UNIVERSIDADE D
COIMBRA

Ana Catarina Miranda Almeida

Development of a platform for generation of icon-based graphic codes

Dissertation supervised by Professor Nuno Gonçalves and submitted to the Electrical and Computer Engineering Department of the Faculty of Science and Technology of the University of Coimbra, in partial fulfillment of the requirements for the Degree of Master in Electrical and Computer Engineering, branch of Computers.

Supervisor:

Prof. Nuno Miguel Mendonça da Silva Gonçalves, PhD

Jury:

Prof. Vítor Manuel Mendes da Silva, PhD

Prof. Paulo Jorge Carvalho Menezes, PhD

Prof. Nuno Miguel Mendonça da Silva Gonçalves, PhD

Coimbra, July of 2020

Acknowledgments

Firstly, I would like to thank my supervisor, Professor Nuno Gonçalves, for his guidance and advice throughout the development of this dissertation, for giving me the opportunity to learn and work in such an interesting area as information coding and for the time devoted to completing and solidifying my academic degree.

I would also like to thank my laboratory colleagues for the availability and help provided for the success of this project.

Finally, I would like to express a special thanks to my family and friends for the moral support, without which this dissertation would not have been as successfully completed.

Abstract

Machine-Readable Codes (MRC) have been used for several purposes over the years and classic approaches like the bar code or the QR Code can be seen everywhere in our day-to-day. However, more recently, a new MRC was created with the ability to combine the communication power of classical methods with a meaningful improvement on aesthetics and data capacity. This method is named UniQode.

The UniQode has two major advantages over classical MRCs: aesthetics and larger coding capacity. Consequently, it opens new possibilities for several purposes such as product identification, tracking, marketing and seller-buyer communication. This new MRC method is composed by several elements, one of them being the Graphic Code (GC). This element can be created in two different ways, either with pixels or with icons, or, sometimes, both.

The pixel-based GC has been extensively researched and its accomplishments will be presented in this document. However, the icon-based GC still represents a new concept with a mostly unstructured creation method.

In this dissertation, the proposed work encompasses not only a formalisation of the generation method for icon-based GC, but also the creation of a tool for this type of encoding process, in order to quicken this operation while also generating its formal documentation.

With the achievement of the proposed work, the industrialisation of this type of code is made possible and facilitated, and future work on this area is widened. Such work could comprise the development of a corresponding decoding tool, the enhancement of this project's tools abilities, among other aspects.

$Keywords$: Information Coding, Graphic Code, Machine Readable Code, Aesthetic Coding, Cryptography.

Resumo

Códigos Lidos por Máquinas (Machine-Readable Codes - MRC) têm sido usados para diversas finalidades ao longo dos anos, e abordagens clássicas como o código de barras ou o código QR podem ser vistas frequentemente no nosso dia-a-dia. Contudo, mais recentemente, um novo MRC foi criado com a capacidade de juntar o poder de comunicação dos métodos clássicos com uma melhoria significativa na estética e na capacidade de codificação. Este método é designado por UniQode.

O UniQode possui duas grandes vantagens sobre os MRCs clássicos: estética e maior capacidade de codificação. Consequentemente, abre novas possibilidades para diversas finalidades, como identificação de produtos, rastreamento, marketing e comunicação vendedor-comprador. Esse novo método MRC é composto por vários elementos, sendo um deles o Código Gráfico (Graphic Code - GC). Este elemento pode ser criado de duas formas diferentes, com pixéis ou com ícones, ou por vezes com ambos. O GC baseado em pixéis foi amplamente estudado e os seus resultados serão apresentados neste documento. No entanto, o GC baseado em ícones ainda representa um novo conceito, com um método de criação não estruturado.

Nesta dissertação, o trabalho proposto abrange não só a formalização do método de geração de GC baseado em ícones, mas também a criação de uma ferramenta para este processo de codificação, de modo de agilizar esta operação e gerar a respetiva documentação formal.

Com a realização do trabalho proposto, a industrialização deste tipo de código torna-se possível e é facilitada, e o trabalho futuro nesta área é ampliado. Esse trabalho engloba, por exemplo, o desenvolvimento de uma ferramenta de decodificação correspondente, o aprimoramento das funções deste projeto, entre outros aspetos.

$Palavras - chave$: Codificação de informação, Código gráfico, Códigos lidos por máquina, Codificação estética, Criptografia.

Acronyms

GC Graphic Code.

GU Graphic Unit.

GUI Graphic User Interface.

IDE Integrated Development Environment.

INCM Imprensa Nacional-Casa da Moeda.

ISR Instituto de Sistemas e Robótica da Universidade de Coimbra.

LSB Least Significant Bit.

MRC Machine-Readable Codes.

MVP Minimum Viable Product.

PEQRC Picture-Embedded QR Codes.

UML Unified Modelling Language.

VI Visual Identity.

List of Figures

Contents

1

Introduction

1.1 Context

Generally speaking, similarly to the constant evolution of the humankind, the technological development has followed several different paths, some of them exist to improve society, and others to hinder it, and, in the case of this dissertation, for each mechanism created for tax-evasion and product forgery, there needs to be a way to counteract it.

Consequently, to create a way to accomplish these goals, the Imprensa Nacional-Casa da Moeda (INCM), and the University of Coimbra together, carried out a research and development project so as to find new ways to avoid counterfeiting, and the consequent tax-evasion, and to ensure not only product authenticity, but also consumer protection.

Everyday new products are created and branded, money is spent on its marketing and advertisement and, to make sure the companies work is paid off, new technologies are needed to enable product verification and authentication, and always with the goal of improving vendor-buyer communication. These technologies need to be cheap and easy to implement, very secure, and, if possible, with the added benefit of improving the product's aesthetic value, and not hinder it.

With the goal of improving the said technology that allows product tracking, facilitates logistic processes and hinders its forgery creation, Machine-Readable Codes (MRC) were created more than 60 years ago. These types of stenographic codes allow each product to have it's own unique identifier, which in turn facilitates traceability by communicating confidential textual information through an image [21].

As experience shows, one of the most commonly used MRCs is the 1D bar-code (figure 1.1a). This MRC is very simple and quick to encode and decode, and very useful for short messages,

which explains its wide reach. As a consequence of its popularity, however, it became necessary to strictly standardise their creation so as to avoid a multitude of conflicts, such as having the same code identifying two different products. Therefore, this problem was then solved by the foundation of the GS1 organisation[1] with the main purpose of creating a coherence between these types of codes.

Later in time, however, when it became necessary to encode slightly longer messages, the codes ventured into the 2D, and options such as the QR Code (figure 1.1b) were created [26]. A comparison between the bar-code and QR Code sizes is shown in figure 1.2, where a QR Code can hold the same amount of data contained in a 1D bar-code in only one-tenth the size [26].

(a) 1D Bar Code [24].

(b) 2D QR Code [26].

Figure 1.1: Machine-Readable Codes.

Figure 1.2: Size comparison between 1D bar-code and QR Code [26].

Whereas these codes were considerable innovations for the time, they still had a drawback in common, and that was that none of them were visually appealing, with only black and white bars or pixels.

As a result, some techniques to improve code appearance were developed, such as colouring some of the code pixels or embedding some pixels of a picture in the actual code. These new options were, in fact, an improvement, but they were still not very visually pleasant, that is, they did not have a good enough aesthetic component, since examples like the QR Code still had to have quite a large area of mandatory elements (3 big squares and 1 smaller one, as show in figure

[1] Global Non-Profit Organisation that defines standards for bar-codes and RFID tags, so that there is a general consensus on the industry for these codes generation [24].

1.3). On one hand, these elements are important for a quick detection and decoding but, on the other hand, they make the code visually unappealing and inadequate whenever the code's aesthetic aspect is an important factor in the product it is applied on.

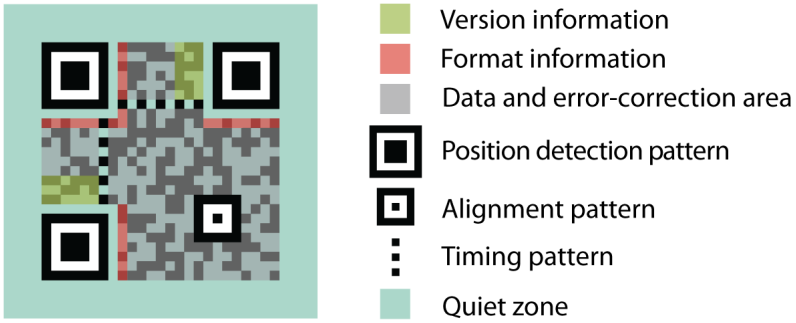


Figure 1.3: QR Code Modules [26].

This challenge to improve appearance and data capacity was taken by Instituto de Sistemas e Robótica da Universidade de Coimbra (ISR) investigators, together with the INCM, and then, as a consequence of a research and innovation project, the UniQode® [2] was created.

This innovative new way to encode messages joins several elements (hologram, Graphic Code, glitter printing) to make up a final complex and visually appealing code (since it does not resemble a typical code) that is also very hard to forge [16] (figure 1.4). Along with this new code, the communication with consumers was also improved by providing them a way to, by themselves through a phone app, check if any product, that the code is implemented on, is authentic or a forgery [16].

(a) Validation stamp of Portuguese tobacco (The grey strip is the hologram part of the UniQode).

(b) Graphic Code.

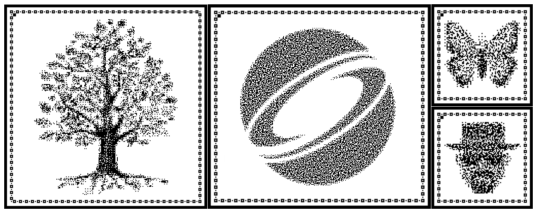
(c) Graphic Code using Pixels.

(d) Graphic Code using Icons.

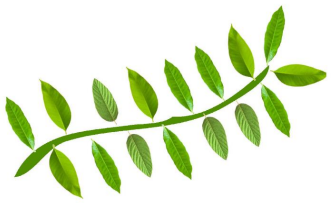
Figure 1.4: Validation stamp of Portuguese tobacco - composing elements of the UniQode [16].

[2] A patent for this product was applied for.

Herewith, the methodology developed for the Graphic Code element of UniQode allowed for it to be applied both with pixels and with icons (figure 1.5). This enabled a way to a greater exploration of UniQode's aesthetic strand.



(a) Pixel-based Graphic Code [15].



(b) Icon-based Graphic Code.

(c) Icon and Pixel-based Graphic Code [17].

Figure 1.5: Types of Graphic Codes.

The main difference between pixel-based and icon-based GC is that the first one is composed by only contrasting pixels (black and white in the case of figure 1.5a) and encoded with a halftone method (further explained in section 2.1), where the pixels are normally distributed in an orthonormal structure (mainly matrix-type structures); and the second one is composed solely by icons and the encoding method begins by associating a sequence of icons to certain information in a much more flexible structure, since the icons do not have to respect a orthonormal structure, as we can see in the example in figure 1.5b. Furthermore, there can also exist codes that incorporate both elements, as we can see in figure 1.5c.

1.2 Motivation

As we saw previously, the UniQode has several composing elements and, with the possibility of each of these being widely variable, there are infinite different codes, in practical terms, that could be created with this method, which, in turn, deems it as a very time-costly and abstract process.

With creation processes like the referred above, the industrialisation of this type of codes is almost impossible to accomplish without setting a standard or exercising some control when it comes to the design options (just like the GS1 organisation standards were needed to make the generation of bar code, QR Code, data matrix, among others, coherent everywhere). In other words, if there are infinite possibilities of the outcome, the industrialisation of the code for practical use becomes almost impossible and, therefore, the development of such a complex code becomes essentially meaningless.

As a result, the need to give more structure and guidelines to the generation of each code, and keeping variables such as the printing process and the codes physical size in mind, originated a need to develop an architecture that can join all of the above in a single place for an easier, quicker and more organised way of design. Consequently, that revolves around an user-interaction platform, with the purpose of helping guide and focus the creation process itself.

Furthermore, this work has a big component on the formalisation of the amounts of information that fit in each visual identity that can be widely explored. Above all else, the icon-based GC has a very high aesthetic component that has yet to be explored, which makes it an appropriate UniQode component to focus on, in particular.

1.3 Goals

Although current UniQode codes and other MRCs have been showed to respond to general needs, the icon-based strand, which is a solution to most aesthetic needs, hasn't yet been profoundly explored (figure 1.6 shows an example of a recent prototype developed by the INCM). In addition to that, it is necessary to have more control over the developed output codes, so as to formalise their creation and enable an easier industrialisation process.

Therefore, from the information stated previously, the goal of this dissertation is to design a well lined framework, with well defined inputs, calculations and outputs, in order to allow the development of a standalone application to guide the creation and generation of the codes themselves, as well as to help create more aesthetically pleasing codes that, not only can encode more information, but also create a communication bridge between the producers and the consumers to help counteract product forgery, while exploring the marketing and communication potential of such codes.

Consequently, the above includes the development of the creation modules, and the definition

of all the possibilities accepted for the generation tool of icon-based UniQode.

Figure 1.6: INCM's first example of a possible icon-based Graphic Code in the UniQode setting.

