Sistema de Protecció del Copyright

MIQUEL CONESA i PELLICER
Estudiant de l'Enginyeria Tècnica en Informàtica de Sistemes i
Autor del Treball

ANTONI MARTINEZ BALLESTE
Consultor Universitat Oberta de Catalunya i
Tutor del Treball

18 de Juny de 2004

I. Agraïments

A totes aquelles persones que donen part del seu temps en benefici de la comunitat, amb especial menció als qui, amb el seu propi exemple, encoratgen i lideren la resta com –en el cas concret de la indústria del software– Richard Stallman.

A Apple Corporation i Atari, pel seu lideratge i tossuderia en confiar què els ordinadors personals disposarien d'entorns gràfics de treball, que s'imposarien finalment a les finestres DOS de la resta de competidors de l'època.

A Microsoft Corporation, per posar a la disposició de tothom el sistema MSDN de documentació i suport al desenvolupador –pels seus sistemes, és clar–.

A Google Inc, per tornar a demostrar a tothom que l'èxit rau en les idees senzilles, ...i una gran dosi de treball pacient al darrera.

II. Dedicatòria

Dedico aquesta senzilla obra a Margarita Flores qui, amb la seva paciència, força de voluntat i nervi, m'ha encoratjat en tot moment a no rendir-me davant els reptes per complexos que semblin. Marga, tota aquesta feina ha estat possible gràcies a tu.

III. Resum

La facilitat amb què la tecnologia actual, a l'abast de tothom, permet obtenir còpies exactes d'imatges digitals esdevé, tanmateix, un problema per a la protecció dels legítims drets d'autor.

Per conèixer l'abast del problema, que també afecta altres produccions multimèdia, només cal recòrrer als informes de la *Sociedad General de Autores de España* [SGAE-BOL], on s'expresa en termes de *pirateria*. Aquest projecte presenta, desenvolupa i analitza un mètode que pot facilitar a un organisme que vetlla pels drets dels autors, com la SGAE, la persecució d'aquells que adquereixen una imatge d'autor i en fan un ús inapropiat.

El mètode insereix una marca a la imatge, de manera que és imperceptible al comprador. Si aquest comprador, o algú a qui la cedeix, publica la imatge sense satisfer els drets d'autor i la SGAE ho detecta, el mètode pot invertir el procés i identificar el comprador de la imatge. Amb aquesta informació la SGAE podrà exercir els drets i efectuar el requeriment a la persona adecuada.

De la mateixa manera que és fàcil obtenir còpies digitals d'una imatge, també és senzill i assequible efectuar-ne alteracions. D'entre l'enorme diversitat d'alteracions i distorsions possibles hem triat el subconjunt més habitual, per a les nostres proves, que han revelat per exemple, que el mètode és robust enfront les proves d'escalatge i a les confabulacions de compradors.

IV. Index de continguts

I. Agraïments 0
II. Dedicatòria 0
III. Resum 0
IV. Index de continguts 0
V. Índex de figures 0
Capítol 1. Introducció 0
1.1. Justificació del treball i context en el qual es desenvolupa: punt de partida i aportació 0
1.2. Objectius del TFC 0
1.3. Enfocament i mètode seguit 0
1.4. Planificació del projecte 0
1.5. Productes obtinguts 0
1.6. Descripció dels altres capítols de la memòria 0
Capítol 2. Especificació del procés de marcatge 0
1.1. Actors 0
1.2. Procediments 0
1.1.1. Procés de generació de la marca 0
1.1.2. Procés de recuperació de la identitat 0
1.1.3. Procés de confabulació de compradors 0
Capítol 3. Implementació del software 0
1.3. Descripció del format BITMAP 0
1.4. Divisió en quadres del bitmap 0
1.5. Inserció de la marca 0
1.6. Recuperació de la marca 0
1.7. Confabulació de compradors 0
Capítol 4. Manual d'usuari 0
1.1. Producte pel gestor dels drets de les imatges 0
1.1.4. Instal·lació del programa 0
1.1.5. Funcionament del producte 0
1.2. Producte dels confabuladors 0
1.1.1. Instal·lació del programa 0
1.1.2. Funcionament del producte 0
Capítol 5. Joc de proves 0
1.3. Proves de confabulació 0
1.4. Proves de reducció de mida 0
1.5. Proves d'ampliació de mida 0
1.6. Proves de compressió 0
Capítol 6. Conclusions 0
Capítol 7. Glossari 0
Capítol 8. Bibliografia 0
Annex 1. Codi font dels programes 0
1.1. Programa que encasta la marca a la imatge 0
1.2. Programa que simula una possible confabulació de compradors 0
1.3. Llibreria auxiliar de tractament d'imatges *bitmap* 0
Annex 2. Edició i compilació dels programes 0
1.1. Estructura del projecte 0
1.2. Distribució del desenvolupament al disc 0
Annex 3. Informació sobre les aplicacions auxiliars 0

V. Índex de figures

Imatges

Imatge 1: Diagrama de GANTT del projecte 0
Imatge 2: Format intern d'un bitmap 0
Imatge 3: Representació del significat dels bytes d'un bitmap 0
Imatge 4: Inserció cíclica de la marca en un quadre 10x10 del bitmap 0
Imatge 5: Comparació ascii de dos bitmap, per comprovar la inserció de la marca 0
Imatge 6: Exemple de finestra DOS 0
Imatge 7: Exemple d'ús del producte de generació d'una imatge protegida 0
Imatge 8: Exemple d'ús del producte per recuperar la identitat del comprador d'una imatge protegida 0
Imatge 9: Exemple de finestra DOS 0
Imatge 10: Exemple d'una confabulació 0
Imatge 11: Directori amb les imatges generades per a les proves 0
Imatge 12: Distribució del desenvolupament en directoris al disc 0

Taules

Taula 1: Definició i temporalització de tasques 0
Taula 2: Taula d'errors que es poden donar en el procés de generació d'una imatge protegida 0
Taula 3: Taula d'errors que es poden donar en el procés de confabulació 0
Taula 4: Resultat de les proves de confabulació 0
Taula 5: Resultat de les proves de reducció de mida 0
Taula 6: Resultat de les proves d'ampliació de mida 0
Taula 7: Resultat de les proves de compressió 0
Taula 8: Taula de referències bibliogràfiques 0
Taula 9: Taula d'aplicacions auxiliars 0

Diagrames

Diagrama 1: Casos d'ús 0
Diagrama 2: Activitat d'inserció de la marca a la imatge 0
Diagrama 3: Activitat de recuperació de la marca a la imatge 0
Diagrama 4: Activitat de confabulació d'imatges entre compradors 0

Capítol 1. Introducció

1.1. Justificació del treball i context en el qual es desenvolupa: punt de partida i aportació.

Aquest treball es justifica en la mesura que cada dia és més senzill copiar i reproduir qualsevol obra. Avui dia la informàtica ens ofereixes moltes eines apropiades per a digitalitzar qualsevol contingut. Un cop el contingut es presenta en format digital, només cal usar el conegut "copiar i enganxar".

Altrament, existeixen les anomenades xarxes P2P com Kazaa [KAZAA] o eMule [EMULE], on milers d'usuaris intercanvien personalment tot tipus d'arxius de contingut audiovisual. Diversos exemples en són:

- els audiovisuals (imatge, so i video)
- els literaris (llibres, articles, ...)

El què moltes persones desconeixen o obvien és que, pel sól fet de tenir una obra en format digital com un arxiu a l'ordinador propi, no en són propietaris. La legislació vigent, la Llei de Propietat Intel·lectual [LPROINT], reconeix a l'autor d'una obra uns drets sobre la mateixa. Entre els drets, té el de cedir-ne el seu ús a canvi d'una compensació. Per exemple, quan comprem un disc no comprem la música, sino el dret de reproduïrla, per a ús personal. I qualsevol altre ús queda expressament desautoritzat, ja que generalment va en contra dels interessos de l'autor. Per a protegir aquests interessos s'han creat associacions d'autors, com la S.G.A.E., que vetllen per la protecció dels drets i el cobrament de les compensacions degudes als mateixos.

En fer duplicacions no autoritzades de material amb drets d'autor s'infringeix la llei. El cas és que és molt difícil de controlar, ja que la pròpia tecnologia existent ni limita les accions sobre el material ni reconeix els drets d'autor. D'altra banda hi ha força pressió a Internet en favor de la *pirateria* i cada cop hi ha més usuaris a les xarxes P2P.

Diverses empreses privades han dedicat els seus esforços a dotar a la tecnologia capacitat per limitar la difusió de les obres (es poden destacar els notables esforços d'Adobe, per exemple, per la limitació d'accions en els documents PDF amb signatures [ADOBE]).

En el terreny de la imatge digital no hi ha cap suport per part de la tecnologia per a la protecció dels drets d'autor. Per aquesta raó s'ha proposat aquest treball, que és un possible complement de la tecnologia actual per a introduir en les imatges informació invisible que permeti, dins el què ès possible, la identificació de l'infractor.

Hi ha diversos mètodes d'inserció d'informació en una imatge. Tots ells tenen el mateix objectiu: identificar el comprador, tot i què hi ha diferències en la seva robustesa enfront a les distorsions més diverses que es poden imaginar.

En aquest treball es descriu i s'aplica un mètode de marca basat en una lleugera alteració dels components menys significatius del color de la imatge. Es treballa directament sobre els bits que composen la imatge digital.

Altres autors opten per aproximacions diferents a la mateixa tasca:

- Wavelet Conversion, adoptat per Jpeg2000 i usat per Canon [CANON]
- Alessandro Piva, Franco Bartolini, de la Universitat de Florència i Mauro Barni de la Universitat de Sien exposen, a un article aparegut a la publicació "IEEE Internet Computing" el Maig de 2002, un mètode matemàtic per inserir una marca a la imatge. Es basa en transformar la imatge al domini de la freqüència (procés DCT discrete cosine transform) i afegir un senyal codificat i diferents marques de sincronització. Els mateixos autors expliquen el mètode amb més detall en altres articles, com al "Interoperability Issues in Watermark-Based Electronic Copyright Management".
- L'empresa "Digital Copyright Technologies Ltd." usa un mètode semblant al descrit anteriorment en els seus sitemes de protecció, avalats per la Universitat de Gènova i l'Escola Politècnica Federal de Laussane [DICOTEL].
- Joseph J. K. Ruanaidh i Shelby Pereira, de la Universitat de Gènova, van publicar el treball "A Secure Robust Digital Image Watermark" [EIPPC98], on analitzen amb tot detall el mètode d'inserció de marques basat en el DCT. El complementen amb una infraestructura de claus públiques i privades per afegir un component de signatura electrònica, molt més eficient que el codi que nosaltres apliquem.

Vista la bibliografia descrita, que es troba a Internet, el mètode triomfador i que resol de manera més eficient el problema que es planteja, fins i tot comercialment, és el mètode basat en DCT.

Aquesta tecnologia es basa en l'aplicació de mètodes matemàtics avançats (transformades del cosinus i de Fourier), que es complementen amb lleis esencials de la física de les ones (interferència de la llum), nocions del comportament de l'òrgan visual de l'home i d'aspectes psicològics de la percepció cerebral. Es tracta d'un camp apassionant i multidisciplinar, però queda fora de l'abast d'aquest treball.

1.2. Objectius del TFC.

L'objectiu d'aquest treball és realitzar i avaluar un sistema de protecció del copyright mitjançant detecció de còpia amb utilització de marques d'aigua. El producte resultant és un programa capaç d'enterrar la identitat d'un comprador en una imatge determinada. Es treballarà amb imatges de tipus bitmap, que no perden qualitat un cop es comprimeixen. Les funcionalitats bàsiques del sistema són les següents:

El venedor ha de poder inserir una marca (una tira de bits) de forma invisible dins una imatge BMP. Per a inserir una marca, cal modificar lleugerament el valor d'alguns pixels. S'intentarà que el fet d'alterar lleugerament aquesta imatge marcada no ha d'impedir recuperar la marca (això s'anomena robustesa del sistema de marca d'aigua). La idea és dissenyar un senzill mètode de marcatge i analitzar la seva robustesa.

La marca serà una paraula d'uns codis especials. Aquests codis s'usaran com a marca quan un comprador adquireixi certa imatge, de manera que si el venedor recupera la marca de certa imatge es pugui conèixer qui n'és el comprador. Es demana que els codis usats resisteixin confabulacions de dos compradors.

Dos compradors de la mateixa imatge disposen de dues còpies que difereixen, justament, en alguns dels bits que formen part de la marca que els identifica. Canviant el valor d'aquests bits poden generar nua nova còpia que no identifiqui cap dels compradors, o bé que identifiqui un comprador innocent. Hi ha codis que resisteixen confabulacions: malgrat esborrar o alterar part de l'identificador, se segueix coneixent la identitat d'algun dels confabulats.

1.3. Enfocament i mètode seguit.

Es tracta d'un treball de recerca, on es tracta de desenvolupar un programa que aplica un mètode de marcatge. Un cop obtingut el programa s'ha d'aplicar un joc de proves exhaustiu per tal de demostrar la seva robustesa enfront diversos tipus d'accions que poden portar a terme els compradors. L'objectiu sempre serà recuperar-ne la seva identitat.

El mètode seguit per al desenvolupament del projecte ha estat el mètode estructurat. S'ha de dissenyar un producte d'ús senzill, sense interficie d'usuari i amb unes dades d'entrada i sortida molt ben definides. Altres mètodes, com els orientats a objectes introduirien complexitat addicional que en aquest cas no és necessària.

