObject>
#include <QStringList>

#include <dibuho/elements.h>

class BasicLib : public QObject,
                 public PropElement,
                 public BackgroundElement,
                 public BodyElement
{
    Q_OBJECT
    Q_INTERFACES(PropElement)
    Q_INTERFACES(BackgroundElement)
    Q_INTERFACES(BodyElement)

public:
   QStringList props() const;
   QStringList backgrounds() const;
   QStringList eyes() const;
   QStringList ears() const;
   QStringList nose() const;
   QStringList lips() const;
   QStringList hair() const;
   QGraphicsItem *insertElement(const QString &name);
   QGraphicsItem *queryBackground(const QString &name);
   QGraphicsItem *getPart(const QString &name, const
QString &orientation);

};

#endif
```

basiclib..cpp

```
#include <QtGui>
#include <QtSvg>
#include <math.h>
#include <stdlib.h>

#include "basiclib.h"

QStringList BasicLib::props() const
{
   return QStringList()    << tr("Cd") << tr("Box") <<
tr("Clock")
                           << tr("Cards") << tr("Salbabida");
}

QStringList BasicLib::backgrounds() const
{
   return QStringList() << tr("Dirt") << tr("House");
}

QStringList BasicLib::eyes() const
{
   return QStringList() << tr("Simple") << tr("Scary") <<
tr("Round") << tr("Eyelashes");
}

QStringList BasicLib::hair() const
{
   return QStringList() << tr("Straight") << tr("Thin");
}

QStringList BasicLib::nose() const
```

```
{
    return QStringList() << tr("Small") << tr("Big");
}

QStringList BasicLib::lips() const
{
    return QStringList() << tr("Pale") << tr("Red");
}

QStringList BasicLib::ears() const
{
    return QStringList() << tr("Normal");
}

QGraphicsItem *BasicLib::getPart(const QString &name,
const QString &orientation)
{
    Q_INIT_RESOURCE(svg);
    QGraphicsItem *item = new QGraphicsSvgItem;
    if (name == tr("Simple"))
    {
        item = new QGraphicsSvgItem(":/bodyparts/eyes-
simple-front.svg");
        item->setPos(0,35);
        item->setData(2,"Simple");
    }
    else if (name == tr("Scary"))
    {
        item = new QGraphicsSvgItem(":/bodyparts/eyes-scary-
front.svg");
        item->setPos(0,35);
        item->setData(2,"Scary");
    }
    else if (name == tr("Round"))
    {
        item = new QGraphicsSvgItem(":/bodyparts/eyes-
round-front.svg");
        item->setPos(0,35);
        item->setData(2,"Round");
    }

    else if (name == tr("Eyelashes"))
    {
        item = new QGraphicsSvgItem(":/bodyparts/eyes-
eyelashes.svg");
        item->setPos(0,35);
        item->setData(2,"Eyelashes");
    }

    else if (name == tr("Straight"))
    {
        item = new QGraphicsSvgItem(":/bodyparts/hair-
simple-front.svg");
        item->setPos(0,0);
        item->setData(2,"Straight");
    }
    else if (name == tr("Thin"))
    {
        item = new QGraphicsSvgItem(":/bodyparts/hair-
gone.svg");
        item->setPos(0,0);
        item->setData(2,"Thin");
    }

    item->setFlags(QGraphicsItem::ItemIsMovable |
QGraphicsItem::ItemIsSelectable |
QGraphicsItem::ItemIsFocusable);
    item->setData(0,"BasicLib");
    item->setData(1,"facial part");
    return item;
}