1.4 Implementations and key contributions

The following implementations and contributions have been achieved:

- Uniformity and specification of icon-based GC elements (graphic unit, cell, alphabet and dictionary) in both quantities and types;
- Development of a standalone program for the icon-based Graphic Code creation, by implementing the above formalised standards and specifications in a C++ programming language program that receives a set of characters and images and generates a graphic code, a configuration file and a dictionary file containing the final code's information.
- Creation of a Graphical User Interface (GUI) for the program;
- Documentation of the whole program using Doxygen standards (chapter 6).

1.5 Structure of the dissertation

The course flow of this dissertation proceeds, in chapter two, with a general explanation of Machine-Readable Codes and some of the concepts behind it, added also to a presentation and discussion of some examples to illustrate the concept. Afterwards, a brief review of existing code

security and validation methods will be made. Then, chapter three describes the process of the program creation, from the used tools to the developed algorithms and implementation process. And finally, chapter four presents the analysis and results, including a few examples of generated codes using the developed tool, while final conclusions and future work propositions are made in chapter five.

2

State of The Art

This chapter is divided into two sections and reviews the state of the art that supports the developed work in this dissertation.

Firstly, this state of the art will begin with a brief description of stenography and its development into Machine-Readable Codes. Then, it will present different types of MRCs and how they generally operate, followed by a brief comparison between them and, lastly, some coding and decoding notions. In addition to that, code security will be explained, with additional focus in validation methods.

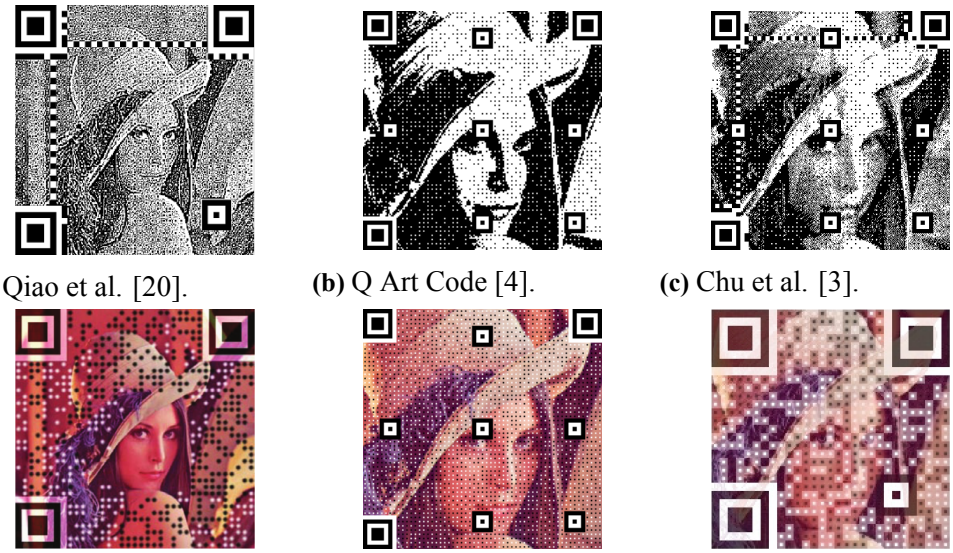
2.1 Machine-Readable Codes

Since the creation of cryptography (encoding of a message into something that is unreadable to the average person) with the purpose to keep the integrity of data intact, several other methods for concealing messages have been developed, one of them being stenography, which represents the concealment of information in appropriate multimedia carriers, like images or videos [21].

Consequently, MRCs are part of a stenographic approach to coding where the output is an image that goes from the simple 1D bar-code, to the more complex 2D codes that arose from the need to encode more information on a single carrier. What makes the codes above machine-readable ones is the fact that they can be decoded using some kind of machine-vision systems consisting of optical laser scanners or cameras and an interpreting software [26]. The use of scanning tools, instead of manually checking the codes, speeds data collection and eliminates manual data collection errors like illegible handwriting and data entry errors [24].

2.1.1 Types of MRCs

As the use of MRCs has been growing exponentially since its creation, the need to enhance its appearance and increase the size of the encoded information grew with it, and various types of 2D codes variations have been developed. As a consequence, a few Picture-Embedded QR Codes (PEQRC) were created to enhance the code appearance [3] [4] [5] [9] [10] (figure 2.1).



(a) Qiao et al. [20]. **(b)** Q Art Code [4]. **(c)** Chu et al. [3].

(d) Li et al. [9]. **(e)** Q Art Code [4]. **(f)** VisualLead [2].

Figure 2.1: Several PEQRC examples.

Initially, several approaches emerged where the halftone method was used. In this case, Halftoning is a method inspired by the dispersed-dot dithering approach [23] where it uses a highly contrasting pair of colours and takes each pixel in a greyscale image and transforms it into a $k * k$ set of the 2 contrasting colour pixels, whilst preserving the human perception of the original greyscale image.

Specifically, the dispersed-dot dithering approach consists in converting a greyscale image to a black and white one by dividing the 0-255 greyscale values into a set of quantized levels (quantum) and taking each greyscale pixel of the image and associating it to a dithering pattern. This dithering pattern is a $k * k$ matrix of black and white pixels, in which the number of black and white pixels depend on the quantized level that the greyscale pixel value, from the original image, fits into (figure 2.2) [23].

One of the first approaches that used the halftone method was known as Halftone Visual Cryptography [27] and could encode an image into two complementary others, where one has black pixels the other has white ones, and vice versa. This method uses only images as input and the

Quantum	0	1	2	3	4	5	6	7	8	9
Distribution Black/White	9/0	8/1	7/2	6/3	5/4	4/5	3/6	2/7	1/8	0/9
Quantity of Patterns	1	9	36	84	126	126	84	36	9	1
Gray Scale Range	0-26	27-51	52-76	77-102	103-127	128-153	154-178	179-204	205-229	230-255
Quantized Color										
Example of Pattern										

Figure 2.2: Quantization of greyscale pixels into 10 levels of 3*3 black and white pixels (used in dithering) [19].

resulting images seemed to have a great amount of white noise. However, in some cases, it can use other types of images, besides the complementary ones described previously (figure 2.3a). Another approach was done by Liu et al. [11] to enhance the previous visual cryptography methods by preserving edge structures trying to suppress noise in smooth areas of the image, and resulted in coded images that kept the overall appearance of the original ones, except with only a little added noise (figure 2.3b).

Both of the above methods (figure 2.3) coded an input image into two, or more, different output images, and the decoding process involved the stacking of the output images [27] [11]. Moreover, they both have a common limitations of very low contrast in the resulting image of the decoding process.

Afterwards, a stenographic technique that performs small changes in the Least Significant Bit (LSB) [21] appeared for information encoding, which produced the original image, but with some imperceptible changes to the human eye. However, this technique is very sensitive to image deformations, so it ended up causing problems with the image decoding whenever the actual code was physically printed.

The examples above are only a small number of the stenographic methods created, but more recently however, the Graphic Code (GC) was created [15].

As explained in section 1.1, the GC is a composing element of UniQode and is a complex code whose main features are its high data storage capacity and its easy design integration. The pixel-based GC is also an approach based on dithering in order to create halftone stenographic images,

(a) Zhou et al. [27].

(b) Liu et al. [11].

Figure 2.3: Halftone-based Stenography (both with the complementary images first and the encoded image at the end).

but in a way that it is able to encode more information in a better-looking final image [15].

Also in section 1.1, it was mentioned that Graphic Code can be created in two different ways: with pixels or with icons. Both of these approaches give added flexibility to the code design and its contained information [14].

2.1.2 Comparison between MRCs

Although most of methods presented above were a considerable progress from the first 1D barcodes, most of them weren't aesthetically pleasing (random dispersion of black and white pixels, even when there was a possibility to have a base image in the background) and most couldn't encode a great amount of information.

In the table bellow (2.4) some of the previously presented codes are compared. It is shown the analysis of different 5 machine-readable methods and the Graphic Code method, where the data capacity and main features are represented for each code.

As can be observed, the QR code has all the important elements of the remaining codes on the right side which are: larger data capacity, small size and high-speed scanning. Nonetheless, the aesthetic part of the QR Code was clearly lacking, therefore the PEQRCs, by joining the most important QR Code qualities with a more pleasing appearance, introduced a new aspect to classic MRCs.

		Graphic Code	PEQRC	QR Code	PDF417	DataMatrix	MaxiCode
							
Developer		Ours	Many developers	DENSO Wave	Symbol Technologies	RVSI Acuity CiMatrix	UPS
Type		Graphic and PE	PE Matrix	Matrix	Stacked barcode	Matrix	Matrix
Data Capacity	Numeric	15288	7089	7089	2710	3116	138
	Alphanumeric	15288	4296	4296	1850	2355	93
	Binary	7644	2953	2953	118	1556	-
Main features		Enhanced Aesthetic, very large capacity, medium size, high-speed scanning	Aesthetic, large capacity, medium size, high-speed scanning	Large capacity, small size, high-speed scanning	Medium capacity, small size	Medium capacity, small size	High-speed scanning, small size

Figure 2.4: Comparison between the graphic code and other machine-readable methods [19].

However, as a result of striving for a code that could both be more visually appealing and encode more information, the GC method shows an improvement in both areas. Although comparing the visual aspect of codes is quite subjective, it is mostly agreeable that the aesthetic value of the GC is superior, when compared with other state of the art MRCs.

2.1.3 Code Generation and Decoding

According to Guruswami 2002 [6], the Encoding, also referred to encryption or code generation, is a function that maps a sender's input message, consisting of symbols, over an alphabet into a longer, redundant string, termed codeword.

Similarly, the Decoding function, also called decryption, maps strings of noisy received words to strings that the decoder believes were the originally transmitted message, which happens when the receiver gets a possibly distorted copy of the transmitted codeword, and needs to figure out the original message, which the sender intended to communicate [6].

The notion of encoding (or encryption) and decoding (or decryption) gained great importance in secure information communication in Claude Shannon's work (1948), and will be expanded upon in section 2.2.1.

In the case of the GC with pixels, the encoding method is based on the dispersed-dot dithering approach [23], explained before, where it takes each pixel in a greyscale image and expands it into a $k * k$ block of black and white pixels. These pixels are then quantized, producing a quantum for each pixel (figure 2.2). Each quantum represents the amount of black or white pixels of the $k * k$

block, and is then associated with a symbol from a set, called alphabet. This association between the quanta and the alphabet symbols produces a a dictionary [18].

Afterwards, some specific pixels in the image that fit the selected quanta are chosen and those pixels are encoded (according with the quantum-symbol association), resulting in the final coded image [18].

In the same way, according to the previous reference, the decoding process resembles the reverse of the above, as in taking the dictionary and the coded image, detecting the quantized symbols hidden and decrypting the input message.

The whole process is demonstrated in the image below (figure 2.5).

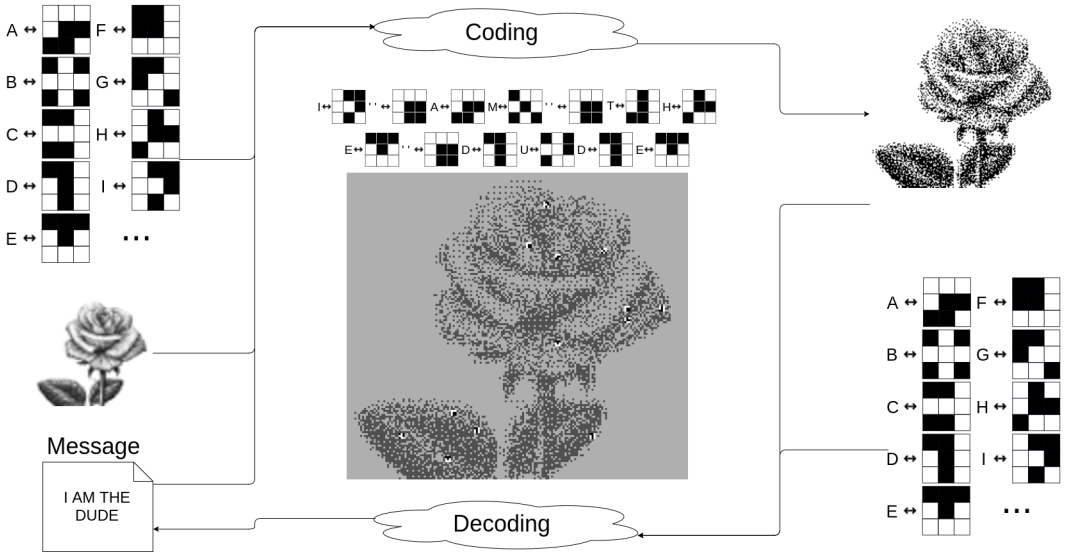


Figure 2.5: Encoding and decoding pipeline [17].

In contrast, when it comes to GC with icons, there is no need to quantize the base image pixels. Further work about this topic (coding and decoding of icon-based GC) will be developed with the progression of this dissertation.

However simple the encoding process might seem, it provides satisfactory security. For instance, even if it is known that an image contains a message, the decoder still has to identify the alphabet used in the dictionary, the pattern size, and several other elements [14]. In the next section (2.2) the security aspect of the cryptographic process will be expanded upon, as well as the added elements that a code can integrate as validation mechanisms.