1.4. Planificació del projecte.

Per a la realització del projecte s'han definit un conjunt de tasques, i una temporalització, que s'exposen en el quadre següent:

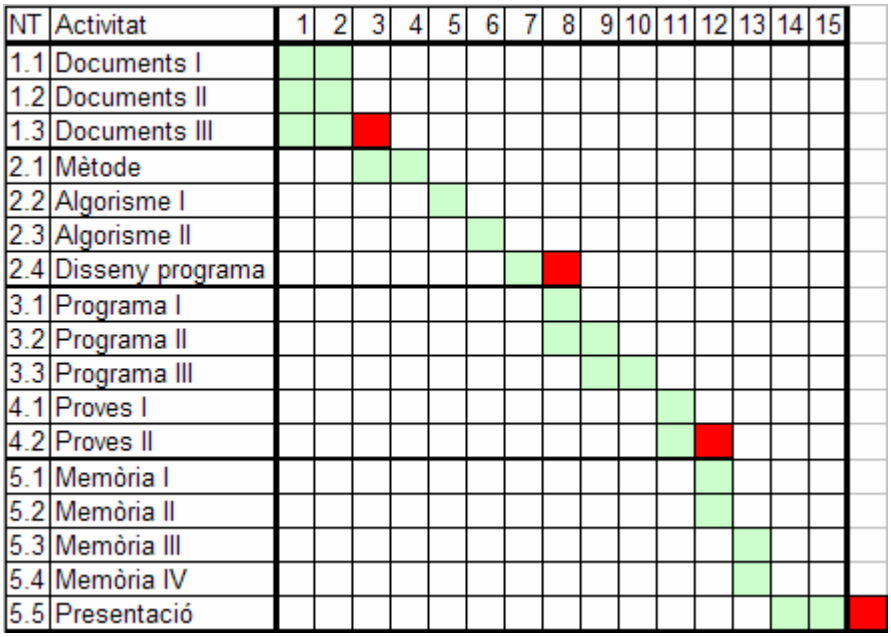
Tasca	Tempo (setmana)	Descripció de l'activitat
1.1	1 i 2	Cerca d'informació sobre el format d'imatge BMP.
1.2	1 i 2	Cerca d'informació sobre protecció del copyright amb detecció de còpia usant tècniques de marca d'aigua.
1.3	1 i 2	Cerca d'informació sobre codis de Hamming duals i el seu ús contra confabulacions.
2.1	3 i 4	Dissenyar un senzill mètode de marca d'aigua.
2.2	5	Dissenyar l'algorisme que permetrà encastar una marca que identifiqui un comprador. Caldrà pensar en usar un petit fitxer de dades.
2.3	6	Dissenyar l'algorisme que permetrà llegir la marca d'un BMP i obtenir la identitat del comprador. Caldrà pensar en usar un petit fitxer de dades.
2.4	7	Dissenyar un programa que, donades dues imatges, en generi una a partir de la confabulació dels dos compradors
3.1	8	Implementació del mètode de marca d'aigua i prova sobre imatges
3.2	8 i 9	Implementació del programa d'encast / extracció de marca
3.3	9 i 10	Implementació del programa de confabulacions.
4.1	11	Fer jocs de proves sobre robustesa del sistema. Bàsicament s'aplicaran vàries transformacions sobre una imatge marcada i es comprovarà que se n'extreu la marca correctament. Com es poden millorar els resultats?
4.2	11	Fer jocs de proves sobre la resistència a confabulacions de 2 usuaris. A partir de dues còpies de la mateixa imatge venudes a dos compradors, generareu amb la vostra eina una imatge nova. Passant aquesta imatge al programa que extreu la marca i retorna la identitat del comprador, haurem d'obtenir la identitat d'un dels confabulats. Com es poden millorar els resultats?
5.1	12	Elaboració de la memòria: estat de l'art
5.2	12	Elaboració de la memòria: disseny del programa i algorismes.
5.3	13	Elaboració de la memòria: breu manual d'usuari de l'aplicació.
5.4	13	Elaboració de la memòria: apèndix amb codi font, degudament comentat.
5.5	14 i 15	Fer presentació del projecte, amb PowerPoint o altres aplicacions

Taula 1: Definició i temporalització de tasques

Com es pot apreciar, s'ha dividit el treball en quatre grans etapes:

- Recerca d'informació,
- Disseny del mètode,
- Implementació i proves,
- Memòria

Totes les activitats i la seva temporalitat s'exposen al següent diagrama de GANTT:



Imatge 1: Diagrama de GANTT del projecte

Cada tasca agrupa un conjunt d'activitats, que s'han de realitzar en la setmana que es marca amb color verd, tot seguint la previssió.

En vermell s'indiquen els lliuraments que s'han de realitzar durant el transcurs del projecte. Es tracta de fites en les que s'han d'haver resolt les activitats programades segons el calendari.

1.5. Productes obtinguts

El producte que es presenta amb aquesta memòria és un programa que permet:

- crear una imatge marcada, donat un comprador i una imatge original,
- guardar una referència de tots els compradors, les marques i les imatges,
- recuperar la identitat d'un comprador donada una imatge.

Tanmateix, es presenta un programa addicional per a generar jocs d'imatges confabulades al·leatòriament a partir de dues imatges marcades de diferents propietaris.

1.6. Descripció dels altres capítols de la memòria

Els capítols següents de la memòria descriuen amb detall les activitats realitzades i els objectius assolits:

Especificació del procés de marcatge

Aquest capítol recull l'especificació de requisits: defineix funcionalment tot el procés, des de la compra de la imatge i la generació d'una marca per a cada comprador, fins a la recerca i extracció de la marca d'una imatge per tal de descobrir la identitat del comprador.

Descriu addicionalment el procés de confabulació entre usuaris i proposa una possible solució per a la seva implementació.

Implementació del software

Aquest capítol exposa els fonaments en què s'ha basat l'elaboració del software, algorismes i decisions de disseny que han portat a la creació del producte que s'entrega.

Addicionalment incorpora els manuals d'usuari de les aplicacions, amb exemples del seu funcionament.

Jocs de proves

Aquest capítol informa de les proves que s'han efectuat i recull tots els resultats obtinguts, amb explicacions detallades dels problemes apareguts en la fase de test i les solucions aportades. Addicionalment s'esmenten altres efectes distorsionadors de la imatge que no s'han inclós en el joc de proves.

Capítol 2. Especificació del procés de marcatge

El context del nostre projecte està format per diversos actors i procediments. Tots ells es mostren en el diagrama de casos d'ús següent:

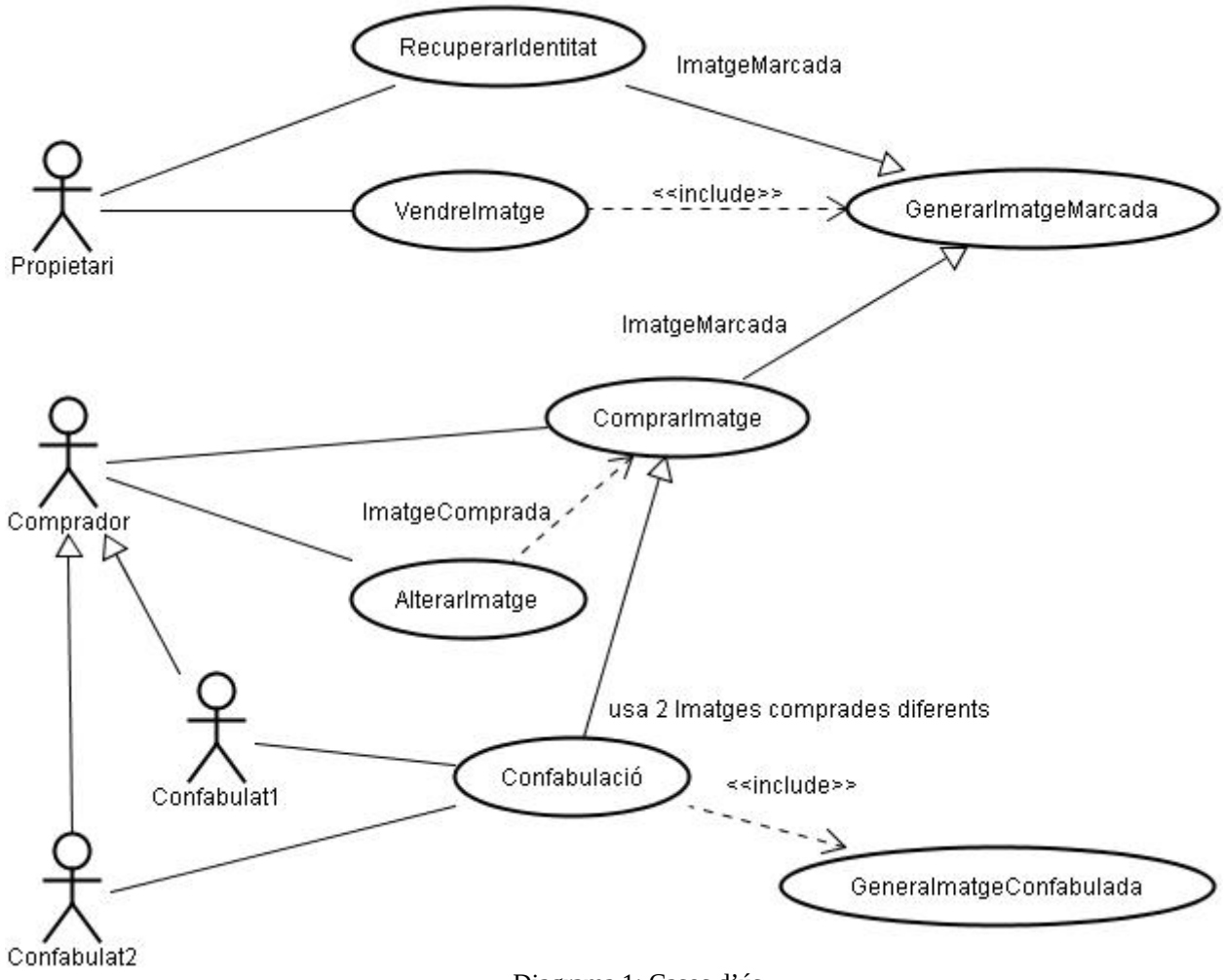


Diagrama 1: Casos d'ús

1.1. Actors

Tenim dos tipus d'actors identificats:

- Propietaris dels drets de les imatges i
- Compradors, on hi ha el tipus Confabulat que n'és una especialització.

Els primers poden Vendre Imatges, que inclou el procés de Generació d'Imatges Marcades (un primer programa, el resultat del qual és una Imatge Marcada) i poden Recuperar la Identitat a partir d'una Imatge Marcada.

Els segons poden Comprar Imatges (obtingudes en el procés de Venda ja descrit) i fer-ne l'ús personal que desitgin (que no s'indica en el diagrama).

De cara al mal ús que se'n pugui fer, hem afegit altres processos que Alteren la Imatge Comprada.

Dos Compradors, especialitzats en Confabulacions, poden generar una nova Imatge Confabulada, que queda fóra del procés de marcatge, a partir de les dues imatges marcades. El que no saben els confabulats és que el nostre sistema els enxamparà igualment.

1.2. Procediments

Els procediments indicats anteriorment ens porten a resoldre alguns processos mitjançant programes. Aquests processos són:

- GenerarImatgeMarcada, a partir de la imatge original
- RecuperarIdentitat, a partir d'una imatge marcada i de la imatge original
- GenerarImatgeConfabulada, a partir de dues imatges marcades

Aquests processos es descriuen a continuació:

1.1.1. Procés de generació de la marca

El procés de generació de la marca es descriu pel següent diagrama d'activitat:

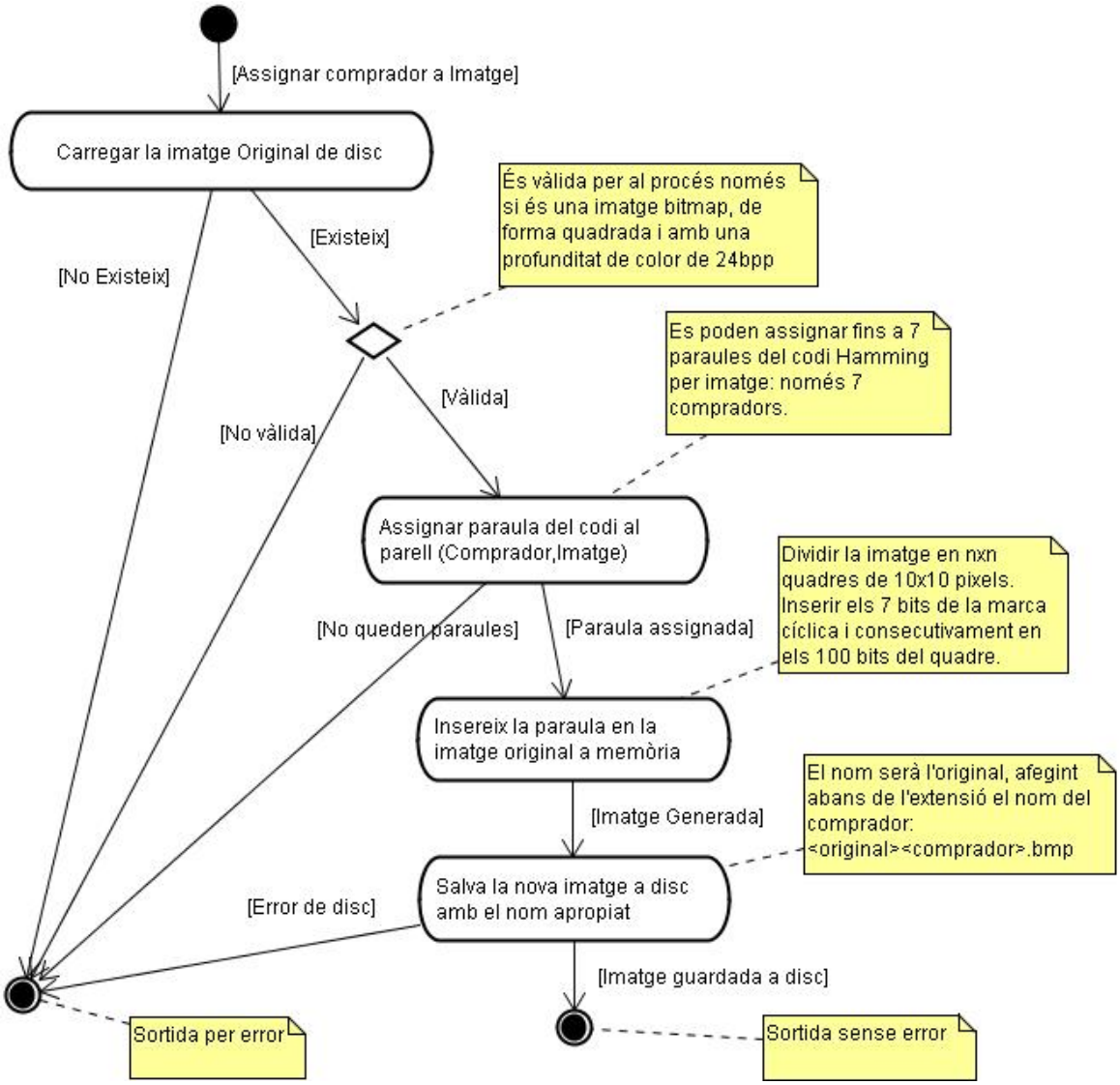


Diagrama 2: Activitat d'inserció de la marca a la imatge

Segons el diagrama, s'ha de guardar un arxiu amb les paraules del codi assignades per a cada imatge. Això ho farem amb un arxiu d'aplicació, de text i editable (arxiu INI). Aquest arxiu s'haurà de crear al mateix lloc on hi hagi l'executable (per tenir-lo sota control).

Existeixen dos camins de sortida: amb error i sense. L'error es senyalitza amb un codi de retorn que identifica de manera única la causa de l'error. En cas de sortida sense error, el codi de retorn serà zero.

Es poden donar altres errors no especificats (d'inicialització, paràmetres incorrectes, etc...), que també s'hauran de tenir en compte.