QGraphicsItem *BasicLib::insertElement(const QString
&name)
{
    QGraphicsItem *item = new QGraphicsSvgItem;

    if (name == tr("Cd"))
    {
        item = new QGraphicsSvgItem(":/props/cd.svg");
        item->setData(2,"Cd");
    }
    else if (name == tr("Box"))
    {
        item = new QGraphicsSvgItem(":/props/box.svg");
        item->setData(2,"Box");
    }
    else if (name == tr("Clock"))
    {
        item = new QGraphicsSvgItem(":/props/clock.svg");
        item->setData(2,"Clock");
    }
    else if (name == tr("Salbabida"))
    {
        item = new QGraphicsSvgItem(":/props/salbabida.svg");
        item->setData(2,"Salbabida");
    }
    else if (name == tr("Cards"))
    {
        item = new QGraphicsSvgItem(":/props/cards.svg");
        item->setData(2,"Cards");
    }

    item->setFlags(QGraphicsItem::ItemIsMovable |
QGraphicsItem::ItemIsSelectable |
QGraphicsItem::ItemIsFocusable);
    item->setData(0,"BasicLib");
    item->setData(1,"prop");
    return item;
}

QGraphicsItem *BasicLib::queryBackground(const QString
&name)
{
    QGraphicsItem *item = new QGraphicsSvgItem;
    Q_INIT_RESOURCE(svg);
    if (name == tr("Dirt"))
    {
        item = new
QGraphicsSvgItem(":/backgrounds/dirt.svg");
        item->setData(2,"Dirt");
    }
    else if (name == tr("House"))
    {
        item = new
QGraphicsSvgItem(":/backgrounds/house.svg");
        item->setData(2,"House");
    }

    item->setData(0,"BasicLib");
    item->setData(1,"background");
    return item;
}

Q_EXPORT_PLUGIN2(dibuho_basiclibrary, BasicLib)
```

XI. Acknowledgement

This Special Problem would not have been unraveled with success if not for the encouragement, support, and inspiration I derived from other people. Some of them deserves mentioning. Credit goes to the following people:

My adviser, Sir Baes, who, fortunately, still remembers me and my thesis topic inspite of my year-long silence and disappearance. Thanks for pushing my defense to take place this 2nd semester.

My professors, especially Ma'm Carpio, whose encouragement and prayers counted a lot; Doc Magboo, for allowing my defense to proceed despite the hectic schedule; Ma'm Rubio, Ma'm Bastero, Ma'm Nangan, Ma'm Magboo, Sir Chua, Sir Co, Sir De Armas, and all my other professors for – well, teaching me. I will always look back in gratitude. Credit also goes to Ate Shula and Ate Eden for always being there, be it in DPSM or in the computer rooms, always ready to lend a helping hand to confused students.

The trolls at Trolltech, mainly for having Qt Open Source so well documented that I opted to abandon my already long Gtk+ codebase and start from scratch. I do not regret the decision, and I do think I will be developing quite a few more Qt-based programs in the future. Special credit is also given to the people in the Qt-dev mailing list and the countless members of the Open Souce community, whose names I may have failed to remember, but not their deeds for never failing to lend support and offer advice to programmers who are at lost, as I was, when starting their project.

My friends and blockmates, for keeping me company during my stay in UP and for making a seemingly tedious course like Computer Science a remarkably enjoyable one. Special credit goes to Regina, for the frequent nagging and reminder that I finish my thesis, and for lending me her thesis documentation from which I based the format of mine; to Nuni, for promptly giving light to all the questions I have; to Luwis, Lianne, and Bily for the well-needed goodlucks; to Mitchie for trying and successfully making me feel better when I thought all is lost with my defense.

My Brods and Sisses from the Pi Sigma Fraternity and Pi Sigma Delta Sorority, for the understanding, concern, and support on my thesis.

The Becquerel people, notably Philip for sending a link to Trolltech's website, which unexpectedly became the turning point of my special problem; Andrea for offering to produce drawings for my program and for imparting some knowledge with comic strips (as I know these things are her specialty) which are especially useful in my related literature.

My brother Vincent for funding my unemployed days; his girlfriend Lisa, for lending me her laptop for my consultations; my sister Tess, also for lending me her old laptop, which fortunately was able to run my program despite being primitive; and the rest of my family, who are too many to mention here – what with the exponential growth of the number of my nieces and nephews – but still deserve more than what this credit could offer.

And most importantly, Ma. Feliza Nolasco, to whom this project is dedicated to, for making entirely difficult and unbearable problems, such as a Special Problem, effortless and meaningful.

Again, to all those mentioned, I am exceedingly thankful.