2.2 Code Security and Further Data Processing

Communication secrecy is, in itself, assured by the applied cryptographic methods in a way that preserves the integrity and the confidentiality of the communicated information. This concept is usually referred to as ”cryptography for communication security” [25].

In contrast, to ensure that the encrypted information is decrypted correctly further data processing can be added, therefore validating the output information. Additionally, other methods can be added to the encryption pipeline, which is the case of data compression, where information to encrypt can go through a compression process in order to decrease its size and facilitate its transmission.

The following sections will explain some views on cryptography and its effects on code security, and a few methods for further data processing in order to either assure information integrity, or to compress more information in a single code.

Nevertheless, since this dissertation will focus mainly in UniQode's Graphic Code, the presented methods are all applicable for this case, as mentioned in [14], and will be expanded upon in section 3.4.

2.2.1 Cryptography as a security mechanism

The historical development of cryptography has been to try to design cryptographic systems (defined below) where one key can be used to encrypt a relatively long string of plaintext (i.e., one key can be used to encrypt many messages) and still maintain some measure of computational security [22].

We must firstly refer to a few important definitions in cryptography.

According to [25], *Cryptographic Algorithms* are mathematical algorithms that enforce information protection; a *Cryptographic System* is a set of cryptographic algorithms; *Plaintext* denotes the information to encode, that is, the input of an encryption algorithm, and, subsequently, *Ciphertext* stands for the encoded information by a cryptographic system; *Encryption* and *Decryption* are basic cryptographic processes that represent an action to transform a plaintext into a ciphertext or the opposite, respectively; A *Secret Key* (or cryptographic key) is an element of the cryptographic system that, added to the plaintext input, generates the ciphertext; *Cryptanalysis* is, according to Vaudenay [25], the theory of security analysis of cryptographic systems or, according to Koblitz

[8], the science of breaking codes. *Computational Security*, relative to cryptography, is equitably defined as a measure of how much computational effort is required to break a cryptosystem ([22]). Image (2.6) illustrates a few of these concepts .

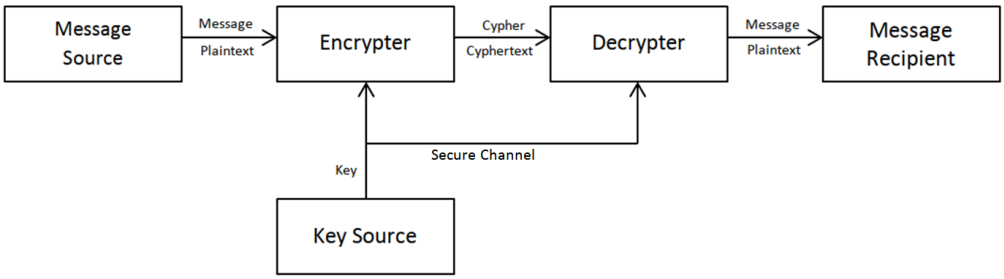


Figure 2.6: Adaptation of the Shannon encryption model [25].

A great influence in the study of cryptography was Claude Shannon's paper, in 1949, entitled "Communication Theory of Secrecy Systems" [22].

According to Shannon, cryptography is a method used to establish secure communications over insecure communication channels by using an extra hypothesis: a channel which already provides security. This channel is basically used to set up a confidential and authenticated symmetric key (cryptographic key that is symmetrically used in encryption and decryption processes). Once set up, symmetric keys can be used to communicate securely (i.e., confidentially and in an authenticated way) over insecure channels by using conventional cryptography [25].

Still, according to [25], there are contrasting views to Shannon's extra channel approach, but this discussion is beyond the scope for this project.

One essential element in cryptography is its cryptographic keys. This element can be labelled into two different types: private and public keys, and they are always associated in pairs, one for encoding and the other for decoding. The Private Key is associated with a symmetric cryptosystem (figure 2.6) and, as a result, it can also be called a symmetric key, and it describes a key that is only known to the encrypter and the decrypter, and used in both the encoding and decoding process. In contrast, a Public Key, also called asymmetric key, is accordingly associated with an asymmetric cryptosystem (figure 2.7).

The Public Key cryptosystem was formalised in 1976 by W. Diffie and M. Hellman [8]. This type of cryptosystem uses both a private and a public (or asymmetric) key, and consists on performing of the encryption process using a key know to the general public, while the decryption process

is accomplished with the private key, know only by a limited number of entities [25]. In other words, according to [8], in a public key cryptosystem the encipher cannot use the enciphering key to decipher the code. This system is represented in figure 2.7 and is called an asymmetric cryptosystem, since its encryption process is not symmetric to its decryption one [25].

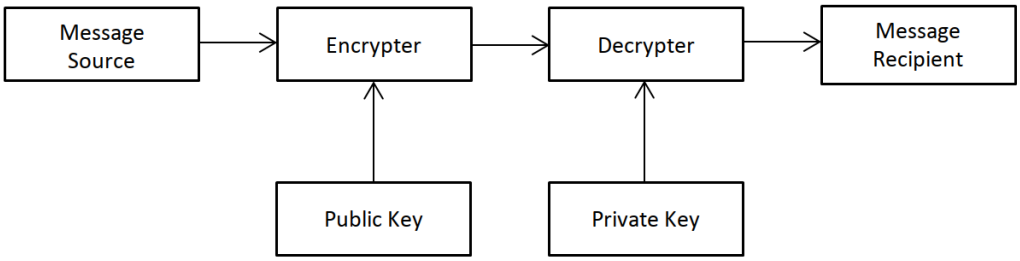


Figure 2.7: Asymmetric cryptosystem with a public key setting.

In some cases, a public key setting is advantageous since the sender does not need to know the private key, therefore keeping the private key in only the receiver's knowledge [25]. However, this benefit has the cost of being slower and riskier to implement, since there needs to exist, not one, but two different keys and there is no guaranteed security since half of the system is public knowledge [8].

As referred before, the Cryptanalysis of a cryptosystem is the science of breaking a code [8]. In order to break a cryptosystem, [8] states that there are two types of information that are needed. The first one is the general nature, or structure, of the system, that is, what kind of enciphering and deciphering techniques it uses. The second one is the choice of the parameters that are used in both of the previous techniques. Consequently, by having the two types of information, one has full knowledge of the system procedures.

However, [25] refers that cryptanalysis has a usually negative connotation, since the entities that try to break codes generally also break laws. Nevertheless, cryptanalysis can also be used by the cryptosystem creators to gauge how safe and robust the created system is, which, in turn, gives it a positive and constructive connotation.

Furthermore, in Claude Shannon's paper the concept for Perfect Secrecy for secret key systems is defined and its existence demonstrated. This concept is a mathematical term that denotes that a certain cryptographic algorithm is "unbreakable" [22].

On the other hand, the author also refers that the perfect secrecy term is mostly applicable in theoretical circumstances, that is, the only way to create an unbreakable code is to have at least as

many cryptographic keys communicated securely as the amount of plaintext, which is a considerable disadvantage in practical implementations, mainly in commercial use [22]. Douglas Stinson [22] refers to a previously existent cryptographic method (or cryptosystem), the One-Time Pad (Gilbert Vernan, 1917), that accomplished the perfect secrecy concept. However, the One-Time Pad cryptosystem had a practical implementation obstacle, that is, it was very hard to use commercially due to having to adopt a single different cryptographic key for every distinct message, which explained its lack of widespread use. Nonetheless, the author also adds that the One-time Pad has been employed in military and diplomatic contexts, where unconditional security is of great importance.

2.2.2 Further Data Processing for Cryptographic Systems

First of all, validation methods can be added to cryptographic systems to assure data integrity and error detection in the decryption process, in other words, to validate the decrypted code.

There are several validation methods that can be added to codes, mainly MRCs, to ensure that the encrypted message is transferred correctly from the sender to the receiver. In other words, to exchange information effectively, even when the medium of communication introduces errors [6].

Guruswami's work [6] centres on adding data redundancy to the encrypted message in a way that, even if the encrypted data is slightly tarnished, the message can still be decoded to its original form. The developed method, Reed-Solomon Error Correction Algorithm [6], is applied particularly in QR Codes (where up to 30% of the code area can be damaged [26]), and is one of the main features that promoted its popularisation [14].

Whereas the data redundancy method ensures that the code is decrypted even in the case of some damage, another existing validation method, the check digit, functions in a way that further guarantees the decoded information corresponds to the encoded one.

This other technique is a widely used approach in which a function analyses a message and always produces the same number for the same message [14]. This number is then added to the encrypted message and will serve as a means to check if the decrypted information corresponds to its check digit [7]. If the encoding and decoding check digits correspond, the message reconstruction is thus validated [14].

Besides being used as a validation mechanism, the check digit can also be an added security measure for the code. For instance, if the algorithm for the check-digit is slightly complex and

secret, it can generate a unique number for each message that anyone trying to hack the system would have great difficulty at recreating or corrupting.

Although both of the above methods are used to validate decoded information, there are other types of data processing methods that can be added to the cryptosystem pipeline and not for validation purposes. Such is the case of data compression.

Similarly to the check digit and data redundancy approaches, data compression is also added in the encryption process of the cryptosystem, and seeks to reduce the number of bits used to store or transmit information [13].

Generally speaking, data compression consists of taking a stream of symbols and transforming them into a smaller stream, if the compression is effective [13], and is widely used to increase the capacity of data storage devices or to transfer data faster on networks [12].

As stated by [13], there is a great amount of existing compression methods at present which encompass such a wide variety of software and hardware compression techniques that can be so unlike one another that they have little in common except that they compress data.

Mainly, there are two types of data compression: lossless and lossy.

Lossy data compression concedes a certain loss of accuracy in exchange for greatly increased compression. That is, greater compression has more value than the retention of all of the original information, and proves effective when applied to graphic images and digitized voice, where the output data loses definition and accuracy, when compared to input one, but can still be easily usable. Most lossy compression techniques can be adjusted to different quality levels, gaining higher accuracy in exchange for less effective compression [13].

In contrast, lossless compression denotes the type of compression that consists of techniques guaranteed to generate an exact duplicate of the input data stream after compressing, and later expanding, the data. This is the type of compression used when storing database records, spreadsheets, or word processing files. In these applications, the loss of even a single bit could be catastrophic [13].

Furthermore, it is important to note that this method ties in with Shannon's Information Theory (mentioned above) because of its concern with redundancy. Redundant information in a message takes extra bits to encode, and if we can get rid of that extra information, we will have reduced the size of the message [13]. As data redundancy was presented above in a good light because of

its ability to avoid some decoding errors, it also increases the quantity of information to encode, which is not usually a favourable factor. However, as [13] states, this fact can be helped with the addition of data compression.

3

Program Development Process

This chapter will describe the methods used in the development of this project. Firstly, a general overview of the project's goals will be made; then all the functional elements comprising the used coding system (Graphic Code) will be described, as well as an outline on the developed algorithms and their implementation; and, finally, the application of the security and validation methods to icon-based GC and the used tools to accomplish all of the above, joined by a few use-case examples.

3.1 Program Creation

The goal of this project was not only to further establish the icon-based GC creation process and regulations, but also to develop a tool to assist with the design creation while automatically generating each code's documentation.

Therefore, firstly, the Minimum Viable Product (MVP) was established and it comprised the development of a standalone program that, with input encoding information, would generate a fully encoded VI. Furthermore, the program would also have to generate extensive documentation on each type of created code, in order to describe its encryption keys, along with proper printing files with the purpose of aiding the industrialisation process.

Moreover, this program needed to have an intuitive and easy to use interface, so as to facilitate as much as possible a process that was originally quite complex and time-consuming.

The early establishment of the desired framework and work-flow for the design tool was crucial to its development and, consequently, the created UML use-case diagram (using the online Visual Paradigm [1] tool) is represented in figure 3.1.

This UML represents the system variables that most influence it, which are its inputs, message and graphic elements, and the placement of the graphic elements within the code image. These

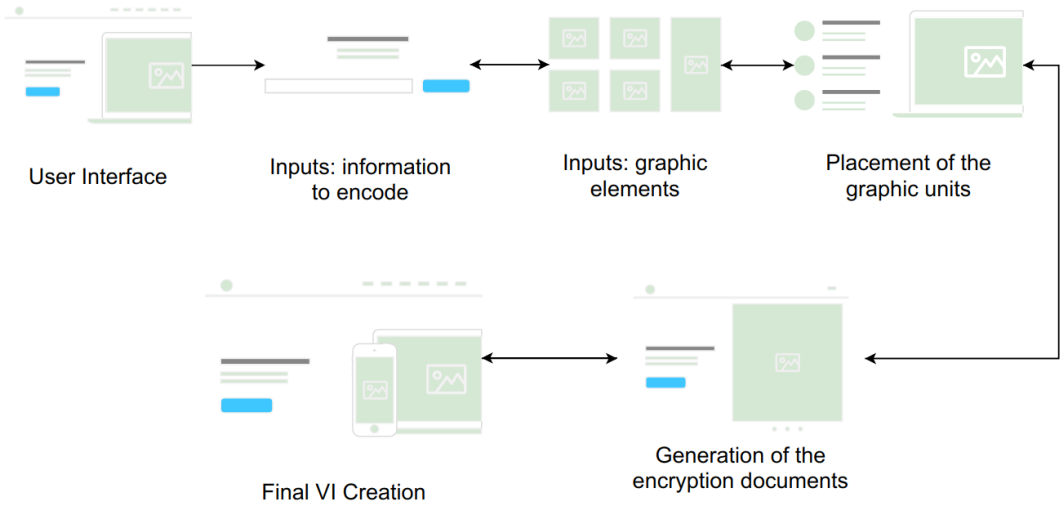


Figure 3.1: Use-Case Diagram for the icon-based GC platform.

variables and their roles will be discussed at length in the following sections.