1.1.2. Procés de recuperació de la identitat

El procés de recuperació de la identitat es descriu pel següent diagrama d'activitat:

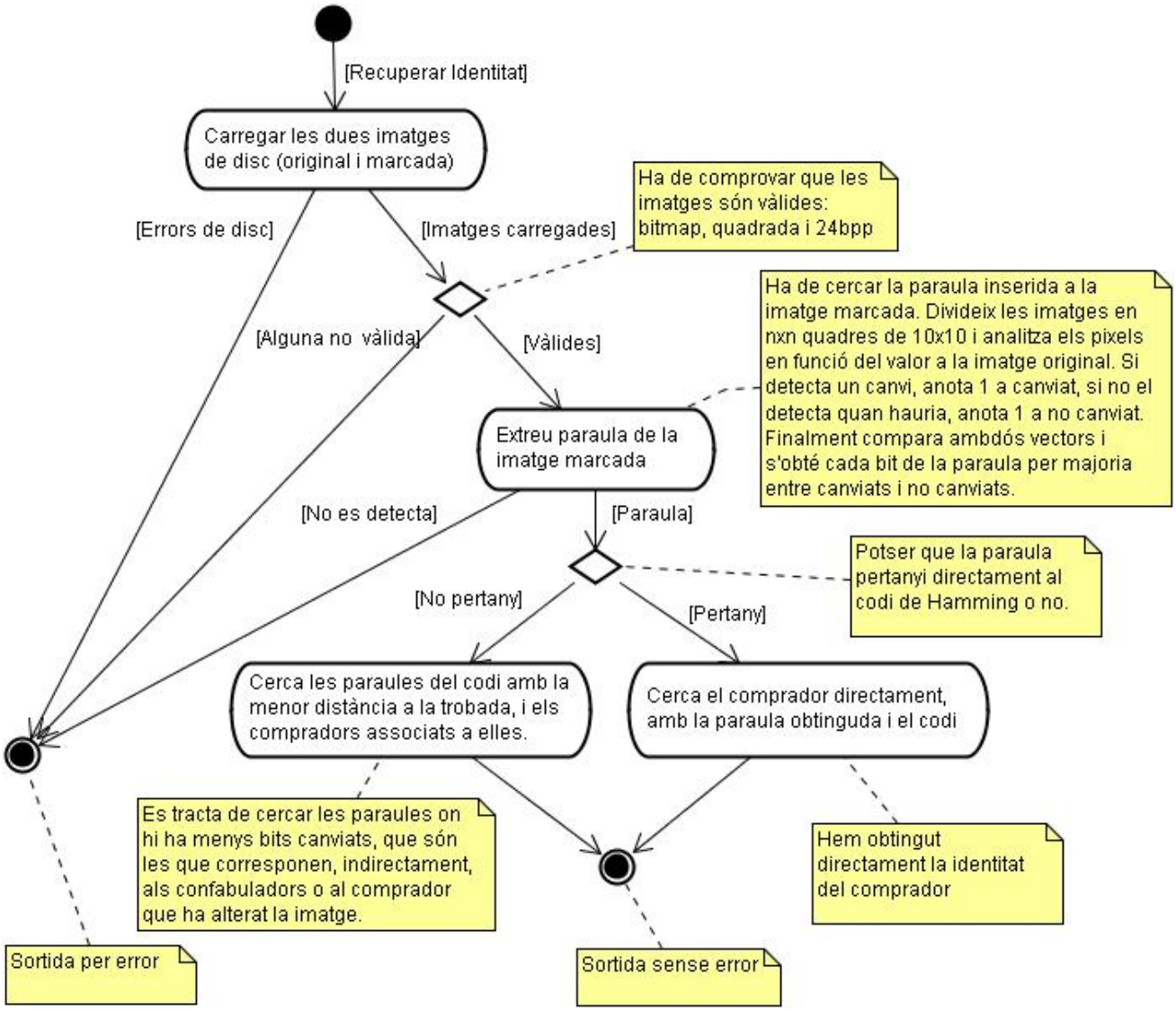


Diagrama 3: Activitat de recuperació de la marca a la imatge

Els dos subprocessos més importants en aquest cas són el de "extreure paraula" i el de "cercar paraules del codi amb la menor distància". El primer ha d'analitzar els pixels de la imatge (els tres bits de color) i afegir un 1 al bit actual que s'analitza del vector de Uns només si detecta, en els bits menys significatius, dos o tres bits canviats respecte de la imatge original (hi ha informació de marcatge). En cas contrari, afegeix un 1 al vector de Zeros. En acabar aquest subprocés, la paraula resultant es trobarà bit a bit comparant quin és el valor major, dels dos vectors per a la posició analitzada. Si Uns[i] > Zeros[i] aleshores considerem que paraula[i] és un 1, sino considerem que és un 0.

Si la paraula obtinguda pertany al codi de Hamming, aleshores és una imatge original i sabem immediatament qui n'és el comprador. És un encert directe.

Si la paraula obtinguda no pertany al codi aleshores la imatge pot haver estat fruit de degradació: degut a alteracions geomètriques, compressió, conversió a b/n, confabulació, distorsió, o altres tècniques més complexes.

Aleshores analitzem quines paraules del codi tenen la menor distància possible amb la paraula obtinguda. Això és possible comptant el nombre de bits que varien entre totes les combinacions del codi Hamming i la paraula obtinguda. Primer es cerca quina és la distància mínima (quin és el mínim nombre de bits que canvien entre la paraula i totes les altres paraules del codi, triades d'una en una). Un cop tenim la distància mínima, mostrem la llista de compradors associats a les paraules del codi que tenen exactament aquesta distància mínima amb la paraula trobada. En aquest cas, no hi ha un encert directe, però hi pot haver un ó més encerts indirectes.

Cal remarcar que el subprocés d'extracció de la paraula de la imatge pot no detectar cap marca. Això és possible, per exemple, en el cas de subministrar al programa dues còpies de la imatge original. En el nostre procés de generació de marca el codi '0000000' queda reservat a la imatge original i, pertant, identifica una imatge sense marcar.

1.1.3. Procés de confabulació de compradors

El procés de confabulació de compradors es descriu en el següent diagrama d'activitat:

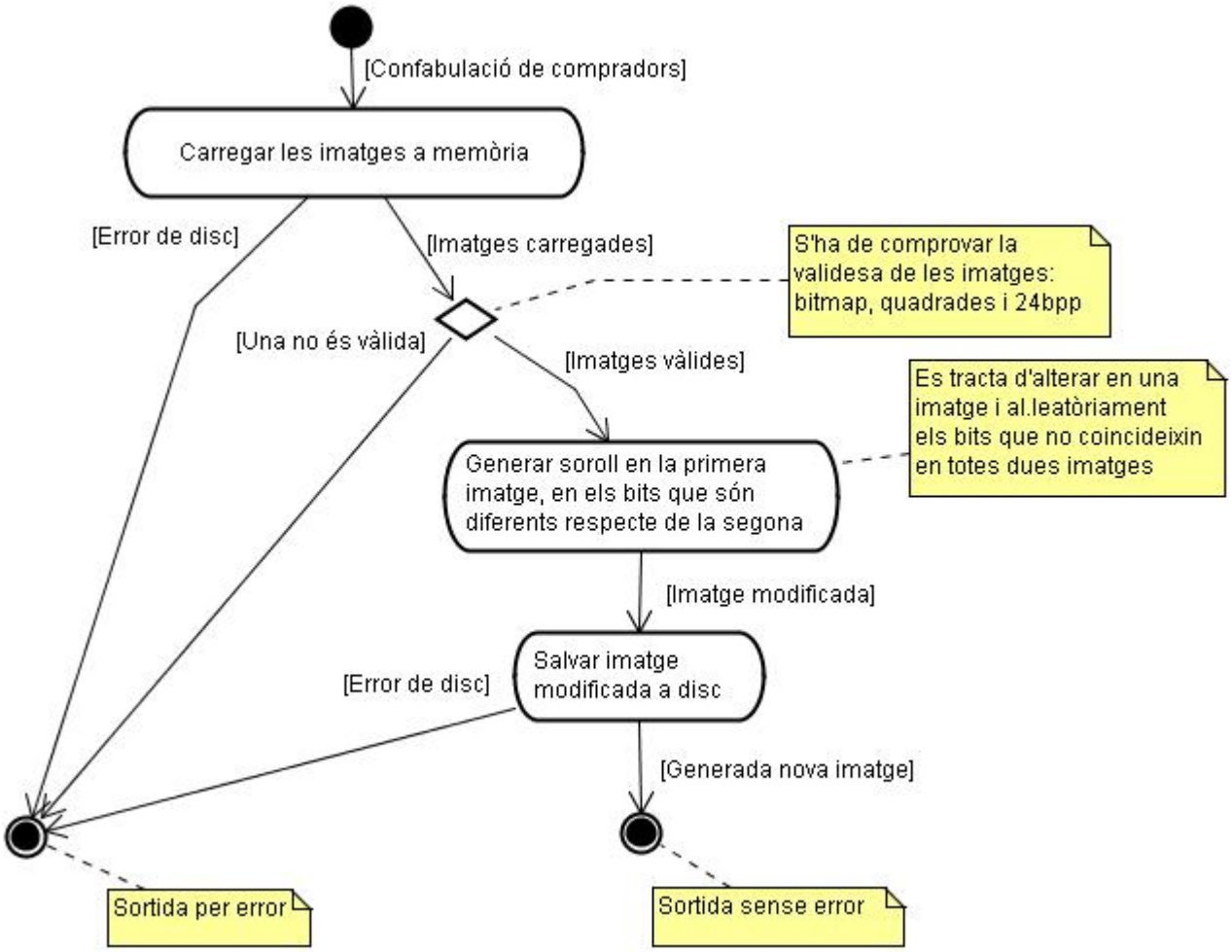


Diagrama 4: Activitat de confabulació d'imatges entre compradors

El subprocés més significatiu és el de generació de soroll en els bits que canvien entre totes dues imatges (on, segons pensen els confabuladors, segurament es guarda la informació de la marca).

Com que ja tenim un algorisme que fa el recorregut de la imatge, el podem aprofitar, sempre i quan assegurem que els bits amb diferències s'alteren de manera al·leatòria. Amb aquesta premisa al cap, qualsevol algorisme possible obtindria el mateix resultat, així que ens podem estalviar el disseny i la implementació d'un nou algorisme per recòrrer els bits de la imatge.

Capítol 3. Implementació del software

Els tres processos que es descriuen funcionalment en el capítol anterior s'han dividit en dues categories:

- activitats que realitza el propietari de la imatge i
- activitats que realitza el comprador

En el cas de les activitats que realitza el propietari (generar marca i recuperar identitat) es desitja que es trobin en un únic programa executable.

Òbviament, el comprador podrà fer l'ús que vulgui de la imatge i (disposa d'eines de tractament d'imatge, com les que s'indiquen en la taula d'aplicacions auxiliars) també es podrà confabular amb un altre comprador.

Les dues aplicacions executables es realitzaran sota les següents condicions:

- funcionament sota qualsevol sistema operatiu Windows de 32 bits (98, NT, XP, ME i 2000)
- funcionament en linia de comandes, dins una finestra DOS del sistema operatiu
- pas de la informació a l'aplicació a través de la linia de comandes

L'eina de desenvolupament triada és el Microsoft Visual C 6.0 pels següents motius:

- coneixement del llenguatge i de l'entorn de treball, per part de l'autor
- llibreries, aportades pel fabricant, que simplifiquen les funcions bàsiques per treballar amb el format bitmap
- facilitat del llenguatge per recòrrer la memòria i efectuar-ne modificacions

Així que el projecte s'enfoca inicialment, des del Visual C, amb un únic workspace i dos projectes. El workspace s'anomenarà "marca" i els projectes "marca" i "confabula". Els projectes es creen en mode consola de 32 bit i amb ús de les classes MFC (framework de Microsoft). El projecte i el codi font s'aporta a la memòria en un arxiu ZIP adjunt.

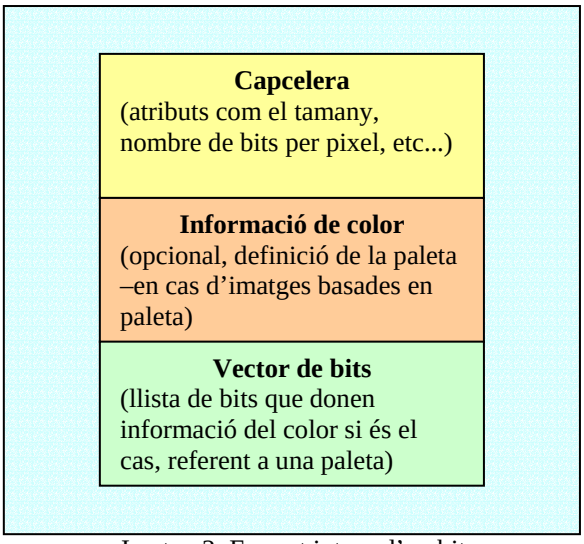
Pel que fa al desenvolupament del software, les quatre activitats més complexes han estat:

- Comprendre l'estructura interna del format bitmap
- Elaborar un mecanisme de divisió en quadres del bitmap i moure's a dins
- Inserir la marca i verificar que realment la imatge queda modificada
- Recuperar la marca i comprovar que s'encerta la identitat

1.3. Descripció del format BITMAP

El format bitmap es descriu en diversos articles que es citen a la bibliografia, en concret [BITMAP], [BITMAP2], [BITMAP3] i [BITMAP4].

Un resum senzill i entenidor del format bitmap es pot veure en el gràfic següent:



Imatge 2: Format intern d'un bitmap

Els *bitmap* amb els que treballem tenen una estructura més simple, ja que la informació de color no es troba en una estructura de paleta, sino en el propi punt, que indexa un de 2^24 possibles colors amb 3 bytes d'ocupació.

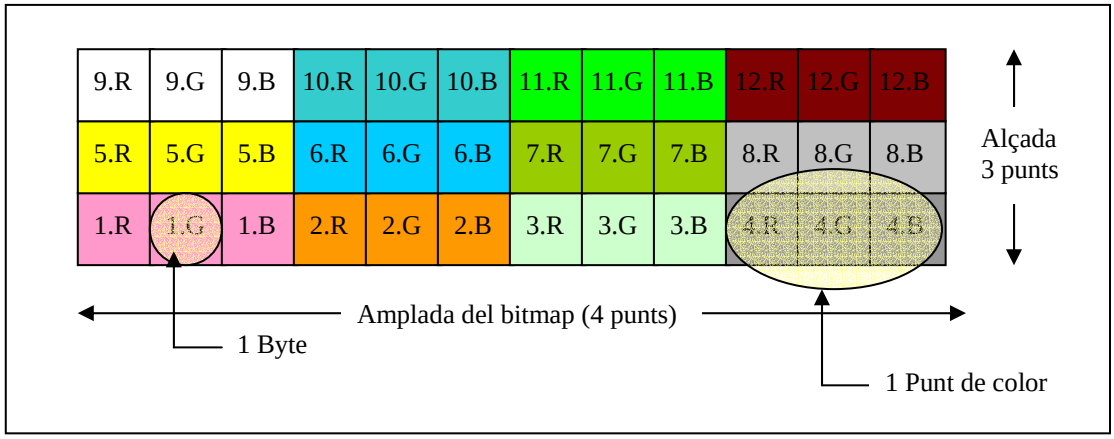
Per mouren's dintre del bitmap tindrem dos tipus de funcions, les que accedeixen als atributs de la capcelera (obtenen característiques relacionades amb la geometria i volum del mapa de bits) i les que accedeixen al vector de bits, que recuperen el gràfic en memòria i permeten modificar-ne les parts.

El bitmap es representa generalment cap per avall. és a dir, els tres primers bytes del vector representen el punt de sota a l'esquerra. A mesura que llegim els punts següents ens movem en la mateixa linia cap a la dreta fins arribar a l'amplada del mapa. En aquest punt, el següent grup de tres bytes és el de l'esquerra de la linia de sobre de la que hem llegit.

És a dir llegim el bitmap per linies, d'esquerra a dreta i des de baix cap a dalt.

1.4. Divisió en quadres del bitmap

En el cas que ens ocupa, els bitmap tenen 24 bits per pixel, això vol dir que cada punt de color queda representat per tres bytes, un per cada component (vermell, verd, blau). Si llegim el vector de bits, l'hem d'entendre de la manera següent:



Imatge 3: Representació del significat dels bytes d'un bitmap

Per recòrrer el bitmap usem bucles que divideixen els bytes en linies, en funció de l'amplada del bitmap, les linies en grups de tres bytes (s'agrupa per punt de color o *pixel*) i les fileres en funció de l'alçada de la caixa.