3.2 Graphic Code Taxonomy

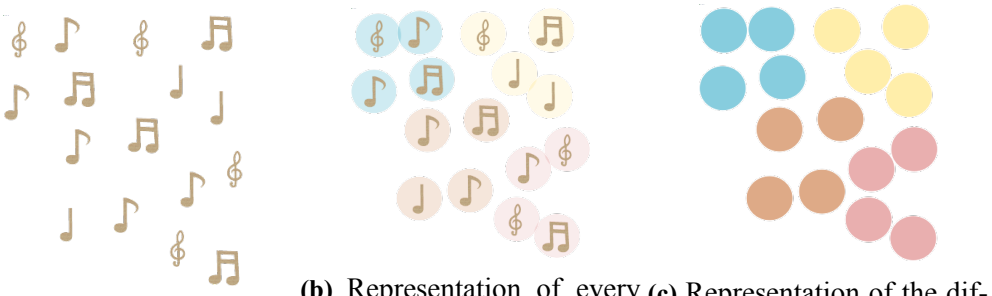
To better understand the Graphic Code system, a few key concepts need to be clarified and explained. Therefore, a list with the essential definitions is presented below [14]:

- A *Visual Identity* (VI) is an image containing an encoded message (figure 3.2a), with a suitable placement of *graphic units*;
- A *Graphic Unit* (GU, figure 3.2b) is, in the case of this project, a cryptographic primitive[1] and, therefore, an *Icon*;
- A *Cell* (figure 3.2c) is a group of one, or more, GUs and its creation is the basis for the encoding process;

In addition to the presented above, one of the essential elements to encrypt a message is the encryption key. Within this project, the encryption key is partially represented by a *Dictionary* (which will be further explained in section 3.4). A dictionary integrates a set of associations between a symbol (mainly a character) and sequence of GUs in a cell, in a way that one sequence is associated with, at most, one symbol. A set of dictionary symbols defines an *Alphabet*.

An example representation of a dictionary and alphabet is presented bellow, in figure 3.3. In figure 3.3a, each icon is associated with a number (’0’ to ’3’); the alphabet is composed by the elements

[1] A cryptographic primitive is equivalent to the most basic building block.



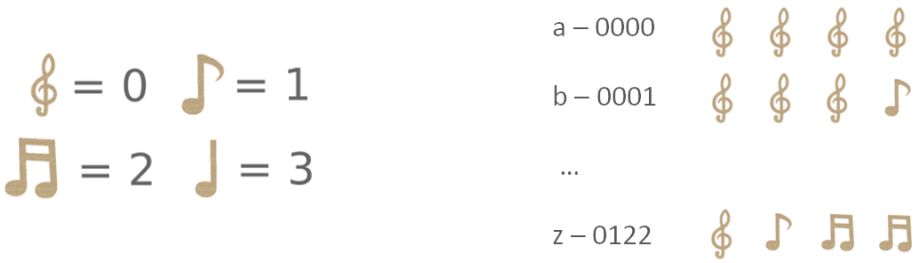
(a) Visual Identity.

(b) Representation of every Graphic Unit (GU).

(c) Representation of the different cells within the VI.

Figure 3.2: Graphical representation of icon-based Graphic Code taxonomy [14].

from 'a' to 'z' (figure 3.3b); and the association between each alphabet symbol and a sequence of icons ('0000' to '0122') makes up the dictionary (figure 3.3b).



(a) Enumeration and coding of the icons.

(b) Resulting alphabet and dictionary.

Figure 3.3: Graphical representation of a Graphic Code dictionary, adapted from [14].

3.3 Developed Algorithms

3.3.1 Quantification of Code Information

One critical algorithm for icon-based Graphic Codes design is the quantification of maximum coded information. That is, the computation of how many symbols we need to encode a certain input message, and the most efficient combination of parameters.

First of all, an important requirement of this project was to allow the encoding of special characters, for example Latin letters such as 'ç'. These types of characters, however, are usually in a multiple-byte format and, in order to encode as many different characters, in the simplest way possible, the UTF-8 format was adopted.

This format, 8-bit Unicode Transformation Format (UTF-8), is a way to represent a character in sets of 8 bits, which meant, in this case, that every possible character could be encoded, but not

in only one cell (one byte). In other words, as an illustrative example, since the 'ç' character is represented by two sets of 8 bits (two bytes), it will, therefore, be encoded in two cells.

Nevertheless, for demonstration and explanation purposes, we will assume that to encode *x* amount of symbols, we need to have *x* amount of cells in the code.

Consequently, if there is an *x* amount of symbols to encode, we need, at least, as many dictionary icon sequences to associate to each message symbol. We can demonstrate the above statement by referring back to figure 3.3b, where, to represent the alphabet's cardinality of 26 elements (a-z), there needs to be one singular sequence of icons for each element.

Therefore to obtain the necessary number of icon sequences, a specific number of different icons is needed and, by using the mathematical concept of arrangements with repetition, we can represent this number of icons by distributing them within the graphic units in each cell and check if the condition for the icon sequences is met, as is represented in the following equation:

$$IconSequences \leq Icons^{GraphicUnits} \tag{3.1}$$

This algorithm explanation is quite abstract, but the main concept to understand is: the number of different icons, raised to the power of the number of graphic units per cell has to be greater or equal to the number of alphabet elements.

As a consequence of this main algorithm, a few other resulting notions should be underlined:

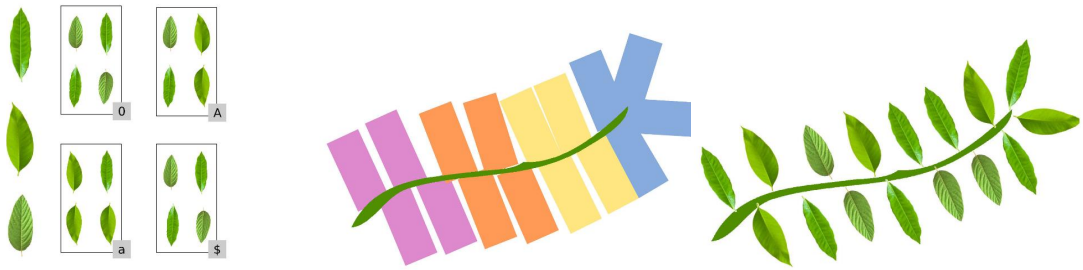
- The minimum number of different icons (I) needed for the code depends on the number of cells (C) and the number of graphic units per cell (GUs):

$$C \leq I^{GUs}, I \in \mathbb{N} \Rightarrow I = \lceil log_{GUs} C \rceil = \left\lceil \frac{logC}{logGUs} \right\rceil \tag{3.2}$$

- The distribution of the icons within the cell's graphic units make up the icon sequence. If we refer back to figure 3.3, the sequence of the four treble clef icons (with the value 0 in figure 3.3a) distributed within the cells four graphic units, represent the character 'a';
- At least one sequence needs to be associated to each alphabet element, which means that more than one sequence can be associated to an element. By taking the above example into consideration again, since a sequence of four quarter notes (with the value 3 in figure 3.3a) is not associated with any other alphabet element, we could also represent the letter 'a' with

this sequence;

As an additional example of the method above we can analyse the following image (figure 3.4).



(a) Input icons, alphabet and dictionary. **(b)** Graphic units locations. **(c)** Final code.

Figure 3.4: Icon-based GC Example.

The first image (figure 3.4a) shows the input icons, $I = 3$ (3 vertical leaves), the graphic units per cell, $GU = 4$ (4 leafs represent each message symbol, therefore, each cell), the alphabet with up to $Icons^{GraphicUnits} = 3^4 = 81$ elements/icon sequences, and the dictionary (association of the input icons with the alphabet). Then, the second image (figure 3.4b) shows all the available graphic units positions, 16 in total, distributed by 4 different cells, $C = 4$, which means we can encode 4 message symbols in this case. The third, and final, image represents an example of a final code that uses the previous elements to encode a 4 symbol messages.

3.3.2 Check Digit

As explained in section 2.2.2, the check digit is an added measure to check the validation of the code during decryption.

Moreover, since this mechanism's complexity could be easily increased and work as an additional security measure, it motivated its development and implementation and, consequently, the algorithm to associate a check-digit during the encoding process was developed.

The check-digit, in itself, is a value that is calculated from each input message and is incorporated in the encoded information, in such a way that, during the decoding process, the decoder could check if the output message was equivalent to the input one, without knowing specifically what that message was. This was achieved by recalculating the check digit from the decoded message and matching it up to the one present in the code.

This algorithms, as many others, is not foolproof and has its limitations such as, if the check

digit is not correctly decoded, the decoding process would fail, even if the rest of the message was decoded properly.

This algorithm performed as described in the following steps:

1. The input message symbols are associated with a numbered sequence;
2. The number of each sequence is added. That is, the first sequence has the number 1, the second sequence has the number two and so forth, and these numbers are the ones that are added;
3. The remainder of the previous value by the number of cells is calculated;
4. The resulting value is corresponded with a numbered sequence. In other words, if the resulting value is 5, the corresponding icon sequence is the fifth present in the dictionary;
5. That sequence of icons is added as another cell to the VI, and represents the Validation Cell;

For example, if the alphabet's elements were 'a-z' and the message to encode was "adab", the step one of the algorithm was to associate 'a' with the first sequence, 'b' with the second and 'd' with the fourth. The next step would be to add 1+4+1+2 = 8 and then calculating their remainder by 4, which returned the value 0. Finally, the check digit value would be the 0+1 sequence, therefore, the first one, which meant that, in the final code, the check digit icon sequence would be the same as the one for the character 'a'. Therefore, if the check-digit is decoded properly and returned the value of the first sequence, the decoder would verify, hence validate, that the decoded message is equivalent to the originally encoded one.

However simple this algorithm might seem, it was implemented in such a way that its complexity could be easily increased, if ever needed, therefore increasing the cryptosystem's security. This topic is an important strand of this project and will be further explored in the following section 3.4.

3.4 Security and Validation Aspects of Icon-based GC

As presented in this dissertation's state of the art, code security can be assured by its cryptographic system and be measured by performing its cryptanalysis. Additionally, a multitude of validation methods can be applied to assure its correct decryption which may, subsequently, add to the system's security aspect.

As mentioned in section 2.2.1, the two types of information needed for a systems cryptanalysis

are the general nature of the system and the parameters used in the system's procedures. Therefore, applied to the developed system, the first type of information is quite easy to obtain or deduce (it encodes each message symbol in icons), however, the second type of information is different for every created code and would be very difficult to gauge without the input details.

Consequently, as mentioned in [18], the computational cost of the cryptanalysis for pixel-based GC is very high because of the multiple degrees of freedom that it presents. Moreover, since the process of encryption of icon-based GC is similar to the pixel-based one, seeing that both need elements like an alphabet and dictionary and specified positions for the encoded elements, the developed process in this dissertation has also a very high cryptanalysis computational cost.

Therefore, to perform the cryptanalysis for this process, one would have to discover which alphabet was used in the dictionary, the position of each cell within the image, the position of every graphic unit within each cell, among a few other aspects [18]. These aspects, however, have such small limitations (resulting of high degrees of freedom) that to figure out the correct pattern of every code element and their actual meaning would demand a great computational cost, consequently proving the high cryptanalysis value.

Seeing that, in the case of the developed method, the used cryptosystem is symmetric, the resulting cryptographic key is private. With this key a decision was made to divide into two parts, the dictionary and the configuration (figure 3.5), with the purposes of both data organisation and additional code security. These parts, that were created in the form of XML format files, contained all the necessary information to either encode or decode a visual identity, and are represented in the following listings 3.1 and 3.2.

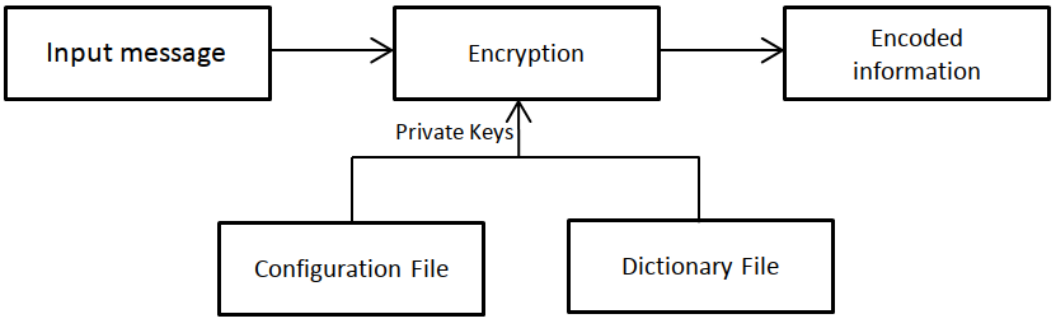


Figure 3.5: Developed cryptosystem with the separate private keys.

Listing 3.1: Configuration File

```
<?xml version="1.0" encoding="UTF 8"?>
<ConfigurationFile>
    <CodeElements Cells="5" Icons="3" GraphicUnits="3"/>
    <Positions Cell="1" Icon="1" X="369" Y="85"/>
    <Positions Cell="1" Icon="2" X="363" Y="179"/>
    <Positions Cell="1" Icon="3" X="361" Y="251"/>
    <Positions Cell="2" Icon="1" X="348" Y="322"/>
    <Positions Cell="2" Icon="2" X="280" Y="248"/>
    <Positions Cell="2" Icon="3" X="287" Y="195"/>
    <Positions Cell="3" Icon="1" X="170" Y="294"/>
    <Positions Cell="3" Icon="2" X="144" Y="429"/>
    <Positions Cell="3" Icon="3" X="211" Y="416"/>
    <Positions Cell="4" Icon="1" X="211" Y="416"/>
    <Positions Cell="4" Icon="2" X="211" Y="416"/>
    <Positions Cell="4" Icon="3" X="211" Y="416"/>
    <Positions Cell="5" Icon="1" X="339" Y="418"/>
    <Positions Cell="5" Icon="2" X="414" Y="383"/>
    <Positions Cell="5" Icon="3" X="427" Y="330"/>
    <Positions Cell="6" Icon="1" X="459" Y="295"/>
    <Positions Cell="6" Icon="2" X="494" Y="375"/>
    <Positions Cell="6" Icon="3" X="471" Y="440"/>
    <Paths>/baseImage.png</Paths>
    <Paths>/icon1.png</Paths>
    <Paths>/icon2.png</Paths>
    <Paths>/icon3.png</Paths>
</ConfigurationFile>
```

As is shown in the configuration file example above, the main encryption inputs are stored in this file with the names ”cells”, ”icons” and ”graphic units”. Moreover, the coordinates of the positions of every icon in each cell and the paths for the input images are also written in this file. As a result, this constitutes all the basic configurations necessary for each created code and represents the first half of the cryptographic key. This half was created in such a way that it would be independent from the other half, the dictionary file, and could be used with multiple different dictionary files, depending on the user preferences.

Listing 3.2: Dictionary File

```
<?xml version="1.0" encoding="UTF 8"?>
```

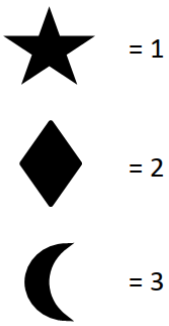
```
<DictionaryFile>
    <ConfigFile>predefinedDictionaries/config_A  Z_3i_3gu.xml</ConfigFile>
    <MaxElements>27</MaxElements>
    <Alphabet>
        <Sequence>111</Sequence>
        <Sequence>112</Sequence>
        <Sequence>113</Sequence>
        <Sequence>121</Sequence>
        <Sequence>122</Sequence>
        <Sequence>123</Sequence>
        <Sequence>131</Sequence>
        <Sequence>132</Sequence>
        <Sequence>133</Sequence>
        <Sequence>211</Sequence>
        <Sequence>212</Sequence>
        <Sequence>213</Sequence>
        <Sequence>221</Sequence>
        <Sequence>222</Sequence>
        <Sequence>223</Sequence>
        <Sequence>231</Sequence>
        <Sequence>232</Sequence>
        <Sequence>233</Sequence>
        <Sequence>311</Sequence>
        <Sequence>312</Sequence>
        <Sequence>313</Sequence>
        <Sequence>321</Sequence>
        <Sequence>322</Sequence>
        <Sequence>323</Sequence>
        <Sequence>331</Sequence>
        <Sequence>332</Sequence>
        <Sequence>333</Sequence>
        <Correspondence>65</Correspondence>
        <Correspondence>66</Correspondence>
        <Correspondence>67</Correspondence>
        <Correspondence>68</Correspondence>
        <Correspondence>69</Correspondence>
        <Correspondence>70</Correspondence>
        <Correspondence>71</Correspondence>
        <Correspondence>72</Correspondence>
        <Correspondence>73</Correspondence>
        <Correspondence>74</Correspondence>
```

```
43          <Correspondence>75</Correspondence>
44          <Correspondence>76</Correspondence>
45          <Correspondence>77</Correspondence>
46          <Correspondence>78</Correspondence>
47          <Correspondence>79</Correspondence>
48          <Correspondence>80</Correspondence>
49          <Correspondence>81</Correspondence>
50          <Correspondence>82</Correspondence>
51          <Correspondence>83</Correspondence>
52          <Correspondence>84</Correspondence>
53          <Correspondence>85</Correspondence>
54          <Correspondence>86</Correspondence>
55          <Correspondence>87</Correspondence>
56          <Correspondence>88</Correspondence>
57          <Correspondence>89</Correspondence>
58          <Correspondence>90</Correspondence>
59          <Correspondence>32</Correspondence>
60      </Alphabet>
61  </DictionaryFile>
```

In the dictionary file example (listing 3.2) however, the elements pertaining the dictionary are present. Firstly, this half of the cryptographic key is always dependent upon the configuration file, since the cardinality of the alphabet depends on the number of cells, graphic units and icons. As a result, the first element of this file is the name of the corresponding configuration file. Secondly, the maximum number of elements the dictionary can hold is written, and represents the resulting value of the equation 3.1. In this case, with the elements from the configuration file we obtain $3^3 = 27$, which is the maximum number of icon sequences we can create with 3 icons and 3 graphic units per cell, and, therefore, the maximum number of elements the dictionary can accommodate.

The following written elements are the icon sequences. The first sequence, '111', represents a cell which has three identical icons and is illustrated in figure 3.6b, where the icon with the value '1' is accordingly exemplified by the star icon in figure 3.6a. If, for instance, we needed to represent the sequence '313', the icons order would be the moon icon, followed by the star icon and the moon icon again. The same process applies for the remaining sequences.

Last but not least, the final elements in the dictionary files are the values of the symbols that will be corresponded with each sequence. These numbers represent the decimal value of each UTF-8 symbol that makes up the alphabet. In the case of the example presented above, the first value is



(a) Example of code icons

(b) Example of the first sequence of the presented dictionary

Figure 3.6: Exemplification of a dictionary's icon sequences.

65 which corresponds to the upper-case letter 'A' in UTF-8 format and, similarly, the fourth value, 68, corresponds to the UTF-8 character 'D'. Consequently, the value 65 is represented by the first sequence, '111', and, therefore, 3 star icons in a row.

The set of elements presented in the segments explained above make up the created cryptosystem's cryptographic key. And, consequently, by dividing the encoding information into two separate files we made sure that, to decrypt the code, one always needed both files.

However, as a validation mechanism, the previously described check digit method (section 3.3.2) was used. Consequently, as the check digit was added as another encrypted cell (validation cell) to the rest of the code, it also became another element to verify the code's integrity during decryption. This element can be observed in the lines 14 and 15 of the configuration file (listing 3.1) as the sixth cell of a five cell code. This means that the input message has five symbols, therefore five cells, and the check digit is encoded and added as the sixth one.

The resulting process and its implementation is represented in figure 3.7.

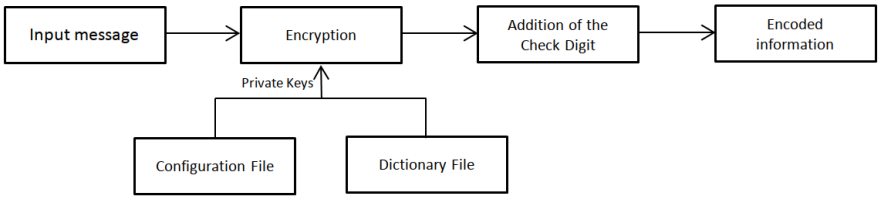


Figure 3.7: Encryption model of the developed cryptosystem.

Additionally to the methods referred above, two other data processing methods were presented in this thesis' state of the art: data redundancy and data compression.

Neither of these methods were implemented during the development of this program, as they did not represent the essential parts of the MVP. However, as is referred in [18], both of these methods could easily be incorporated in the cryptosystem's pipeline GC encoding (both pixels and icons), and the resulting process would resemble the model in figure 3.8.

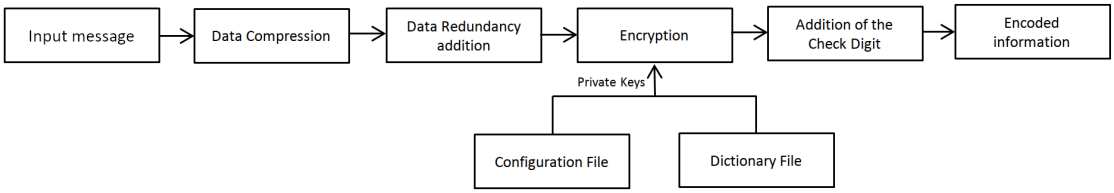


Figure 3.8: Possible cryptosystem encryption model with added data redundancy and compression.

As is shown in the above figure, both the data compression and data redundancy affect the encryption process and its values, since they need to be implemented before the data is encrypted, unlike the check digit addition, that does not influence the encryption input, but only adds to it [14]. Both of these method's implementation can be considered as future work of this dissertation. Nevertheless, the proposed work would consist on the application of a data compression algorithm to the input message, then applying a redundancy algorithm to the resulting data and, finally, proceeding to encrypt the compressed data.

Moreover, since the presented cryptosystem is symmetric, the decryption process would be represented by the inverse of the encryption one. However, the decryption of icon-based graphic codes is outside of the scope of this project and could also be considered future work, along with the suggestion presented above.

3.5 Implementation Tools

To achieve the standalone program described previously as the MVP in section 3.1, a few tools for image manipulation and software development were required.

Firstly, to more easily create a software program of greater dimensions, an Integrated Development Environment (IDE) was used, Qt Creator in particular, with C++ language. This singular tool was chosen not only because it eased the organisation of all the files, but it also had a feature for GUI creation, which proved very useful to accomplish the needed intuitive and easy to use final product.

As an image manipulation tool, however, the OpenCV library was used in such a way that the user would be able to interact with a window containing and image in order to place the code icons.

Additionally, whenever a new type of VI is created, its encryption elements needed to be stored for further work. Therefore, the icons placements, the dictionary elements and other code components were appropriately stored in configuration and dictionary files in as XML format, by using an XML parser, for a more standardised and formal document.

Finally, to generate the program documentation needed to formalise the product creation, Doxygen documentation generator was used. Part of the resulting documentation is annexed in section 6.

The tools described above were used in a way that the process of the program creation was formalised, by using XML format files and creating the program documentation, and resulting product achieved an acceptable value, by completing the proposed MVP.

4

Discussion of Results

This chapter presents the final developed product, its advantages and its limitations, followed by examples of created Visual Identities and their corresponding documents. Lastly, an analysis of every example will be made pertaining the quantification of information and each resulting code, and an industrial design use-case will be presented.

4.1 Graphic User Interface

Since the main goal of this dissertation was to create a user interface that allowed icon-based Graphic Code creation, the resulting platform is represented in the following image (figure 4.1).

The user interface window has, per requirements, various functionalities and different possible outputs.

First of all, the GUI window is divided in 4 parts:

- The first part, ”To encode”, is where the user chooses whether to input a number of cells, a message or a message file. This segment represents the input of the encoding pipeline and, consequently, the element the user wants to communicate securely and confidentially;
- The second section, ”Configuration file”, is where the first part of the encryption key is formed. The option of either creating a new file or loading an already existing file is given to the user. This is where the user chooses the number of GUs per cell and the number of icons, while also loading the input images and, finally, placing the icons. Or, in contrast, the user can simply load an existing file. In order to have both more creation flexibility and added safety, this first part of the encryption key is independent from the second, as explained in section 3.4;
- The next segment, ”Dictionary file”, represents the remaining half of the encryption key.