En el codi font s'utilitzen diversos punters per accedir directament als components de color del bitmap. Aquests punters s'incrementen en funció dels comptadors dels bucles, permetent així adreçar qualsevol dels bytes que el composen.

1.5. Inserció de la marca

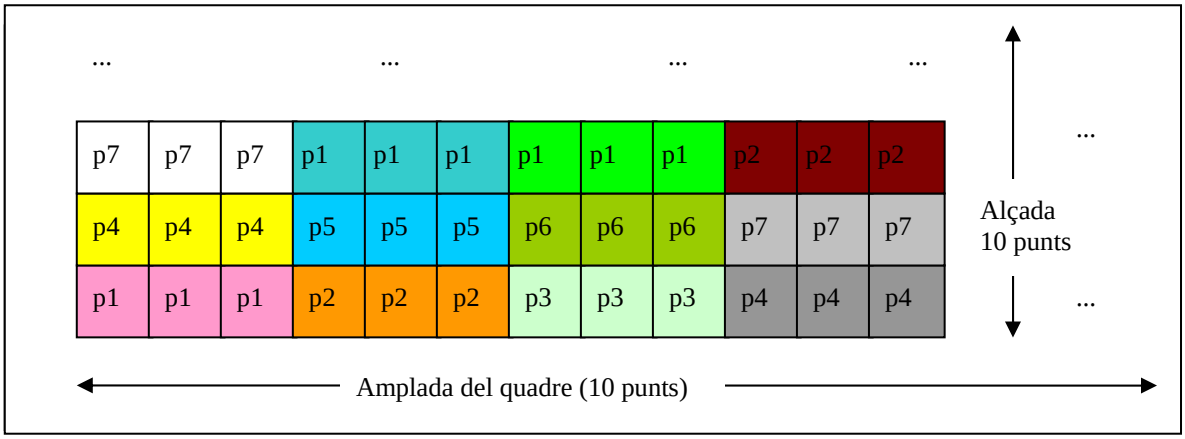
Un cop coneixem la paraula de 7 bits que hem d'inserir en la imatge i la geometria de la imatge (amplada * alçada), ja podem efectuar el procés d'inserció.

El procés d'inserció divideix la imatge en n quadres, on n = amplada / 10. La divisió és entera, fet que pot deixar de marcar alguns quadres en cas que l'amplada no sigui múltiple de 10 (recordem que es tracta d'imatges quadrades).

Cadascun d'aquests quadres es divideix en 10 x 10 pixels. Cadascun d'aquests pixels (de 3 bytes) porta informació de color. Els bits menys significatius (bms) de cada byte porten informació potencialment modificable sense que sigui apreciable el canvi. Pertant, podem recòrrer els 10 pixels i el vector de la paraula, bit a bit, fent la comprovació següent:

- Si el bit de la paraula és 1, invertir els bms dels 3 bytes del pixel
- En cas contrari, no fer res

Podem repetir aquesta comprovació més enllà del 7è bit, retrocedint al primer bit de la paraula, de forma cíclica. Així, marcariem els 10 bits amb la següent seqüència:



Imatge 4: Inserció cíclica de la marca en un quadre 10x10 del bitmap

Tot i que en la imatge només es representen els quadtre primers bits de les tres primeres fileres, es pot apreciar que a la posició 10 (1er byte de la 2ª filera) s'insereix el 4art bit de la marca (als tres darrers punts de la filera anterior els correspon els valors ordinals 8, 9 i 10, que s'han associat als bits 1,2 i 3 de la marca).

Es pot verificar la alteració dels punts de color amb una utilitat com el WinDiff (eina que incorpora el Visual C), que permet visualitzar les diferències entre arxius.

Per exemple, entre una imatge no marcada i una marcada, el WinDiff ens mostra la següent informació:

Imatge 5: Comparació ascii de dos bitmap, per comprovar la inserció de la marca

Com es pot apreciar a la primera linia vermella i a la primera linia groga (no cal mirar més enllà), els sis primers bytes són iguals (2 bits zero seguits de la marca), 3 bytes diferents (un bit a un), 6 bytes més coincidents (2 zeros), 3 bytes diferents (un un), … Així veuriem quina marca hi ha inserida, en el ben entès que el primer bit que contem coincideixi amb el primer bit de la paraulamarca.

Aquesta diferència, però, no és perceptible. L'observació de totes dues imatges, des del Microsoft Paint, revela que ambdúes imatges són visiblement iguals. Aparentment no s'aprecia cap diferència entre la imatge original i la imatge marcada. Això significa que el procés de marcatge és robust, si més no, a l'observació de l'usuari (no ho seria, per exemple si s'hagués afegit a la imatge una banda amb informació sobre el copyright i, per exemple, un codi de barres).

1.6. Recuperació de la marca

Aquest és el procés més complexe del programa. No ho és per l'algorisme en sí, que s'ha especificat prèviament amb facilitat, sino per la capacitat de l'algorisme d'extreure informació de la imatge amb fiabilitat.

L'algorisme suposa que la imatge original és de la mateixa mida geomètricament que la imatge marcada. És per això que si la imatge marcada ha sofert algun tipus d'alteració en la seva mida, ha d'haver un procés previ d'adaptació (escalatge) de la imatge marcada. Aquest procés s'haurà d'efectuar amb un programa d'edició.

El primer problema que es presenta és: què estem comparant? Suposem que les dues imatges són iguals excepte en els bits de la marca (més o menys), però, ho hem de comprovar? En el cas que ens ocupa, la decissió de disseny ha estat comprovar que els continguts d'ambdúes imatges són similars.

Les votacions que s'efectuen en analitzar cada punt de la caixa 10x10 només s'anoten en els vectors de votació si els punts de les dues imatges difereixen com a molt en els 4 bits menys significatius de tots els bits de color. Una alteració superior del punt el faria visible i això seria comparar imatges diferents.

En el codi font es pot observar que, inicialment, el procés seguit en la funció d'extracció de la marca va ser d'acumulació de votacions de uns, però aquest mètode va demostrar la seva ineficàcia, ja que la meitat de la informació del codi es troba en els uns i l'altra meitat en els zeros. No podem despreciar cap de les dues informacions.

Pel que fa al procés en sí, l'algorisme no és eficient, ja que és fàcil perdre el sincronisme de la marca si:

- la imatge s'ha canviat de posició dins el quadre,
- s'han duplicat files o columnes, degut a una reducció i una ampliació posterior,
- s'han perdut files o columnes, degut a una ampliació i una reducció posterior.

Donada la naturalesa cíclica i a la distribució de la marca s'ha d'esperar que més tard o més d'hora es retrobarà el sincronisme en un altre quadret, a mesura que s'analitza. Les proves demostren que el nombre de votacions disminueix a mesura que augmenta la distorsió.

Es pot millorar: afegint senyals de sincronisme (o patrons de bits) a la imatge, de manera que cercant els patrons ens poguem adonar ràpidament de les distorsions que s'han produït en la imatge, i actuar en conseqüència.

Un cop trobada la marca, ens trobem amb un altre procés difícil: trobar-li identitat. Ens trobem concretament en dos casos diferents:

- la paraulamarca existeix en el codi Hamming original,
- la paraulamarca no existeix en el codi Hamming original,

El primer cas és trivial, es tracta d'una imatge no modificada, o que la modificació que ha sofert no ha suposat cap problema per a la marca. En aquest cas es troba directament la identitat del comprador en accedir a l'arxiu on hi són les dades del comprador i la imatge.

El segon cas ja no és trivial, ja que indica que la imatge ha estat alterada. En funció del grau d'alteració es podrà trobar o no la identitat del comprador.

La primera idea va ser resoldre el problema per 'aproximació': quina paraula de la marca se sembla més a la obtinguda? Qeust mètode no va tenir éxit, ja que no encertava els confabulats. Certament, el què es demanava és un mètode que fes la recerca de la paraula del codi amb distància mínima.

Cercar la distància mínima és un procés de dos pasos. En primer lloc es cerca efectivament la distancia mínima: es compara la paraula obtinguda amb totes les paraules del codi, reduint la variable "dMin" que hem inicialitzat a un valor per sobre de la distància màxima, que és 7. Un cop obtingut el valor de dMin, tornem al codi cercant aquelles paraules que tenen dMin amb la trobada. D'aquesta manera trobem la identitat d'una o més persones que poden haver alterat la imatge.

Aquest mètode té defectes, ja que en alguns casos a les proves ha demostrat que – de vegades enxampa els confabulats i algun innocent. També passa amb algunes alteracions, on apareix la identitat del comprador i la d'algun innocent.

Es pot millorar aquest comportament aprofitant la capacitat del codi per a detectar i corregir bits erronis. El codi de Hamming s'ha dissenyat amb l'objectiu d'aprofitar la redundància i distància entre combinacions per arribar a descobrir fins a "*d*" bits modificats i corregir-ne "*d-1*". Si el nostre algorisme aprofités aquesta característica, probablement eliminaria els innocents en la identificació d'identitats.

1.7. Confabulació de compradors

El procés de confabulació és molt fàcil d'obtenir a partir del procés d'extracció de la marca. Només es tracta de canviar la funció de recuperar marca.

En el moment en què en el procés d'extracció s'anava a comparar els bytes de cada *pixel*, ara s'ha de comprovar si es detecta una variació en els bits menys significatius.

- En cas afirmatiu, s'han de canviar per soroll
- En cas negatiu, s'ha de deixar igual.

Per introduir soroll a la imatge, s'inicialitza la seqüència al·leatòria amb el valor del comptador d'impulsos del sistema (TickCount), què el sistema operatiu actualitza en funció de les interrupcions que es produeixen per *hardware*. Un cop assegurem la seqüència de valors al·leatoris, quan s'han de canviar els bits de color obtenim un numero al·leatori. D'aquest nombre només ens interesa el bit menys significatiu (si és parell o imparell). Afegit el nostre bit al bit menys significatiu de la imatge amb una operació "or-exclusiva" (degut a la naturalesa al·leatòria del nombre, es podria haver triat qualsevol funció lògica, en realitat).

En finalitzar el procés, s'obté una imatge visualment igual que les dues originals, però significativament diferent a nivell de bits (si s'observa, per exemple, amb el WinDiff).

En qualsevol cas, la modificació no és perfecta i per això s'enxampa els confabulats. Com que la distància minima entre els codis de totes dues imatges és de 4 bits, sempre hi ha (com a mínim) tres bits de les dues combinacions, en posicions fixes, que conserven i coincideixen en els valors del codi original. Aquests tres bits es transmeten a la imatge generada, a més del soroll.

El nostre sistema d'extració de marca detecta fàcilment els tres bits i el soroll només contribueix a "ajudar" en el procés de descobrir un dels culpables. Si el soroll confirma un confabulat, es detecta. Si el soroll no confirma cap confabulat, podem mostrar la llista d'identitats amb la distància minima que hem calculat.

Es pot millorar el procés, com ja s'ha dit abans, explotant les característiques de codi Hamming com a detector i corrector d'errors.

Capítol 4. Manual d'usuari

El present manual d'usuari descriu el procés d'instal·lació dels productes i el se funcionament extern, des del punt de vista de l'usuari que l'ha adquirit.

1.1. Producte pel gestor dels drets de les imatges

1.1.4. Instal·lació del programa

Els producte es composa d'un executable i un arxiu de configuració que s'han de copiar en una nova carpeta. Es recomana seguir el següent procés d'instal·lació:

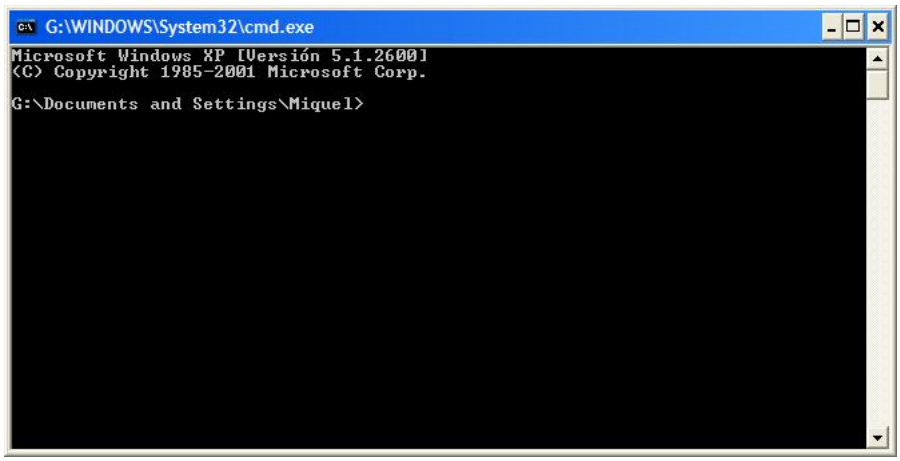
- Crear un directori al disc per al programa
  (per exemple, C:\GESTIODRETS)
- Crear un directori al disc per a les imatges venudes, que contindran informació relacionada amb el comprador
  (per exemple C:\ GESTIODRETS \IMATGESVENUDES)
- Crear un directori al disc per a les imatges originals, que s'endreçaran amb les imatges venudes
  (per exemple C:\GESTIODRETS\IMATGESORIGINALS)
- Copiar el programa MARCA.EXE i l'arxiu auxiliar de l'aplicació, subministrats per l'autor, al primer directori
  (C:\GESTIODRETS)

En acabar el procés d'instal·lació, l'usuari ja pot començar a treballar amb el programa.

1.1.5. Funcionament del producte

El producte funciona des d'una finestra DOS, per tant s'ha d'obrir primer una finestra DOS abans de continuar. Aquest procés es pot realitzar fàcilment des del "*Start menu > Programs > Accessories > Command Prompt*". En cas de no trobar aquesta opció (els nous sistemes cada cop l'amaguen més), sempre es pot executar des de la finestra "*Start menu > Run > escriure "cmd" i apretar el botó "OK"*"

Un cop oberta la finestra DOS, tindrem una imatge similar a la següent:



Imatge 6: Exemple de finestra DOS

Ara ens hem de desplaçar al directori on hem instal·lat prèviament l'aplicació, per exemple C:\GESTIODRETS. Per fer això, hem d'escriure la seqüència de comandes del sistema següents:

- C:
- CD \GESTIODRETS

Un cop ubicats al directori de l'aplicació, ja tenim accés al programa.

Ara hem de decidir quin dels dos modes de funcionament volem usar:

- creació d'una imatge apta per a la venda o
- recuperació d'identitat d'una imatge

Segons el mode de funcionament que triem, les comandes que s'han d'introduir són diferets.

Mode de creació d'una imatge apta per a la venda

En aquest cas, necessitem tenir a mà la informació següent:

- nom i ubicació de la imatge original
- nom del comprador (o DNI)

Des de la finestra de comandes oberta prèviament, on ja ens hem posicionat al directori on es troba l'aplicació, podem escriure la comanda següent:

marca <<ubicació i nom de la imatge>> m <<nom del comprador>>

En executar el programa, observarem que el producte ens dona informació sobre el procés.

Finalment, si no hi ha cap problema, el programa crearà la imatge en la mateixa ubicació on es troba la imatge original. És possible que durant el procés es generi un error. La taula d'errors es troba al final del capítol.

En la imatge següent es mostra una execució del programa seguint les indicacions anteriors:

```
G:\WINDOWS\System32\cmd.exe
Microsoft Windows XP [Versión 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

G:\Documents and Settings\Miquel>c:

C:\>cd GESTIODRETS

C:\GESTIODRETS>marca c:\palmeres.bmp m 46571865Q
Proces de marcatge d'una imatge original

Imatge original: c:\palmeres.bmp
Comprador       : 46571865Q
Carrega imatge  : OK
Geometria img.  : 400x400/24bpp
Codi assignat   : 0011011
Generacio marca: OK
Imatge generada: c:\palmeres.46571865Q.bmp

C:\GESTIODRETS>
```

Imatge 7: Exemple d'ús del producte de generació d'una imatge protegida

Ara es recomana copiar les imatges original i generada a les carpetes creades per emmagatzemar-les. D'aquesta manera creem un arxiu històric del que podrem crear còpies de seguretat fàcilment. Aquesta operació es pot fer des de l'explorador de carpetes de Windows:

- Cercar la imatge original (en el nostre cas c:\palmeres.bmp) i copiarla a la carpeta d'imatges originals (c:\gestiodrets\imatgesoriginals) i
- Cercar la imatge protegida (en el nostre cas c:\palmeres.46571865Q.bmp) i copiarla a la carpeta d'imatges venudes (c:\gestiodrets\imatgesvenudes).

Mode de recuperació de la identitat del comprador d'una imatge

En aquest cas, necessitem tenir a mà la informació següent:

- nom i ubicació de la imatge original
- nom i ubicació de la imatge venuda

Des d'una nova finestra de comandes oberta prèviament, on ja ens hem posicionat al directori on es troba l'aplicació, podem escriure la comanda següent:

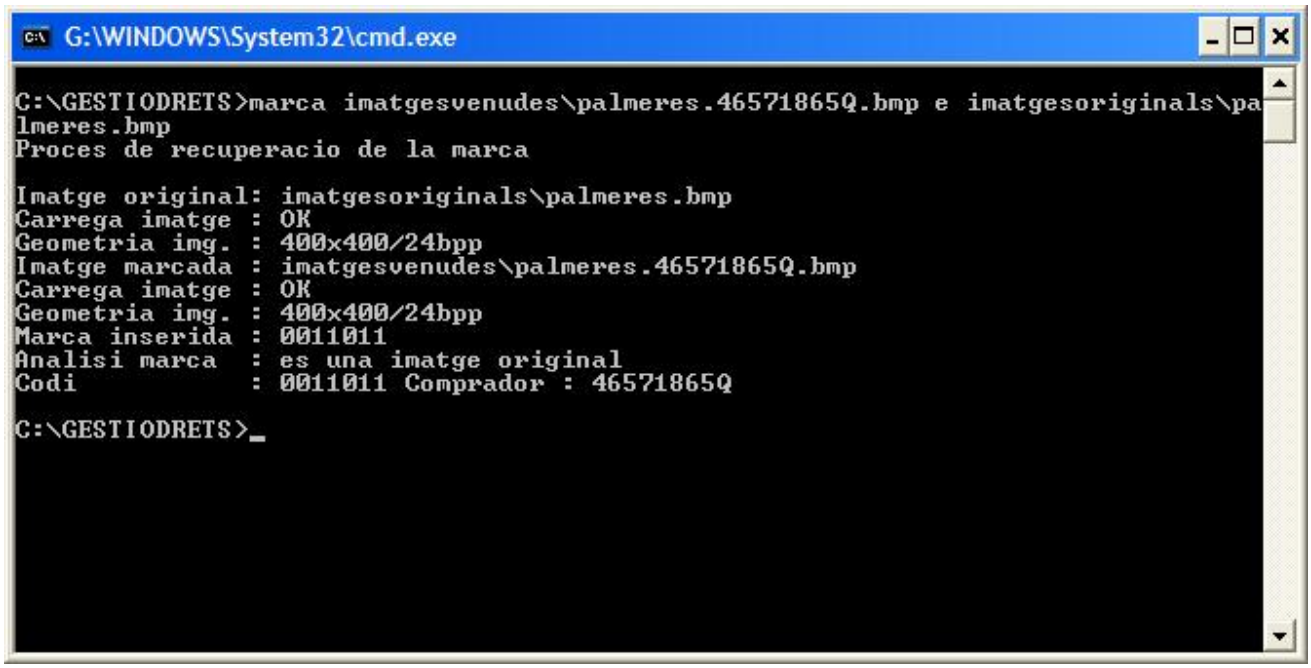
marca <<ubicació i nom de la imatge venuda>> e << ubicació i nom de la imatge original>>

En executar el programa, observarem que el producte ens dona informació sobre el procés.

Finalment, si no hi ha cap problema, el programa mostrarà un o diversos compradors de la imatge (segons la qualitat de la imatge original i l'abast de les alteracions realitzades, és possible que el programa no sigui capaç de trobar el comprador o en trobi més d'un).

És possible que durant el procés es generi un error. La taula d'errors es troba al final del capítol.

En la imatge següent es mostra una execució del programa seguint les indicacions anteriors:



Imatge 8: Exemple d'ús del producte per recuperar la identitat del comprador d'una imatge protegida

Taula d'errors que genera l'aplicació

En cas que l'aplicació detecti una excepció, funcionament incorrecte o l'incumpliment de les especificacions, emtrà un dels errors que s'enumeren a continuació:

Codi	Error	Descripció
1	Ha fallat l'inicialitzacio de les MFC	Error intern. El sistema operatiu no funciona correctament.
2	No s'ha pasat el nombre correcte de parametres	No s'ha introduït correctament la comanda, segons s'indica en aquest capítol.
3	No es pot obrir l'arxiu <<nom>>	L'arxiu no es troba en la ubicació que s'ha indicat. Si no s'indica cap, la ubicació per defecte és el directori del sistema des d'on s'executa l'aplicació que, segons les indicacions, és el directori on es troba el programa.
4	No s'ha trobat l'arxiu de parametres <<nom>>	L'arxiu de paràmetres s'ha de copiar segons les instruccions d'instal.lació al mateix direcori que l'executable marca.exe. És possible que aquest error apareixi si s'executa el programa des d'un directori diferent al que es troba instal·lat el programa.
5	L'accio especificada no es correcta (ha de ser m o e)	El paràmetre que identifica la operació està bén definit: M (marcar) ó E (extreure marca)
6	No queden codis d'aquesta imatge per assignar	Ja s'han venut les 7 còpies possibles per la imatge que té aquest nom. S'haurà de canviar de nom per a poder continuar la venda.
7	L'arxiu indicat no es una imatge BMP	S'ha facilitat un nom d'arxiu correcte, però no es tracta d'un mapa de bits.
8	No es pot guardar la imatge marcada en l'arxiu <<nom>>	Si es treballa amb poc espai al disc (com és el cas dels disquetts), és possible quedar-se sense espai. També es pot donar aquest error, en cas de fer servir unitats de xarxa, si hi ha problemes en la mateixa o en l'ordinador remot que s'usa.
9	L'arxiu indicat no conte una imatge quadrada	Error de validació: la geometria de la imatge no és apropiada.
10	L'arxiu indicat no conte una imatge de 24bpp	Error de validació: la imatge només es pot protegir si té una profunditat de color de 24 bits (truecolor).
11	No consten compradors de la imatge	Error de recuperació: no s'ha detectat informació del comprador en la imatge marcada.
12	No hi ha cap comprador de la imatge amb el codi <<codi>>	Error de recuperació: s'ha trobat un codi correcte, però n es troba el comprador en l'arxiu de paràmetres. Potser està usant un arxiu incorrecte o s'ha esborrat.
13	No es detecta cap marca en la imatge	Error de recuperació: No s'ha detectat cap inserció d'informació de comprador en la imatge.
14	No es pot adaptar la geometria de la imatge	Error de recuperació: Les imatges han de tenir la mateixa geometria, ja que el programa no fa adaptacions.

Taula 2: Taula d'errors que es poden donar en el procés de generació d'una imatge protegida

Errors addicionals:

S'ha detectat que sota determinades circumstàncies, les dades de les imatges marcades es guarden a l'arxiu "marcatge.ini", però que s'ubica al directori de Windows (c:\windows). En aquest cas, és necessari també que existeixi l'arxiu "marcatge.ini" al directori on es troba el programa "marca.exe"

1.2. Producte dels confabuladors

1.1.1. Instal·lació del programa

Els producte es composa d'un executable que s'ha de copiar en una nova carpeta. Es recomana seguir el següent procés d'instal·lació:

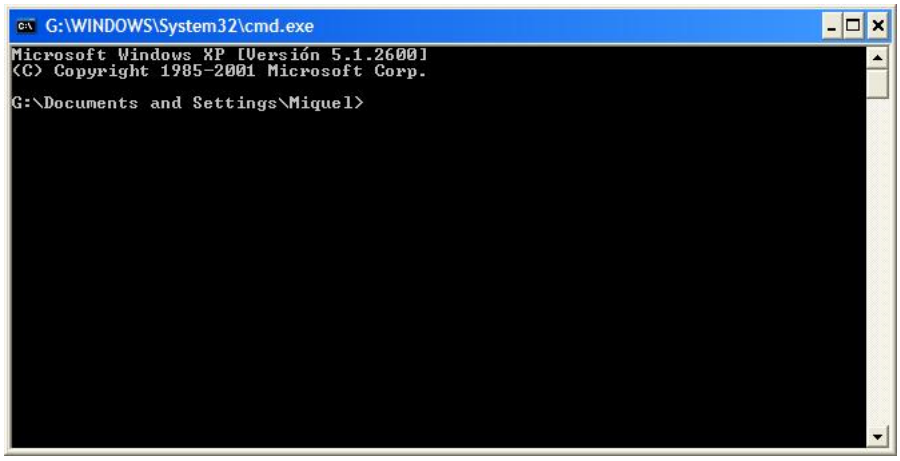
- Crear un directori al disc per al programa
  (per exemple, C:\CONFABULA)
- Copiar el programa CONFABULA.EXE a aquest directori
  (C:\CONFABULA)

En acabar el procés d'instal·lació, l'usuari ja pot començar a treballar amb el programa.

1.1.2. Funcionament del producte

El producte funciona des d'una finestra DOS, per tant s'ha d'obrir primer una finestra DOS abans de continuar. Aquest procés es pot realitzar fàcilment des del "*Start menu > Programs > Accessories > Command Prompt*". En cas de no trobar aquesta opció (els nous sistemes cada cop l'amaguen més), sempre es pot executar des de la finestra "*Start menu > Run > escriure "cmd" i apretar el botó "OK"*"

Un cop oberta la finestra DOS, tindrem una imatge similar a la següent:



Imatge 9: Exemple de finestra DOS

Ara ens hem de desplaçar al directori on hem instal·lat prèviament l'aplicació, per exemple C:\CONFABULA. Per fer això, hem d'escriure la seqüència de comandes del sistema següents:

- C:
- CD \CONFABULA

Un cop ubicats al directori de l'aplicació, ja tenim accés al programa.

En aquest cas, necessitem tenir a mà la informació següent:

- nom i ubicació de la imatge comprada pel primer confabulat
- nom i ubicació de la imatge comprada pel segon confabulat

Des d'una nova finestra de comandes oberta prèviament, on ja ens hem posicionat al directori on es troba l'aplicació, podem escriure la comanda següent:

confabula <<ubicació de la primera imatge >> << ubicació de la segona imatge >>

En executar el programa, observarem que el producte ens dona informació sobre el procés.

Finalment, si no hi ha cap problema, el programa generarà una nova imatge a partir de les dues subministrades, alterant al·leatòriament els bits d'informació que no són iguals entre ambdúes. El nom generat és el mateix de la primera imatge que es subministra, però afegint la paraula "conf" al final. En la imatge d'exemple s'aprecia clarament aquest fet.

En la imatge següent es mostra una execució del programa seguint les indicacions anteriors:

```
G:\WINDOWS\System32\cmd.exe

 Directorio de C:\CONFABULA

21/06/2004  09:47    <DIR>          .
21/06/2004  09:47    <DIR>          ..
07/06/2004  22:57            24.576 confabula.exe
21/06/2004  09:04           480.054 palmeres.46571865Q.bmp
21/06/2004  09:46           480.054 palmeres.47839241A.bmp
               3 archivos        984.684 bytes
               2 dirs   2.151.994.880 bytes libres

C:\CONFABULA>confabula palmeres.46571865Q.bmp palmeres.47839241A.bmp
Programa de confabulacio de dos compradors

Primera Imatge : palmeres.46571865Q.bmp
Carrega imatge : OK
Geometria img. : 400x400/24bpp
Segona Imatge  : palmeres.47839241A.bmp
Carrega imatge : OK
Geometria img. : 400x400/24bpp
Geometria      : OK
Confabulant Img: OK
Imatge generada: palmeres.46571865Q.conf.bmp

C:\CONFABULA>
```

Imatge 10: Exemple d'una confabulació

Els confabulats ja tenen a la seva disposició una imatge *piratejada*.

Taula d'errors que genera l'aplicació

En cas que l'aplicació detecti una excepció, funcionament incorrecte o l'incumpliment de les especificacions, emtrà un dels errors que s'enumeren a continuació:

Codi	Error	Descripció
1	Ha fallat l'inicialitzacio de les MFC	Error intern. El sistema operatiu no funciona correctament.
2	No s'ha pasat el nombre correcte de parametres	No s'ha introduït correctament la comanda, segons s'indica en aquest capítol.
3	No es pot obrir l'arxiu <<nom>>	L'arxiu no es troba en la ubicació que s'ha indicat. Si no s'indica cap, la ubicació per defecte és el directori del sistema des d'on s'executa l'aplicació que, segons les indicacions, és el directori on es troba el programa.
4	Les imatges no son de la mateixa mida	Ambdúes imatges han de tenir la mateixa geometria
7	L'arxiu indicat no es una imatge BMP	S'ha facilitat un nom d'arxiu correcte, però no es tracta d'un mapa de bits.
8	No es pot guardar la imatge marcada en l'arxiu <<nom>>	Si es treballa amb poc espai al disc (com és el cas dels disquetts), és possible quedar-se sense espai. També es pot donar aquest error, en cas de fer servir unitats de xarxa, si hi ha problemes en la mateixa o en l'ordinador remot que s'usa.
9	L'arxiu indicat no conte una imatge quadrada	Error de validació: la geometria de la imatge no és apropiada.
10	L'arxiu indicat no conte una imatge de 24bpp	Error de validació: la imatge només es pot protegir si té una profunditat de color de 24 bits (truecolor).