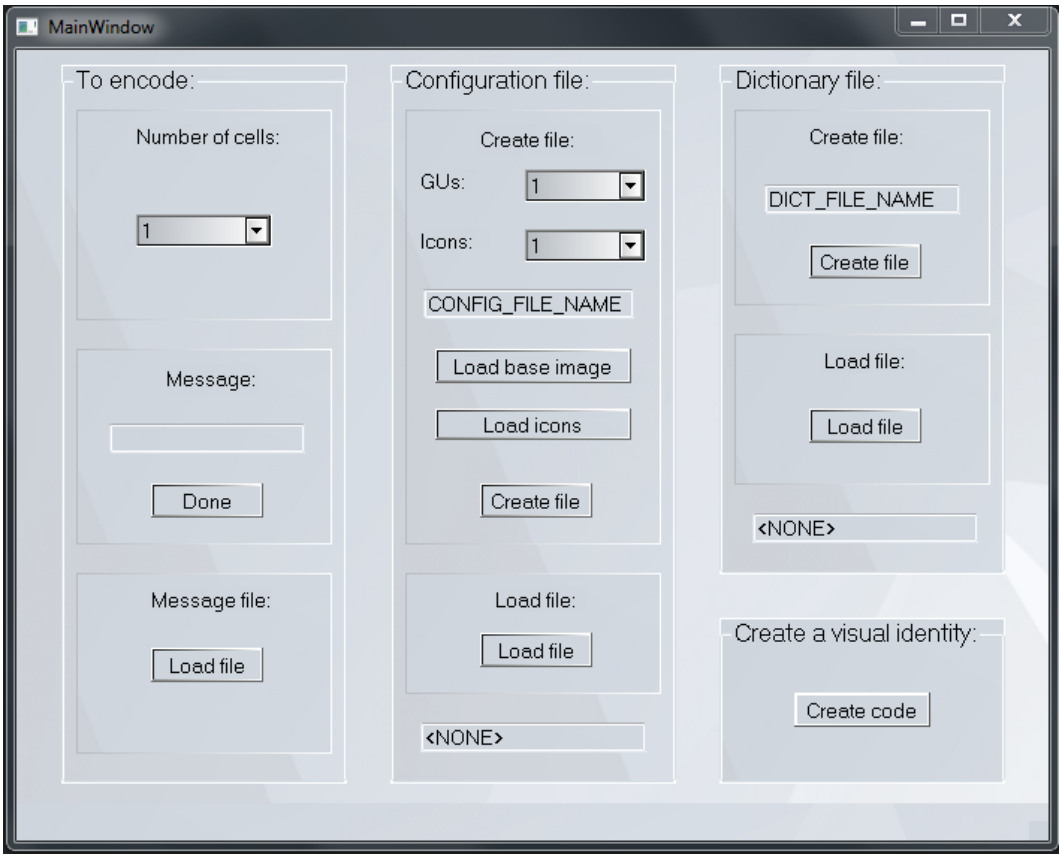
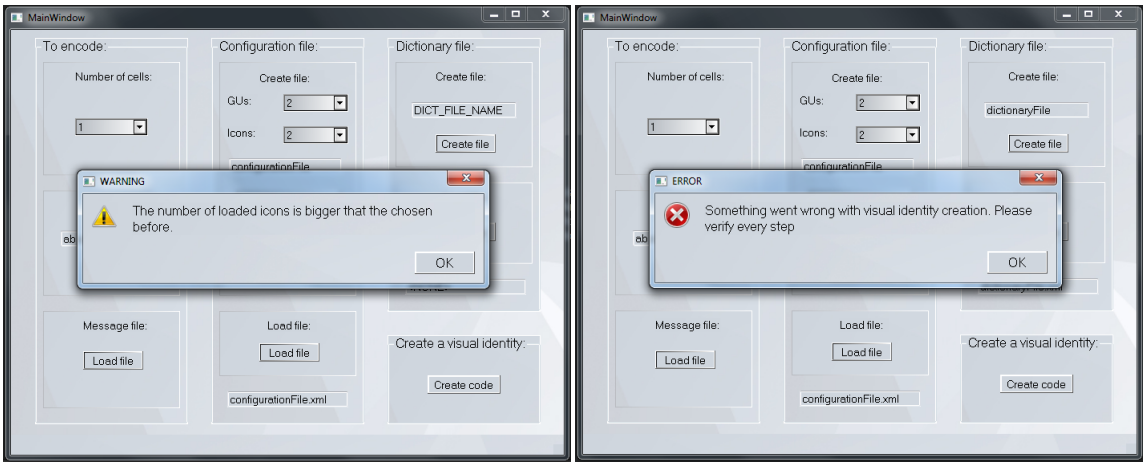


Figure 4.1: Created user interface with Qt Creator's GUI tool.

Whereas the configuration file is independent from this part, the dictionary file always depends on only one configuration file. This has to do with the fact that its elements, mainly the icon sequences, depend on the numbers of graphic units and icons, which are both configuration elements. Also in this section of the window, the user can choose between loading or creating the file, and the process is completed without the need of any additional inputs;

- The fourth, and last, part refers to the creation of the visual identity. This is accomplished by joining all the previous input elements, and generating one, or multiple, codes that represent the output of the encoding pipeline. This segment not only creates the image format of the output codes, but it also creates a PDF file with the images in order to facilitate any possible industrialisation process that might be needed.

Secondly, besides the windows main functionalities, during the development of this interface, usability and practicality were also taken into consideration. For this reason the user can complete a multitude of tasks with several degrees of freedom, while still complying with the algorithm rules by being presented with several warning or error message boxes, in case of faults or failures (figure 4.2).



(a) Warning message box example. **(b)** Error Message Box Example.

Figure 4.2: GUI message boxes.

Typical workflow - creation of a VI with an input message

Additionally, a representation of a typical workflow is represented in figure 4.3. This illustration describes a process with the goal of creating an encoded VI, and starts with the interaction of the user with the GUI. Firstly, the user chooses to input a message to encode, instead of the other two options of loading a message file or only inputting the number of cells to encode. Secondly, the user chooses the number of different icons and graphic units per cell he/she wishes to use. Afterwards, the codes images are inputted and the creation of the configuration file is completed. Then, the user chooses to create a dictionary file based on the previous inputs, the input message and the configuration elements. Finally, the visual identity is created and an image and corresponding PDF file are generated.

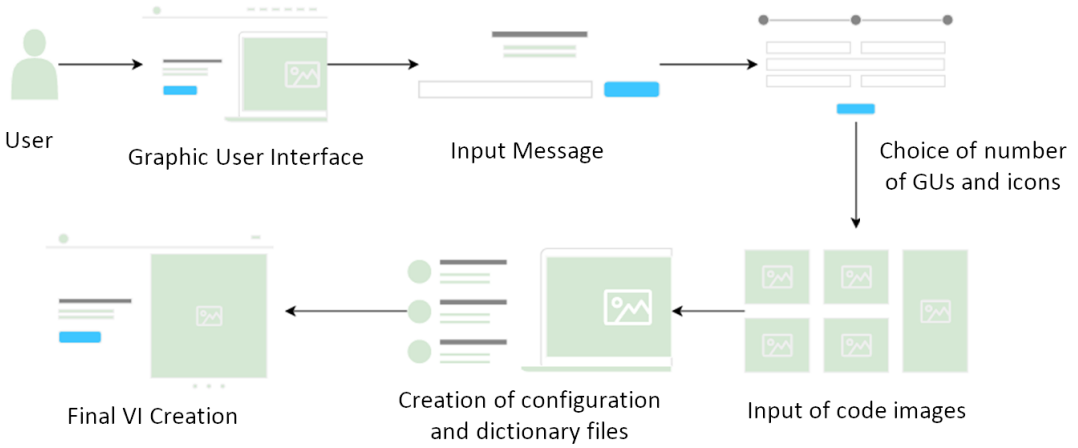


Figure 4.3: Typical workflow - creation of a VI with an input message.

Alternative workflow - creation of configuration and dictionary files

An alternative possible workflow, illustrated in figure 4.4, begins with the goal of only generating the configuration and dictionary files for future use. This process starts with the users choice of a number of cells, instead on the previous input message, and is followed by the choice of the number of graphic units and icons. Afterwards, even if the goal is not to create a final visual identity, the user still needs to load the code images, since one of the elements of the configuration file is the image's paths. Finally, the user can choose whether to create a dictionary file or not, because there is no input message symbols to correspond to icon sequences. That is, if the user chooses to create a dictionary, it would be one without "Correspondence" elements (refer back to listing 3.2), that would be created in a future process.

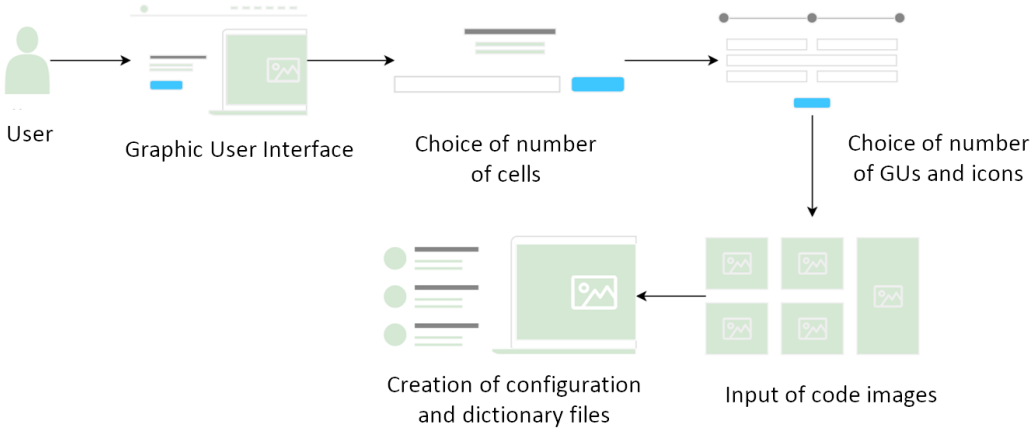


Figure 4.4: Alternative workflow - creation of configuration and dictionary files.

Alternative workflow - creation of multiple output codes from an input message file

This final workflow (figure 4.5), fits with the previous one, as it uses already created configuration and dictionary files. In this case, the user load a text file with several messages he/she wishes to encode. Then, the user loads the configuration and dictionary files. The process of loading the configuration file instead of creating it from scratch eliminates the need for the user to choose the number of graphic units and icons and to load input images, since all of this information is already contained in the configuration file. The dictionary file, however, is loaded, and the "Correspondence" elements (refer back to listing 3.2) are created or updated file. Finally, this process results in multiple encoded images and a respective PDF file with all the codes.

This last presented workflow would be the most appropriate case for industrialisation purposes, since multiple codes would be created at once from a single text file with as many messages as the

user wishes, within the codes limitation. This limitation has to do with the maximum number of codes the cryptosystem can generate with each singular configuration, and will be explained in the next section.

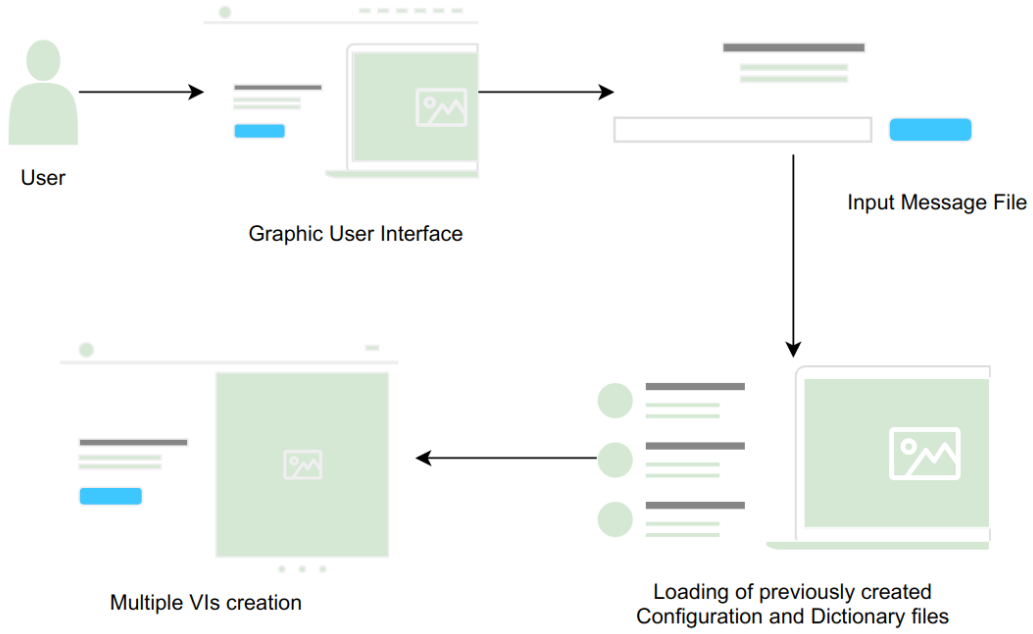


Figure 4.5: Alternative workflow - creation of multiple output codes from an input message file.

4.2 Analysis of Generated Visual Identities Examples

With the formerly presented tool and its flexible usability, and by giving a substantial amount of creation freedom to the user, a great range of output codes can be generated.

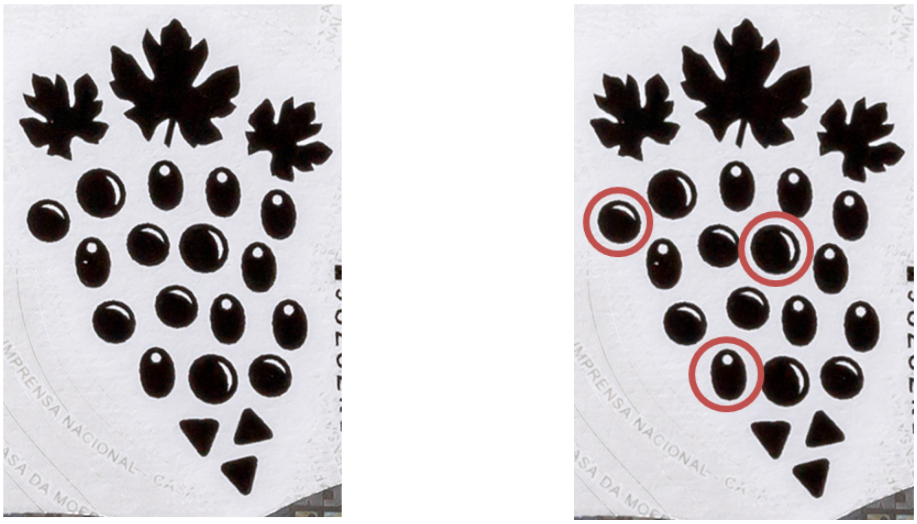
The codes created with the developed tool can be analysed with respect to the quantity of information each can encode, the different encoding elements each has, and the maximum number of possible visual identities (or codes) that can be created from it.