Taula 3: Taula d'errors que es poden donar en el procés de confabulació

Capítol 5. Joc de proves

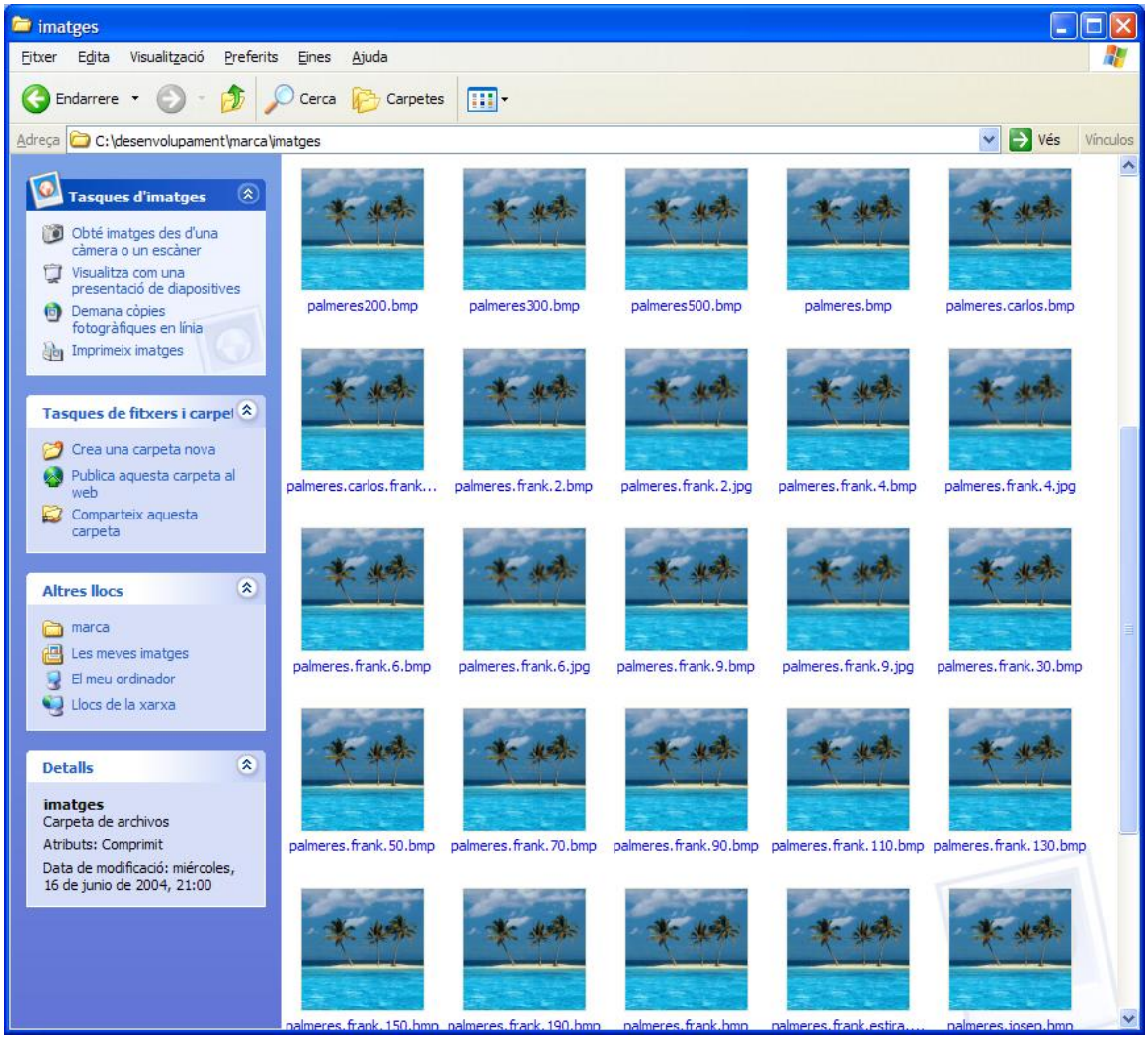
S'ha realitzat un joc de proves, que ha consistit en un seguit d'operacions que s'indica a continuació:

- Creació d'una imatge quadrada de 400x400 punts i 24bpp (palmeres.bmp)
- Creació de set imatges marcades, amb noms diferents
- Creació d'imatges confabulades
- Recuperació de les identitats de les imatges confabulades, respecte els originals
- Reducció de les imatges a diferents mides, gravació, ampliació a la mida normal i gravació
- Recuperació de la marca de les imatges inicialment reduïdes
- Ampliació de les imatges a diferents mides, gravació, reducció a la mida normal i gravació
- Recuperació de la marca de les imatges inicialment ampliades
- Compressió de la imatge a JPEG, amb diferents nivells de qualitat, descompressió i gravació.
- Recuperació de la marca de les imatges inicialment comprimides

No s'han realitzat proves sobre altres mecanismes d'alteració de les imatges, com:

- Reducció de color
- Traslació de parts de la imatge
- Blur, crop, wind i altres efectes gràfics

Es pot observar la generació d'imatges realitzada:



Imatge 11: Directori amb les imatges generades per a les proves

1.3. Proves de confabulació

El resultat de les proves és el següent:

Mida Original	Comprador 1	Comprador 2	Es recupera el comprador?
400x400	Josep	Peio	Si, al Peio
400x400	Lupe	Suso	Si, al Suso
400x400	Carlos	Frank	Si, tots dos, i un altre comprador innocent

Taula 4: Resultat de les proves de confabulació

En aquest cas sempre ha enxampat els confabuladors, però en un 33% de les vegades s'ha enxampat a un innocent.

1.4. Proves de reducció de mida

El resultat de les proves és el següent:

Mida Original	% de reducció	Nova Mida	% d'ampliació	Es recupera el comprador?
400x400	90%	360x360	110%	Si, amb exactitud
400x400	70%	280x280	130%	Si, amb exactitud
400x400	50%	200x200	200%	Si, però inculpa a altres compradors
400x400	30%	120x120	333%	No, inculpa a altres compradors

Taula 5: Resultat de les proves de reducció de mida

En aquest cas, tenim un 50% d'èxit total i un 75% d'èxit parcial. Es tracta d'uns marges alts de robustesa.

1.5. Proves d'ampliació de mida

El resultat de les proves és el següent:

Mida Original	% d'ampliació	Nova Mida	% de reducció	Es recupera el comprador?
400x400	110%	440x440	91%	Si, però inculpa a altres compradors
400x400	130%	520x520	76%	Si, però inculpa a altres compradors
400x400	150%	600x600	67%	Si, amb exactitut
400x400	190%	760x760	52%	No, inculpa un altre comprador

Taula 6: Resultat de les proves d'ampliació de mida

En aquest cas, un 75% dels casos trobem als compradors, tot i que només en un 25% tenim la certesa total. Es pot pensar que simplificarà força la tasca de recerca del culpable, si coneixem les estadístiques.

1.6. Proves de compressió

El resultat de les proves és el següent:

Mida Original	% Qualitat	Es recupera el comprador?
400x400	90%	No, no es detecta cap marca
400x400	60%	No, no es detecta cap marca
400x400	40%	No, no es detecta cap marca
400x400	20%	No, no es detecta cap marca

Taula 7: Resultat de les proves de compressió

En aquest cas, les proves han demostrat que el mètode no funciona per al cas de que el comprador comprimeixi la imatge i la publiqui. En cap dels casos no s'ha pogut trobat la marca.

Capítol 6. Conclusions

La tecnologia actual no facilita la protecció dels drets d'autor de les obres culturals que es poden plasmar en format digital. Pertant estan exposades a la còpia indiscriminada i a la *pirateria*. Així ens situem en un entorn que, inicialment, està reglamentat per la llei, però la realitat és que aquesta reglamentació és de difícil aplicació. Donat que el suport per part de la tecnologia (hardware i software) encara no està madur, mentrestant (2 ó 3 anys) poden prosperar iniciatives com la que s'ha exposat en aquest treball.

El mètode de marcatge a l'aigua es caracteritza per la seva imperceptibilitat. En no apreciar-se cap alteració en la imatge, un comprador a priori no suposa que hi hagi una marca en la imatge. Es tracta d'un mètode més robust que, per exemple, afegir un codi de barres en una banda blanca sota la imatge. Es treballa exclussivament amb imatges tipus *bitmap*, quadrades, no comprimides i de 24 bits per pixel de profunditat de color. Es tracta d'un format digital de qualitat similar als que s'obtenen a les càmeres de fotografia de nova generació.

El producte que s'ha desenvolupat encasta la marca en la imatge molt ràpidament, fins i tot amb imatges d'alta resolució. El procés invers de recuperació de la marca, també troba la identitat del comprador molt ràpidament. Es tracta, doncs, d'un mètode eficient pel que fa al temps de resposta. Usa una paraula única per a cada comprador, triada d'un codi especial.

Es tracta d'un mètode privat de marcatge. Només qui poseeix la imatge original sense marcar és qui podrà recuperar la identitat encastada en una imatge marcada.

Segons les proves efectuades, enfront alteracions perceptibles de la imatge marcada, trobem que el mètode és robust davant les confabulacions de compradors i reduccions de la mida de la imatge. No és tant robust quan s'analitza una imatge ampliada. Finalment, s'ha comprovat que no és capaç de recuperar la marca enfront un procés de compressió i descompressió de la imatge.

Hi ha tècniques que permeten millorar la robustesa del mètode i que s'han enunciat en el capítol dedicat a les proves. En tot cas, aquest augment de qualitat és a costa d'augmentar la capacitat d'anàlisi del software i d'introduir punts de referència en la imatge, que tanmateix poden ser objecte d'estudi dels confabuladors.

Capítol 7. Glossari

Autor: La persona que ha efectuat un treball artístic, que ha estat producte de la seva actuació física i/o intel.lectual.

Bitmap: Veure "Mapa de bits"

BMP: Extensió amb el que es reconeixen els arxius de disc que contenen un "bitmap". També s'anomena de la mateixa manera el propi "bitmap".

Codi binari: Codi composat per 2^n paraules d'n bits cadascuna. La manera de generar els bits defineix diferents codis. De tots ells, el més popular és el binari natural: 000, 001, 010, 011, 100, 101, 110, 111.

Codi anticonfabulació: Codi binari composat per paraules d'n bits cadascuna, que té una distància prou gran com per enxampar confabulacions entre compradors.

Codi Hamming: Codi binari detector i corrector d'errors, compost d'un codi binari natural de distància 1 i un codi afegit tal que el nombre d'uns i zeros en cada combinació (paritat) romangui constant.

Confabulador: Comprador d'una imatge marcada, que participa en una *Confabulació*.

Confabulació: Activitat poc honesta de dos compradors que, en comprovar que imatges iguals venudes a persones diferents tenen bits diferents, els alteren, per tal d'esborrar la identificació del comprador, o inculpar a un altre comprador innocent.

Comprador: Persona o entitat que adquireix el dret de poseir una còpia d'una imatge, per al seu ús personal.

Compressió: Mètode que redueix el volum que ocupa un "bitmap" a costa d'aplicar una combinació d'algorisme estàndar de compressió i de sacrificar-ne una part percentualment variable de la seva qualitat. Veure "JPG".

Copyright: Veure "Drets de còpia".

Descompressió: Procés invers al de compressió, pel qual es recupera un "bitmap" en la seva mida original apropiat per a la seva visualització.

Distància d'un codi binari: El nombre mínim de digits que canvien entre totes les combinacions de dues paraules del codi.

Drets de còpia: Dret reconegut d'un autor envers la seva creació, que permet fer-ne una còpia i cedir-la, llogar-la, donar-la o vendre-la a un tercer. Cal destacar que l'ús d'aquest dret no implica la seva transferència al tercer.

Drets d'autor: Drets que la llei otorga a qui efectua qualsevol tipus de creació artística. L'autor pot cedir, llogar, donar o vendre alguns o tots aquests drets a tercers.

JPG o **JPEG**: Mètode popular i eficient de compressió d'imatges. Aquest mètode elimina informació de la imatge original, que queda lleugerament deteriorada, tot i que generalment el canvi no és perceptible.

Marca: Veure "Marca a l'aigua"

Marca a l'aigua: Mot que s'usa en sentit metafòric, per comparació amb les textures que s'hi posen físicament en el procés de fabricació d'alguns papers (per exemple als

bitllets moneda) per acció d'un motlle i de l'aigua en el procés de creació. En sentit figurat, fa referència al mètode informàtic que es descriu en aquesta memòria i la doble capacitat que el caracteritza: guardar una informació i ser difícilment perceptible.

Mapa de bits: Format gràfic estàndar, que es composa d'una capcelera (amb informació sobre la mida del gràfic, paleta de colors que usa i compressió) i una tira de bits que representa la informació de color dels punts de la imatge. La informació de color pot fer referència a la paleta o a un color natural. Aquest format incorpora generalment la informació sense comprimir.

Pirateria: En el context, activitat il·lícita de reproducció d'obres protegides, sense autorització del propietari dels drets, amb l'objectiu d'ús personal (a petita escala) o de venta al detall (a gran escala).

Pixel: Punt de color. Unitat d'informació de color de la imatge. L'amplada i l'alçada de la imatge es medeixen en el nombre de punts de color o *pixels*.

Sistema privat de marca: Mètode que requereix l'ús de la imatge original per recuperar la marca. D'aquesta manera només qui la poseeix en pot esbrinar la identitat.

Sistema públic de marca: Mètode que no requereix la imatge original per obtenir la marca. S'ha de conèixer però la tècnica com s'ha inserit i les claus assignades al comprador (si s'escau), per recuperar la identitat.

Truecolor: Denominació que rep el color que es representa amb un mínim de 24 bits. Es tracta d'una paleta que pot representar fins a 2^24 colors, més de 16 milions, i que reprodueix de manera bastant fidel l'espectre visible.

Visual C: Software de desenvolupament d'aplicacions, propietat de Microsoft, per al Sistema Operatiu Windows, també propietat de Microsoft.

Watermark: Veure "Marca a l'aigua".

Watermarking: Tècnica que s'usa per inserir una marca a l'aigua en una imatge.

Capítol 8. Bibliografia

Aquest treball no hauria estat possible sense l'aportació realitzada pels autors dels treballs que s'indiquen a continuació:

Referència	Autor	Descripció del contingut	Ubicació
[SGAE-BOL]	SGAE	Informes mensuales de la Unidad de Análisis de la Sociedad General de Autores (8 de Junio, 1 de Junio, 12 de Mayo, 3 de Mayo)	http://www1.sgae.es/ua/
[KAZAA]	Software lliure	Client P2P de la xarxa lliure Kazaa	
[EMULE]	Software lliure	Client P2P de la xarxa lliure Emule	
[LPROINT]	Govern espanyol	Ley de la Propiedad Intelectual	
[ADOBE]	Adobe Inc	Productes i serveis per a la gestió de la documentació a l'empresa.	http://www.adobe.com
[CANON]	Canon	Canon Technology >> Software Technologies >> Digital Watermark Technology for Images	http://www.canon.com/technology/detail/software/digital_watermark/
[IEEE]	IEEE Internet Computing	Managing Copyright: Watermark and Cryptography Algorithms	http://dsonline.computer.org/0205/features/w3piva.htm
[DICOTEL]	Digital Copyright Technologies	Digital Copyright Protection for Multimedia Documents	http://vision.unige.ch/publications/postscript/99/HerrigelPun_telecom99.pdf
[EIPPC98]	Joseph J.K. Ruanaidh i Shelby Pereira	A Secure Robust Digital Image Watermark	http://vision.unige.ch/publications/postscript/98/ORuanaidhPereira_eipppc98.pdf
[BITMAP]	Microsoft Corp	Definició del format bitmap	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/gdicpp/GDIPlus/GDIPlusReference/Classes/Bitmap.asp
[BITMAP2]	Aurigma	Codificació i decodificació del format	http://www.aurigma.com/support/Help/GraphicsProcessor/BMPEncoderInfo.htm
[BITMAP3]	Universitaris	Descripció del format bitmap	http://astronomy.swin.edu.au/~pbourke/dataformats/bmp/
[DIBLOOK]	Microsoft Developer Network	Programa d'exemple pel tractament i visualització d'imatges bitmap en Windows.	
[BITMAP4]	Microsoft Developer Network	GDI+ Definició dels diferents formats gràfics	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/gdicpp/GDIPlus/AboutGDIPlus/ImagesBitmapsandMetafiles/TypesofBitmaps.asp
[PETZOLD]	Charles Petzold	Programación en Windows (v3.0, ISBN 84-7614-384-2)	

Taula 8: Taula de referències bibliogràfiques

Annex 1. Codi font dels programes

A continuació es mostra el codi font dels productes desenvolupats i de les llibreries de codi auxiliars.

1.1. Programa que encasta la marca a la imatge

```
////////////////////////////////////////////////////////////////////////////
// Aplicació de generació i recuperació de marca a l'aigua
// TFC - Enginyeria del Software - Seguretat
// Autor del programa: Miquel Conesa i Pellicer
// Data: 1er Semestre de 2004
////////////////////////////////////////////////////////////////////////////
// Paràmetres:
// Proves generació  : C:\desenvolupament\marca\imatges\aoo.bmp m lenna
// Proves recuperació: C:\desenvolupament\marca\imatges\aoo.lenna.bmp e
C:\desenvolupament\marca\imatges\aoo.bmp
////////////////////////////////////////////////////////////////////////////
//

#include "stdafx.h"
#include <math.h>
#include "dibapi.h"
#include "marca.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////////////////////////////////////////////
// The one and only application object

CWinApp theApp;
using namespace std;


////////////////////////////////////////////////////////////////////////////
// Dades de l'aplicació
////////////////////////////////////////////////////////////////////////////
//
// Aquest és el codi de Hamming dual, anti-confabulacions
char codi_HAMMING[8][8] =
{
        "0000000", "0011011", "0101101", "0110110",
        "1001110", "1010101", "1100011", "1111000"
};

// L'arxiu d'on obté les dades de marcatge
char arxiuParams[_MAX_PATH], *pos;
#define HISTORIC "marcatge.ini"

// Nombre de pixels que ha de contenir cada caixa, on hi va la marca
#define AMPLE_CAIXA 10

// Errorlevel que dona el programa: 0 => OK, major que zero => errors
int nRetCode = 0;

////////////////////////////////////////////////////////////////////////////
// Processos auxiliars generals
////////////////////////////////////////////////////////////////////////////
```

```
// Procés auxiliar que comprova si existeix un arxiu
bool Existeix(CString arxiu)

        //  (Si és la primera vegada, li assigna un codi. Si ho repetim, sempre
        //    obtenim el mateix codi, mentre no canviin el comprador o la imatge).
        char Codi[16];
        int imPos = arxiuOriginal.ReverseFind('\\');
        CString Imatge = (imPos == -1) ? arxiuOriginal : arxiuOriginal.Mid(imPos+1);
        GetPrivateProfileString(Imatge, Comprador, "", Codi, sizeof(Codi), arxiuParams);
        if (!strlen(Codi))
        {
                // Aquest comprador no existeix a la taula, li assignem un codi
                int assigActual = GetPrivateProfileInt(Imatge, "assignats", 0, arxiuParams);
                if (++assigActual != 8)
                {
                        char valor[8];
                        // Actualitzem el comptador de marques assignades (compradors satisfets)
                        WritePrivateProfileString(Imatge, "assignats", itoa(assigActual, valor,
10), arxiuParams);
                        // Fem la marca simètrica, per a poder-la recuperar.
                        strcpy(Codi, codi_HAMMING[assigActual]);
                        WritePrivateProfileString(Imatge, Comprador, Codi, arxiuParams);
                        WritePrivateProfileString(Imatge, Codi, Comprador, arxiuParams);
                }
                else
                {
                        MostraError(6); // No hi ha més valors
                        return CString("");
                }
        }
        cout << (LPCSTR)Codi << endl;
        return (CString)Codi;
}

// Procés auxiliar que carrega tot el DIB de l'arxiu a memòria
HDIB CarregaImatge(CString arxiuOriginal)
{
        HDIB hDIB = NULL;

        cout << _T("Carrega imatge : ");
        if (Existeix(arxiuOriginal))
        {
                CFile arxiuDIB(arxiuOriginal, CFile::modeRead | CFile::shareExclusive |
CFile::typeBinary);
                hDIB = ::ReadDIBFile(arxiuDIB);
                arxiuDIB.Close();
                if (hDIB)
                        cout << _T("OK") << endl;
                else
                        MostraError(7); // No és pas un DIB
        }
        else
                MostraError(3, arxiuOriginal);

        return hDIB;
}

// Procés auxiliar que guarda tot el DIB de la memòria a un arxiu
void GuardaImatge(HDIB m_hDIB, CString arxiuOriginal, CString Comprador)
{
        cout << _T("Imatge generada: ");

        CString nouNom = arxiuOriginal;

                nouNom = nouNom.Mid(0,nouNom.Find('.')+1);
                nouNom += Comprador;
                nouNom += ".bmp";

        CFile nouArxiu(nouNom, CFile::modeCreate | CFile::modeWrite | CFile::shareExclusive |
CFile::typeBinary);

        if (SaveDIB(m_hDIB, nouArxiu))
                cout << (LPCSTR)nouNom << endl;
        else
```

```
        else
                MostraError(8, nouNom);       // No es pot guardar la imatge marcada
}

// Procés auxiliar que comprova si la imatge compleix les restriccions del nostre procés
bool ImatgeValidaPerMarcar(HDIB m_hDIB)
{
        cout << _T("Geometria img. : ");
        // Recollim informació sobre la geometria i bpp de la imatge
        LPSTR lpDIB = (LPSTR) ::GlobalLock((HGLOBAL) m_hDIB);
        DWORD ampleDIB = DIBWidth(lpDIB), alturaDIB = DIBHeight(lpDIB), colorsDIB =
DIBNumBPP(lpDIB);
        ::GlobalUnlock((HGLOBAL) m_hDIB);

        // Comprovem les dues restriccions: quadrada i 3 bytes per pixel
        if (ampleDIB != alturaDIB)
        {
                MostraError(9); // No és una imatge quadrada
                return false;
        }
        //
        if (colorsDIB != 24)
        {
                MostraError(10); // No és una imatge de 24bpp
                return false;
        }

        CString Info;
        Info.Format("%dx%d/%dbpp", ampleDIB, alturaDIB, colorsDIB);
        cout << (LPCSTR)Info << endl;

        return true;
}

// Procés auxiliar que incrusta la marca al bitmap que tenim a memòria,
void MarcaImatge(HDIB m_hDIB, CString Marca)
{
        // Tenim tota la informació i està comprovat que és correcta, pertant,
        // procedim a encastar la marca
        cout << _T("Generacio marca: ");
        //
        char Codi[16];
                strcpy(Codi, (LPCSTR)Marca);
        //
        LPSTR lpDIB = (LPSTR) ::GlobalLock((HGLOBAL) m_hDIB);
        // Dividim el nombre pixels en caixes de 10
        DWORD ampleDIB = DIBWidth(lpDIB);
        DWORD num_qi = ampleDIB / AMPLE_CAIXA;
        LPSTR pixels = FindDIBBits(lpDIB);
        LPSTR qi_pos, cij_pos;
        DWORD ci_pos;
        // Ens posicionem a cada caixa (q[x,y])
        DWORD qy, qx, ci, cj, mi;
        for (qy = 0; qy < num_qi; qy++)
        {
                qi_pos = pixels + (qy * AMPLE_CAIXA * ampleDIB * 3);
                for (qx = 0, mi = 0; qx < num_qi; qx++)
                {
                        // Estem a la caixa q[x,y]
                        qi_pos += AMPLE_CAIXA * 3;
                        // Ara progressem dins qi en fileres i columnes.
                        for (ci = 0; ci < AMPLE_CAIXA; ci++)
                        {
                                // Calculem la filera dins qi
                                ci_pos = ci * ampleDIB * 3;
                                for (cj = 0; cj < AMPLE_CAIXA; cj++)
                                {
                                        // Trobem el pixel c[ij] i li apliquem la marca.
                                        // Avançar un pixel ens costa 3 bytes.
                                        cij_pos = qi_pos + ci_pos + cj*3;
                                        // Apliquem la màscara al pixel trobat
```

```
                                        // Apliquem la màscara al pixel trobat
                                        if (Codi[mi++] == '1')
                                        {
                                                // Només si el bit del codi és '1',
                                                // invertim el bit de menor pes
                                                // dels 3 bytes del pixel.
                                                *(cij_pos)   ^= 1;
                                                *(cij_pos+1) ^= 1;
                                                *(cij_pos+2) ^= 1;
                                        }
                                        // Tornem al principi del vector M
                                        if (mi == 8)
                                                mi = 0;
                                }
                        }
                }
        }
        ::GlobalUnlock((HGLOBAL) m_hDIB);
        cout << _T("OK") << endl;
}
//
////////////////////////////////////////////////////////////////////////////
// Procés que encasta la marca en l'arxiu original
////////////////////////////////////////////////////////////////////////////
//
void EncastaMarca(CString arxiuOriginal, CString Comprador)
{
        // Presentem les dades de la operació
        cout << _T("Proces de marcatge d'una imatge original") << endl << endl;
        cout << _T("Imatge original: ") << (LPCSTR)arxiuOriginal << endl;
        cout << _T("Comprador      : ") << (LPCSTR)Comprador << endl;
        // Llegim la imatge a memòria
        HDIB m_hDIB = CarregaImatge(arxiuOriginal);
        if (m_hDIB)
        {
                // Comprovem si és una imatge vàlida i es pot marcar
                if (ImatgeValidaPerMarcar(m_hDIB))
                {
                        // Assignem un vector del codi de Hamming per a l'usuari i la imatge.
                        CString Codi = AssignaCodi(arxiuOriginal, Comprador);
                        if (!Codi.IsEmpty())
                        {
                                // Tenim tota la informació i està comprovat que és correcta,
pertant,
                                // procedim a encastar la marca.
                                MarcaImatge(m_hDIB, Codi);

                                // Un cop modificada la imatge, la guardem amb el nom que li
correspon
                                GuardaImatge(m_hDIB, arxiuOriginal, Comprador);
                        }
                }
                // Esborrem l'objecte DIB de la memòria
                GlobalFree((HGLOBAL) m_hDIB);
        }
}
//
////////////////////////////////////////////////////////////////////////////
// Processos auxiliars per a la funció de recuperar la marca d'una imatge
////////////////////////////////////////////////////////////////////////////
//
// Procés auxiliar que extreu el codi de la imatge marcada, basant-se en la
// imatge original sense marcar.
CString ExtreuMarca(HDIB m_hDIB_m, HDIB m_hDIB_o)
{
        cout << _T("Analisi geomet.: ");

        int CodiUns[] = {0,0,0,0,0,0,0},      // Votacions de '1's
                CodiZeros[] = {0,0,0,0,0,0,0};        // Votacions de '0's

        // Accedim als bytes de memòria on es guarden els dibuixos
```

```
        // Accedim als bytes de memòria on es guarden els dibuixos
        LPSTR lpDIBo = (LPSTR) ::GlobalLock((HGLOBAL) m_hDIB_o);
        LPSTR lpDIBm = (LPSTR) ::GlobalLock((HGLOBAL) m_hDIB_m);
        LPSTR pixels_o = FindDIBBits(lpDIBo);
        LPSTR pixels_m = FindDIBBits(lpDIBm);

        // Si la imatge marcada no té la mateixa mida que la original,
        //   li fem un escalatge.
        DWORD ampleDIB = DIBWidth(lpDIBo), ampleDIBm = DIBWidth(lpDIBm);
        DWORD altDIB = DIBHeight(lpDIBo), altDIBm = DIBHeight(lpDIBm);
        if (ampleDIB != ampleDIBm || altDIB != altDIBm)
        {
                MostraError(14);
                return (CString)_T("");
        }

        cout << _T("Marca inserida : ");
        // Dividim el nombre pixels en caixes de 10
        DWORD num_qi = ampleDIB / AMPLE_CAIXA;
        LPSTR qi_pos_o, qi_pos_m, cij_pos_o, cij_pos_m;
        DWORD ci_pos;
        // Ens posicionem a cada caixa (q[x,y])
        DWORD qy, qx, ci, cj, mi;
        for (qy = 0; qy < num_qi; qy++)
        {
                qi_pos_o = pixels_o + (qy * AMPLE_CAIXA * ampleDIB * 3);
                qi_pos_m = pixels_m + (qy * AMPLE_CAIXA * ampleDIB * 3);

                for (qx = 0, mi = 0; qx < num_qi; qx++)
                {
                        // Estem a la caixa q[x,y]
                        qi_pos_o += AMPLE_CAIXA * 3;
                        qi_pos_m += AMPLE_CAIXA * 3;

                        // Ara progressem dins qi en fileres i columnes.
                        for (ci = 0; ci < AMPLE_CAIXA; ci++)
                        {
                                // Calculem la filera dins qi
                                ci_pos = ci * ampleDIB * 3;
                                for (cj = 0; cj < AMPLE_CAIXA; cj++)
                                {
                                        // Trobem el pixel c[ij] i li apliquem la marca.
                                        // Avançar un pixel ens costa 3 bytes.
                                        cij_pos_o = qi_pos_o + ci_pos + cj*3;
                                        cij_pos_m = qi_pos_m + ci_pos + cj*3;

                                        // Ara comprovem quin nombre de bits ha canviat en la imatge
marcada
                                        //   respecte de la original. En cas que hi hagi més d'un bit
canviat
                                        //   considerem que hi ha posat un '1', en cas contrari, un
'0'.
                                        // Però només farem això si els 4 MSB de cada byte
coincideixen
                                        //   exactament. En cas contrari, no estariem comparant la
mateixa
                                        //   informació, cosa que distorsionaria el resultat.
                                        if (((*(cij_pos_o) & 0xF0) == (*(cij_pos_m) & 0xF0)) &&
                                                ((*(cij_pos_o+1) & 0xF0) == (*(cij_pos_m+1) & 0xF0))
&&
                                                ((*(cij_pos_o+2) & 0xF0) == (*(cij_pos_m+2) & 0xF0)))
                                        {
                                                if ((((*(cij_pos_o) & 1) != (*(cij_pos_m) & 1)) +
                                                        ((*(cij_pos_o+1) & 1) != (*(cij_pos_m+1) & 1))
+
                                                        ((*(cij_pos_o+2) & 1) != (*(cij_pos_m+2) &
1))) > 1)
                                                        CodiUns[mi]++;
                                                else
                                                        CodiZeros[mi]++;
                                        }
                                        // Tornem al principi del vector M
```

```
                                        // Tornem al principi del vector M
                                        if (++mi == 8)
                                                mi = 0;
                                }
                        }
                }
        }
        ::GlobalUnlock((HGLOBAL) m_hDIB_o);
        ::GlobalUnlock((HGLOBAL) m_hDIB_m);

        // Hem obtingut un vector d'enters que ens indiquen el nombre d'1 i 0 trobats.
        // Ara analitzem quin és el valor mig (de tall) per considerar quin nombre
        //   d'encerts equival a un '1' i a un '0':
//      int Mitjana = max(Codi[0], max(Codi[1], max(Codi[2], max(Codi[3], max(Codi[4],
max(Codi[5], Codi[6])))))) / 2;
//      int     Mitja = (Codi[0] + Codi[1] + Codi[2] + Codi[3] + Codi[4] + Codi[5] + Codi[6]) / 7;
        CString Marca = _T("");

        for (int i=0; i<7; i++) Marca += CodiUns[i] > CodiZeros[i] ? "1" : "0";

        cout << (LPCSTR)Marca << endl;

        return Marca;
}

// Procés auxiliar que cerca quin comprador té assignat el codi indicat per
// a la imatge original.
CString CercaComprador(CString Codi, CString arxiuOriginal)
{
        cout << _T("Codi           : ") << (LPCSTR)Codi << _T(" Comprador : ");

        int imPos = arxiuOriginal.ReverseFind('\\');
        CString Imatge = (imPos == -1) ? arxiuOriginal : arxiuOriginal.Mid(imPos+1);
        int numCompradors = GetPrivateProfileInt(Imatge, "assignats", 0, arxiuParams);
        char Comprador[_MAX_PATH] = _T("");

        if (numCompradors)
        {
                char Comprador[_MAX_PATH];
                // Localitzem qui té assignat el codi indicat per a aquesta imatge
                GetPrivateProfileString(Imatge, Codi, "", Comprador, sizeof(Comprador),
arxiuParams);
                if (*Comprador)
                        cout << Comprador << endl;
                else
                        MostraError(12, Codi);
        }
        else
                MostraError(11, Imatge);      // No consten compradors d'aquesta imatge

        return Comprador;
}

// Procés auxiliar que cerca el codi Hamming més proper al que se li dona i
//   mostra el comprador o la llista de confabulats
void AnalitzaMarca(CString Marca, CString arxiuOriginal)
{
        cout << _T("Analisi marca  : ");

        bool Exacte = false;
        int Distancia = 100, Index = 0;

        for (int j=0, Dist; j<8; j++) // 8 valors definits, té el vector de codis Hamming
        {
                Dist=0;

                for (int i=0; i<7; i++)              // 7 bits té cadascun dels valors del vector
                        if (Marca[i] != codi_HAMMING[j][i])
                                ++Dist;                              // Si el bit canvia, augmenta
la distància

                // Si la distancia trobada és inferior, ens anotem la posició ja que és candidata
```

```
                // Si la distancia trobada és inferior, ens anotem la posició ja que és candidata
                if (Dist <= Distancia)
                {
                        Index = j;
                        if (Dist == 0 && Marca == codi_HAMMING[j])
                        {
                                Exacte = true;
                                break;
                        }
                        else
                                Distancia = Dist;
                }
        }

        // Analitzem el resultat
        if (Exacte)
        {
                // Si és un valor exacte teinm el comprador i la imatge és original
                cout << _T("es una imatge original") << endl;
                CercaComprador((CString)codi_HAMMING[Index], arxiuOriginal);
        }
        else
        {
                // Si és un valor inexacte, cerquem tots els codis amb la mateixa distància,
                // ja que tots ells són candidats
                cout << _T("es una imatge modificada / confabulada") << endl;
                cout << _T("Llista de possibles compradors: (Dmin=") << Distancia << _T(")") <<
endl;

                for (int j=0, Dist; j<8; j++) // 8 valors definits, té el vector de codis Hamming
                {
                        Dist=0;

                        for (int i=0; i<7; i++)              // 7 bits té cadascun dels valors del
vector
                                if (Marca[i] != codi_HAMMING[j][i])
                                        ++Dist;                              // Si el bit canvia,
augmenta la distància

                        // Si la distancia trobada és inferior, ens anotem la posició ja que és
candidata
                        if (Dist == Distancia)
                                CercaComprador((CString)codi_HAMMING[j], arxiuOriginal);
                }
        }
}

//
////////////////////////////////////////////////////////////////////////////////
// Procés que recupera la marca en l'arxiu original
////////////////////////////////////////////////////////////////////////////////
//
void RecuperaMarca(CString arxiuMarcat, CString arxiuOriginal)
{
        // Presentem les dades de la operació
        cout << _T("Proces de recuperacio de la marca") << endl << endl;
        cout << _T("Imatge original: ") << (LPCSTR)arxiuOriginal << endl;
        // Llegim la imatge original a memòria i verifiquem què és correcta
        HDIB m_hDIB_o = CarregaImatge(arxiuOriginal);
        if (m_hDIB_o && ImatgeValidaPerMarcar(m_hDIB_o))
        {
                cout << _T("Imatge marcada : ") << (LPCSTR)arxiuMarcat << endl;
                // Llegim la imatge original a memòria  i verifiquem què és correcta
                HDIB m_hDIB_m = CarregaImatge(arxiuMarcat);
                if (m_hDIB_m && ImatgeValidaPerMarcar(m_hDIB_m))
                {
                        // Anem a cercar el codi de la marca afegit, amb les dues imatges
                        CString Marca = ExtreuMarca(m_hDIB_m, m_hDIB_o);
                        if (Marca.GetLength())
                        {
                                // Si la marca obtinguda és zero, la imatge no està marcada
```

```
                                // Si la marca obtinguda és zero, la imatge no està marcada
                                if (Marca != codi_HAMMING[0])
                                        // Si hem arribat a trobar una marca, mirem de trobar qui
                                        // n'és el comprador (o compradors)
                                        AnalitzaMarca(Marca, arxiuOriginal);
                                else
                                        MostraError(13);
                        }
                        // Esborrem l'objecte DIB de la memòria
                        GlobalFree((HGLOBAL) m_hDIB_m);
                }
                // Esborrem l'objecte DIB de la memòria
                GlobalFree((HGLOBAL) m_hDIB_o);
        }
}

//
////////////////////////////////////////////////////////////////////////////////
// Programa principal
////////////////////////////////////////////////////////////////////////////////
//
int _tmain(int argc, TCHAR* argv[], TCHAR* envp[])
{
        // initialize MFC and print and error on failure
        if (AfxWinInit(::GetModuleHandle(NULL), NULL, ::GetCommandLine(), 0))
        {
                // El programa rep sempre 3 paràmetres:
                // marca lenna.bmp m pere (per marcar la imatge i crear la nova imatge
lenna.pere.bmp)
                // marca lenna.pere.bmp e lenna.bmp (per obtenir la marca de la imatge)
                if (argc == 4)
                {
                        // Comprovem si existeix l'arxiu històric de codis de marcatge
                        strcpy(arxiuParams, argv[0]);
                        pos = strrchr(arxiuParams, '\\');
                        strcpy(pos ? pos+1 : arxiuParams, HISTORIC);
                        if (Existeix((CString)arxiuParams))
                        {
                                // Revisem quina funció hem de fer
                                switch (*argv[2])
                                {
                                        case 'm': // Encastem la marca: argv[1] és l'arxiu original i
argv[3] el comprador
                                        case 'M':       EncastaMarca((CString)argv[1],
(CString)argv[3]);
                                                                break;
                                        case 'e': // Recuperem la marca: argv[1] és l'arxiu marcat i
argv[3] l'original
                                        case 'E':       RecuperaMarca((CString)argv[1],
(CString)argv[3]);
                                                                break;
                                        default : // No s'accepten altres paràmetres
                                                                MostraError(5);
                                }
                        }
                        else
                                MostraError(4, (CString)arxiuParams);
                }
                else
                        MostraError(2);
        }
        else
                MostraError(1);

        return nRetCode;
}
```