Consequently, a few examples of visual identities ensue, accompanied with each respective analysis and comparisons.

4.2.1 Example 1

This example (figure 4.6a) was first created by the INCM without the use of this project's developed tool and will only serve a comparison subject, since the goal of this project was to create a way to easily generate these types of codes. So, with this, we will analyse its elements.

First of all, this visual identity was intended to represent a bunch of grapes and was designed for the wine sector. It is important to note that the three leaves on top and the three triangles on the bottom are used as static icons for position detection, much like the mandatory squares in QR Codes, presented in chapter 1. Therefore, the actual encoding is done with the ”grape” icons in between. Since this is a demonstration code, its correct configurations were not made public and the composing elements that will be deduced from the image may not be the actual ones.



(a) First icon-based stamp created by INCM. **(b)** Different icons within the visual identity.

Figure 4.6: Example 1.

By analysing this practical case image, we can detect three different grape icons, singled out in figure 4.6b, therefore the number of icons of this VI is 3. However, this number could easily be adjusted to accommodate other grape icons or other configuration goals.

No other information is obvious from this example which also shows how strong its cryptanalysis value is. But, for comparison purposes, let's assume that, with the total of 16 grape icons in the image, there are 4 cells with 4 graphic units each, as is demonstrated in figure 4.7.

The number of possible icon sequences, as demonstrated before with the equation 3.1, is $3^4 = 81$. Then, if we want to determine the maximum number of codes that can be created with this configuration, we have to, again with arrangements with repetition, distribute the number of sequences by the number of cells in the code, which returns the value $81^4 = 43.046.721$. Consequently, with this code's configuration a company could label, trace and verify above 43 million units of products.

The process to obtain the value of 43.046.721 different possible codes would start with a code

Figure 4.7: Grouping of the example code's icons by cells.

where all the icon sequences, in all 4 cells, would be '1111', which is something similar to figure 4.8a, and would end with a code where all the cell's icon sequences would be '3333', as is shown in the example in figure 4.8b. Therefore this value is obtained by associating every single one of the 81 sequences to the 4 existing cells.

(a) Example of a first code generated with the presented configuration.

(b) Example of a last code generated with the presented configuration.

Figure 4.8: Exemplification of a code's icon sequences.

Element	Value
Number of cells	4
Number of graphic units per cell	4
Number of different icons	3
Number of possible icon sequences	81
Maximum number of VIs to generate	43.046.721

Table 4.1: Analysis of Example 1.

4.2.2 Example 2

By choosing the following icons and base image (figure 4.9), we can create the subsequent code, represented in figure 4.10.

(a) Base image. **(b)** Icon 1. **(c)** Icon 2. **(d)** Icon 3.

Figure 4.9: Input images.

This example, unlike the one presented before, was created with the developed tool.

During its creation process the number of cells chosen was 3 and the number of graphic units and icons was also 3.

Moreover, the validation cell is also present, which accounts for 12 icons in total, distributed by 4 cells (3 code cells plus 1 validation cell) with 3 graphic units each (figure 4.10).

As a consequence of the chosen configuration, the number of icon sequences that are possible is $3^3 = 27$.

However, as can be seen in table 4.2, the maximum number of generated VIs is considerably lower than the first one. That is due to the fact that this second example has one less cell and graphic unit per cell, which goes to show that a codes outcome can be very flexible by applying only small adjustments to a code's elements.

Also, with the use of the developed tool, in order to give the user all the necessary information, the program calculates the maximum number of VIs each specific configuration is able to generate,

Figure 4.10: Final code example.

and communicates that information to the user.

However, if the configuration excluded the need for a check digit, we would be able to encode not 3, but 4 cells and the maximum number of VIs would be $27^4 = 531.441$, and not the original $27^3 = 19.683$ with a validation cell.

Element	Value
Number of cells	3
Number of graphic units per cell	3
Number of different icons	3
Number of possible icon sequences	27
Maximum number of VIs to generate	19.683
Maximum number of VIs to generate w/o validation cell	531.441

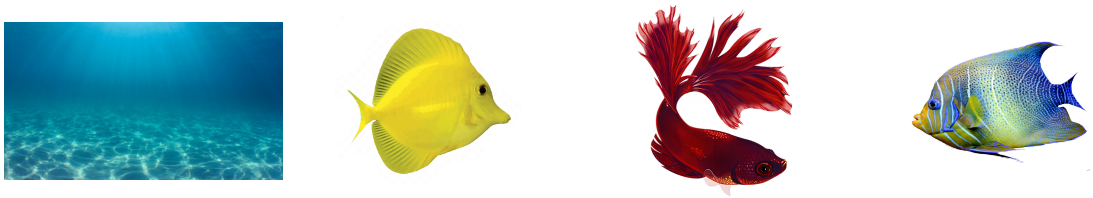
Table 4.2: Analysis of Example 2.

4.2.3 Example 3

Another example was created by using following icons and base image (figure 4.11).

This example (figure 4.11) encodes a 5 characters message. It has 5 cells (plus the validation cell) with 2 graphic units each. It also has the exact same number of icons in the code as the example before, which is another way to demonstrate that hardly any of the code elements can be deduced from only analysing the final code image image.

However, one element that does not have great importance in the encoding process but may



(a) Base image. **(b)** Icon 1. **(c)** Icon 2. **(d)** Icon 3.

Figure 4.11: Input images.

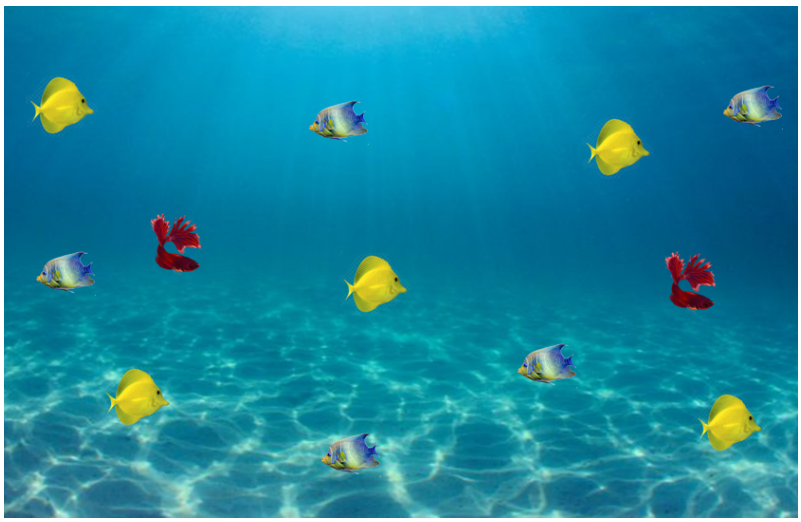


Figure 4.12: Final code example.

greatly influence the decoding process is the base image. This image has no apparent role in the code, except for aesthetic purposes, however, as the mandatory elements present in the first example serve as position detection, the base image can serve much the same purpose, since, in this case, the 4 corners of the blue background image are always in the same position in comparison to the icons.

4.2.4 Example 4

To show the versatility strand of the developed tool, we can also use simpler, cleaner elements, such as the following in figure 4.13. With the use of the developed program there are quite a lot of degrees of freedom and not many limits on the type of visual identities that can be created, so the designer will have a lot of flexible options to work with.

Comparing this example with the first one, this one has a smaller number of total icons in the code (14 to the 16 present in example one). Even so, with only one additional input icon (4 input icons to the 3 in example one) we can obtain more than 6 times the maximum number of VIs we are able to generate with this configuration. This means that a higher number of icons in a code

Element	Value
Number of cells	5
Number of graphic units per cell	2
Number of different icons	3
Number of possible icon sequences	9
Maximum number of VIs to generate	59.049
Maximum number of VIs to generate w/o validation cell	531.441

Table 4.3: Analysis of Example 3.

(a) Base image. **(b)** Icon 1. **(c)** Icon 2. **(d)** Icon 3. **(e)** Icon 4.

Figure 4.13: Input images.

does not imply that more codes can be created with a certain configuration.

4.2.5 Comparisons Between the Examples

In the table below (4.5) all of the above example's values are shown for comparison's sake.

A few conclusions we can take from this small sample of values are that the increase of the number of icons by one, boosts the maximum number of possible VIs, or that a smaller number of graphic units per cell does not mean an unavoidable decrease on the maximum number of VIs.

However, to further analyse the actual impact of these elements in the quantities of icon sequences or maximum number of VIs, a greater sample would have to be considered. Even so, this kind of study would fall outside the scope of this project.

4.3 Use-Case for Industrial Design

To demonstrate the functional use of this thesis' developed tool, a use-case of an industrial process will be described and illustrated in figures 4.15 and 4.16.

In the following case, let's assume that a designer of a certain company, that is a common target of product forgeries, wishes to create a product label for a quantity of about 1 million pieces.

The goal label would have to encode a 5 character message to identify each product, with letters from A-F and numbers from 0-9, which, with 6 letters and 10 numbers, equals the amount of

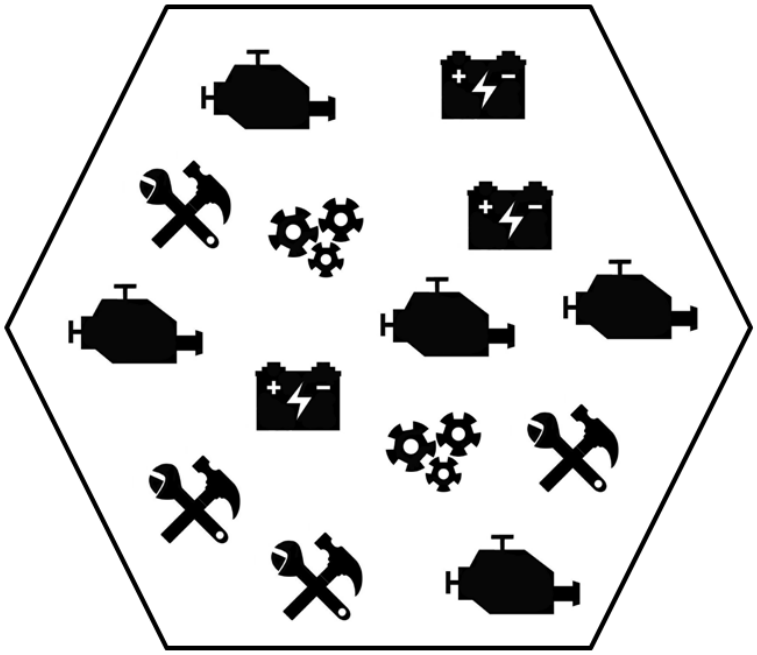


Figure 4.14: Final code example.

Element	Value
Number of cells	7
Number of graphic units per cell	2
Number of different icons	4
Number of possible icon sequences	16
Maximum number of VIs to generate	268.435.456
Maximum number of VIs to generate w/o validation cell	4.294.967.296

Table 4.4: Analysis of Example 4.

1.048.576 possible combinations. Therefore, this number represents the maximum number of VIs that we are able to create with the given requirements.

At that point, the designer would use the provided tool and input a message file with the 1 million messages. By reading the file, the program would recognise the number of cells as 5. Thereafter, the designer chooses, for example, 3 as the number of graphic units per cell, which would make the program automatically limit the minimum number of accepted icons to 3, since equation 3.1 has to be complied with. That is, since we need 1 million codes, which, distributed by the 5 cells implies that there needs to be at least 26 different sequences ($25^5 < 10^6$ and $26^5 > 10^6$) and, those sequences distributed in 3 graphic units per cell imply at least 3 icons ($2^3 < 26$ icon sequences and $3^3 > 26$ icon sequences).

Code Element	Example 1	Example 2	Example 3	Example 4
Number of cells	4	3	5	7
Number of graphic units per cell	4	3	2	2
Number of different icons	3	3	3	4
Number of possible icon sequences	81	27	9	16
Maximum number of VIs to generate	43.046.721	19.683	59.049	268.435.456
Maximum number of VIs to generate w/o validation cell	–	531.441	531.441	4.294.967.296

Table 4.5: Comparisons of the previous examples.

Afterwards, if the designer chose 3 as the number of icons, he/she would, then, have to load the 3 icons and a base image, and chose the placement of the icons to complete the creation of the configuration file.

The next stage would be the creation of the dictionary file, then the designer would be informed about how many VIs this configuration could generate (1.048.576) and, finally, the visual identities creation. This last step would generate as many images as the number of messages in the input file and would be followed by the creation of a PDF file with all the codes, in order to make the industrialisation process easier.

Finally, the designer would only have to deliver the PDF file to the printing department in the company, and the process would be finished.

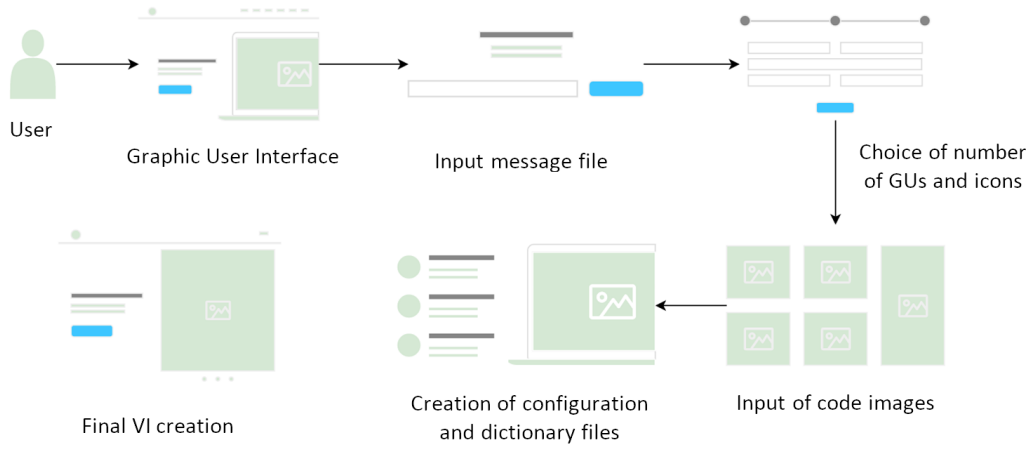


Figure 4.15: Industrial design use-case UML example.

In conclusion, the completion of this process could be done without the used of the developed tool. However, the process would take a lot longer, not only to make sure all the calculations were correctly done, but also to create each one of the needed 1 million labels. Besides that, all the code elements were properly documented and the needed files for the decoding process, configuration and dictionary file, were automatically created.

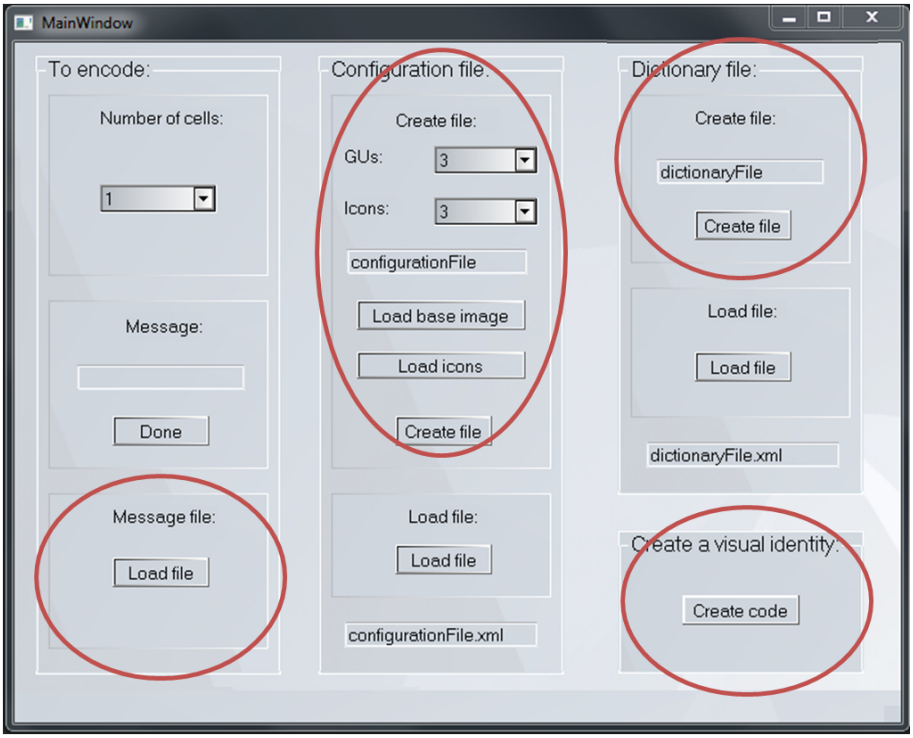


Figure 4.16: Industrial design use-case GUI example.

Therefore, by using this product, the company was able save considerable time and, consequently, money while trying to implement a way to confirm product authenticity and track their products in an aesthetically pleasing and marketable way .

5

Conclusions and Future Work

The goal of this dissertation was to formalise the creation process of an icon-based cryptographic algorithm, in a way that enabled the development of a standalone framework with the purpose of creating icon-based MRCs.

The formalisation of the creation process included the definition of the code inputs and the respective information quantification, while the creation of the framework focused on developing the corresponding algorithms and implementing them, so as to allow easy and documented icon-based GC creation.

While the implemented algorithms provided ample security for the codes due to its cryptographic nature, they also added a validation feature that would allow future code decryption to verify itself, therefore creating a more robust and secure process.

Consequently, the proposed MVP was developed and the goals were achieved with the improvement of the graphic user interface creation, which allowed a more intuitive and appealing creation process and workflow.

Although the initial MVP comprised only the above goals, a few other improvements were made in order to add further usability to the program, such as the ability to input a text file with several messages (instead of a single one) to create multiple codes at once, therefore allowing a more commercial use of this program.

However, improvements on software such as the developed one are almost limitless, since there are several updates that can be made to the already existing algorithms and a multitude of features that can be added. Some of those features could be the following: implementation of a process that compared the colours and shapes of the input icons to ensure they are different; development of a way for the user to re-position, rotate and resize the already positioned icons; and, finally,

development of an application with the already implemented features, but independent from both QtCreator and OpenCV in a way that it could be more easily used by whoever needed it. Besides the improvements on the developed program, the decryption process can also be explored, as explained before in this dissertation, with the creation of a software that perform this action and, in turn, tells the vendor or the consumer if the code corresponds to a original product or a forgery.

6

Appendix

Software Documentation

The following appendix is a part of the resulting documentation, generated with Doxygen, of the program created over the course of this dissertation.

Firstly, a review and descriptions of the overall program are presented , including software requirements and suggested testing. Afterwards a few classes and structures are explained.

User Interface platform to encode information into icon-based visual identities

Overview

This program's goal is to allow an easy generation of icon-based Graphic Code visual identities. An icon-based graphic code is composed by a certain number of graphic units, divided by a certain number of cells (which includes, in this case, a validation cell, used for code authentication). These graphic units are icons, that the user will choose to load, and will be placed in a base image, also loaded by the user.

Requirements

```
To use this program you need to have:
    -> OpenCV installed (v.4.2.0) for image processing
    -> Qt Creator (v.4.12.1) as an IDE
```

High-level architecture

The main method of the program creates configuration files, dictionary files and icon-based codes.

The main flow of the code is divided in 3 work-flows:

1. The user inputs information to encode
2. Then he can choose whether to create or load an existing configuration file Create Configuration File: needs GUs and icons number, loaded base image and icons
3. Afterwards, the same process is applied to the dictionary file creation (create or load)
4. Finally, with all the precious steps completed, the user can create a VI for the final created code

Function Descriptions

All functions are described in detail on the documentation pages for the files.

User Interface

The main user interface is developed in QT's GUI tool and presents the user with intuitive functionalities. The user also may interact with an OpenCV window when placing the code icons.

Testing

Test the main code by running the program on Qt Creator. The configuration and dictionary functions are the most prone to fail, so those should be tested first.

Besides those, the OpenCV functions are dependent on external libraries so those should also be tested by trying to create a new configuration file. While testing, make sure that the images are loaded properly, with the correct paths, the XML files are correct and parsed properly and the code elements are saved properly.

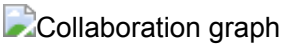
Finally, test for wrong user inputs to make sure the program handles them correctly.

MainWindow Class Reference

Inheritance diagram for MainWindow:

[legend]

Collaboration diagram for MainWindow:



[legend]

Public Member Functions

MainWindow (QWidget *parent=nullptr)

~MainWindow ()

Constructor & Destructor Documentation

◆ MainWindow()

MainWindow::MainWindow (QWidget * parent = nullptr)

Main Window constructor

◆ ~MainWindow()

MainWindow::~MainWindow ( )

Main Window destructor

The documentation for this class was generated from the following files:

- Icon-based_GC_Encoder/**mainwindow.h**
- Icon-based_GC_Encoder/mainwindow.cpp

config Struct Reference

`#include <config.h>`

Public Attributes

const char *	**fileName**
int	**nCells** = 0
int	**nIcons** = 0
int	**nGUs** = 0
int *	**positionsX** = 0
int *	**positionsY** = 0
const char **	**iPaths**

Detailed Description

Contains all the information needed for the creation of the .xml configuration file; this file must have the chosen number of cells, graphic units per cell (GUs) and different icons, as well as the positioning of every single graphic unit and the paths for all the images used.

The documentation for this struct was generated from the following file:

- Icon-based_GC_Encoder/**config.h**

dict Struct Reference

`#include <dict.h>`

Public Attributes

const char *	**fileName**
int	**nElemsMax** = 0

Detailed Description

Contains all the information needed for the creation of the .xml dictionary file; this file must have the name of the corresponding configuration file, the maximum number of elements that can be generated, and the existing alphabet (icons sequence and the corresponding value).

The documentation for this struct was generated from the following file:

- Icon-based_GC_Encoder/**dict.h**

Bibliography

[1] Visual paradigm tool, *https://online.visual-paradigm.com/pt/diagrams/*, 2020.

[2] Nevo Alva, Uriel Peled, and Itamar Friedman. Visual lead, *http://www.visualead.com/*, 2020.

[3] Hung-Kuo Chu, Chia-Sheng Chang, Ruen-Rone Lee, and Niloy J. Mitra. Halftone qr codes. *ACM Transactions on Graphics*, 32(6):1–8, 2013.

[4] Russ Cox. Q art code, *http://research.swtch.com/qart*, 2020.

[5] Gonzalo Garateguy, Gonzalo Arce, Daniel Lau, and Ofelia Villarreal. Qr images: Optimized image embedding in qr codes. *IEEE Transactions on Image Processing*, 23:2842–2853, 2014.

[6] Venkatesan Guruswami. *List Decoding of Error-Correcting Codes*. PhD thesis, 2002.

[7] Joseph Kirtland. *Identification Numbers and Check Digit Schemes*. The Mathematical Association of America, 2001.

[8] Neal Koblitz. *A Course in Number Theory and Cryptography*. Springer, 2nd edition, 1994.

[9] Li Li, Jinxia Qiu, Jianfeng Lu, and Chin-Chen Chang. An aesthetic qr code solution based on error correction mechanism. *The Journal of Systems and Software*, 116:85–94, 2016.

[10] Yu-Hsun Lin, Yu-Pei Chang, and Ja-Ling Wu. Appearance-based qr code beautifier. *IEEE Transactions on Multimedia*, 15(8):2198–2207, 2013.

[11] Bin Liu, Ralph R. Martin, and Shi-Min Hu. Structure aware visual cryptography. *Computer Graphics*, 33(7):141–150, 2014.

[12] Altan Mesut and Aydin Carus. Issdc: Digram coding based lossless data compression algorithm. *Computing and Informatics*, 29(5):741–756, 2014.

[13] Mark Nelson and Jean-Loup Gailly. *The Data Compression Book*. M&T Books, 1995.

[14] Bruno Patrão, Leandro Cruz, and Nuno Gonçalves. Graphic code: A new reliable machine readable system for coding (technical report).

[15] Bruno Patrão, Leandro Cruz, and Nuno Gonçalves. An application of a halftone pattern coding in augmented reality. *SA '17 Posters*, 2017.

[16] Bruno Patrão, Leandro Cruz, and Nuno Gonçalves. Graphic code: a new machine readable approach. *International Conference on Artificial Intelligence and Virtual Reality (AIVR)*, 2018.

[17] Bruno Patrão, Leandro Cruz, and Nuno Gonçalves. Graphic code: Creation, detection and recognition. *Recpad 2018-24th Portuguese Conference on Pattern Recognition*, 2018.

[18] Bruno Patrão, Leandro Cruz, and Nuno Gonçalves. Halftone pattern: A new steganographic approach. *Eurographics 2018*, 2018.

[19] Bruno Patrão, Leandro Cruz, and Nuno Gonçalves. Large scale information marker coding for augmented reality using graphic code. *International Conference on Artificial Intelligence and Virtual Reality (AIVR)*, 2018.

[20] Siyuan Qiao, Xiaoxin Fang, Bin Sheng, Wen Wu, and Enhua Wu. Structure-aware qr code abstraction. the visual computer. 31(6–8):1123–1133, 2015.

[21] Pooja Rai, Sandeep Gurung, and M. K. Ghose. Analysis of image steganography techniques: A survey. *International Journal of Computer Applications*, 114(1):11–17, 2007.

[22] Douglas R. Stinson. *Cryptography: Theory and Practice*. Chapman & Hall/CRC, 3rd edition, 2006.

[23] Robert Ulichney. *Digital Halftoning*. MIT Press, 1987.

[24] GS1 US. How gs1 standards support product tracing, critical tracking events and key data elements. 2011.

[25] Serge Vaudenay. *A Classical Introduction to Cryptography: Applications for Communication Security*. Springer, 2006.

[26] Denso Wave. Qr code essentials. 2011.

[27] Zhi Zhou, Gonzalo R. Arce, and Giovanni Di Crescenzo. Halftone visual cryptography. *IEEE Transactions on Image Processing*, 15(8):2441–2453, 2006.