1.2. Programa que simula una possible confabulació de compradors

```
////////////////////////////////////////////////////////////////////////////
// Aplicació de generació d'imatges confabulades
// TFC - Enginyeria del Software - Seguretat
// Autor del programa: Miquel Conesa i Pellicer
// Data: 1er Semestre de 2004
////////////////////////////////////////////////////////////////////////////
// Paràmetres:
// Proves generació  : C:\desenvolupament\marca\imatges\aoo.joan.bmp aoo.
C:\desenvolupament\marca\imatges\pere.bmp
////////////////////////////////////////////////////////////////////////////
//

#include "stdafx.h"
#include "..\marca\dibapi.h"
#include "confabula.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////////////////////////////////////////////
// The one and only application object

CWinApp theApp;

using namespace std;

////////////////////////////////////////////////////////////////////////////
// Dades de l'aplicació
////////////////////////////////////////////////////////////////////////////
//
// Nombre de pixels que ha de contenir cada caixa, on hi va la marca
#define AMPLE_CAIXA 10

// Errorlevel que dona el programa: 0 => OK, major que zero => errors
int nRetCode = 0;

////////////////////////////////////////////////////////////////////////////
// Processos auxiliars generals
////////////////////////////////////////////////////////////////////////////
//
// Procés auxiliar que comprova si existeix un arxiu
bool Existeix(CString arxiu)
{
	FILE *fp = fopen(arxiu, "rb");

	if (fp)
	{
		fclose(fp);
		return true;
	}
	else
		return false;
}

// Procés auxiliar que mostra un error i canvia el codi de retorn del programa
void MostraError(int Error, CString mesInfo = _T(""));
void MostraError(int Error, CString mesInfo)
{
	cerr << _T("Error: ");
	switch(Error)
	{
		case 1:		cerr << _T("Ha fallat l'inicialitzacio de les MFC");
				break;
```

```
                case 2:         cerr << _T("No s'ha pasat el nombre correcte de parametres");
                                        break;
                case 3:         cerr << _T("No es pot obrir l'arxiu ");
                                        break;
                case 4:         cerr << _T("Les imatges no son de la mateixa mida");
                                        break;
//              case 5:         cerr << _T("L'acció especificada no és correcta (ha de ser m o e)");
//                                      break;
//              case 6:         cerr << _T("No queden codis d'aquesta imatge per assignar");
//                                      break;
                case 7:         cerr << _T("L'arxiu indicat no és una imatge BMP");
                                        break;
                case 8:         cerr << _T("No es pot guardar la imatge marcada en l'arxiu ");
                                        break;
                case 9:         cerr << _T("L'arxiu indicat no conte una imatge quadrada");
                                        break;
                case 10:        cerr << _T("L'arxiu indicat no conte una imatge de 24bpp");
                                        break;
//              case 11:        cerr << _T("No consten compradors de la imatge ");
//                                      break;
//              case 12:        cerr << _T("No hi ha cap comprador de la imatge amb el codi ");
//                                      break;
//              case 13:        cerr << _T("No es detecta cap marca en la imatge");
//                                      break;

                default:        cerr << _T("Aquesta funcio encara no s'ha implementat");
                                        break;
        }

        if (mesInfo.GetLength())
                cerr << (LPCSTR)mesInfo;

        cerr << endl;
        nRetCode = Error;
}

//
////////////////////////////////////////////////////////////////////////////////
// Processos auxiliars per a la funció de marcar la imatge
////////////////////////////////////////////////////////////////////////////////
//
// Procés auxiliar que comprova si la imatge compleix les restriccions del nostre procés
bool ImatgeValidaPerMarcar(HDIB m_hDIB)
{
        cout << _T("Geometria img. : ");
        // Recollim informació sobre la geometria i bpp de la imatge
        LPSTR lpDIB = (LPSTR) ::GlobalLock((HGLOBAL) m_hDIB);
        DWORD ampleDIB = DIBWidth(lpDIB), alturaDIB = DIBHeight(lpDIB), colorsDIB =
DIBNumBPP(lpDIB);
        ::GlobalUnlock((HGLOBAL) m_hDIB);

        // Comprovem les dues restriccions: quadrada i 3 bytes per pixel
        if (ampleDIB != alturaDIB)
        {
                MostraError(9); // No és una imatge quadrada
                return false;
        }
        //
        if (colorsDIB != 24)
        {
                MostraError(10); // No és una imatge de 24bpp
                return false;
        }

        CString Info;
        Info.Format("%dx%d/%dbpp", ampleDIB, alturaDIB, colorsDIB);
        cout << (LPCSTR)Info << endl;

        return true;
}

// Procés auxiliar que carrega tot el DIB de l'arxiu a memòria
```

```
// Procés auxiliar que carrega tot el DIB de l'arxiu a memòria
HDIB CarregaImatge(CString arxiuOriginal)
{
	HDIB hDIB = NULL;

	cout << _T("Carrega imatge : ");
	if (Existeix(arxiuOriginal))
	{
		CFile arxiuDIB(arxiuOriginal, CFile::modeRead | CFile::shareExclusive |
CFile::typeBinary);
		hDIB = ::ReadDIBFile(arxiuDIB);
		arxiuDIB.Close();
		if (hDIB)
			cout << _T("OK") << endl;
		else
			MostraError(7); // No és pas un DIB
	}
	else
		MostraError(3, arxiuOriginal);

	return hDIB;
}

// Procés auxiliar que guarda tot el DIB de la memòria a un arxiu
void GuardaImatge(HDIB m_hDIB, CString arxiuOriginal)
{
	cout << _T("Imatge generada: ");

	CString nouNom;
	int ppos = arxiuOriginal.ReverseFind('.');
	if (ppos > -1)
		nouNom = arxiuOriginal.Left(ppos+1);

	nouNom += _T("conf.bmp");

	CFile nouArxiu(nouNom, CFile::modeCreate | CFile::modeWrite | CFile::shareExclusive |
CFile::typeBinary);

	if (SaveDIB(m_hDIB, nouArxiu))
		cout << (LPCSTR)nouNom << endl;
	else
		MostraError(8, nouNom);	// No es pot guardar la imatge marcada
}

// Aquesta rutina rep dues tires de bytes, que corresponen a ambdues imatges
//  marcades. La primera s'ha de modificar tant bon punt detectem  diferències.
void ExecutaConfabula(LPSTR lpDIBo, LPSTR lpDIBm)
{
	cout << _T("Confabulant Img: ");
	srand(GetTickCount());
	// Dividim el nombre pixels en caixes de 10
	DWORD ampleDIB = DIBWidth(lpDIBo);
	DWORD num_qi = ampleDIB / AMPLE_CAIXA;
	LPSTR pixels_o = FindDIBBits(lpDIBo);
	LPSTR pixels_m = FindDIBBits(lpDIBm);
	LPSTR qi_pos_o, qi_pos_m, cij_pos_o, cij_pos_m;
	DWORD ci_pos;
	unsigned int rVal;
	// Ens posicionem a cada caixa (q[x,y])
	DWORD qy, qx, ci, cj;
	for (qy = 0; qy < num_qi; qy++)
	{
		qi_pos_o = pixels_o + (qy * AMPLE_CAIXA * ampleDIB * 3);
		qi_pos_m = pixels_m + (qy * AMPLE_CAIXA * ampleDIB * 3);

		for (qx = 0; qx < num_qi; qx++)
		{
			// Estem a la caixa q[x,y]
			qi_pos_o += AMPLE_CAIXA * 3;
			qi_pos_m += AMPLE_CAIXA * 3;

			// Ara progressem dins qi en fileres i columnes.
```

```
                        // Ara progressem dins qi en fileres i columnes.
                        for (ci = 0; ci < AMPLE_CAIXA; ci++)
                        {
                                // Calculem la filera dins qi
                                ci_pos = ci * ampleDIB * 3;
                                for (cj = 0; cj < AMPLE_CAIXA; cj++)
                                {
                                        // Trobem el pixel c[ij] i li llegim la marca.
                                        // Avançar un pixel ens costa 3 bytes.
                                        cij_pos_o = qi_pos_o + ci_pos + cj*3;
                                        cij_pos_m = qi_pos_m + ci_pos + cj*3;

                                        // Ara comparem els tres bytes. Si hi ha diferències,
                                        //   li apliquem un canvi al.leatori al lsb.
                                        if ((*(cij_pos_o) != *(cij_pos_m)) ||
                                                (*(cij_pos_o+1) != *(cij_pos_m+1)) ||
                                                (*(cij_pos_o+2) != *(cij_pos_m+2)))
                                        {
                                                rVal = rand() & 1;
                                                *(cij_pos_o)   ^= rVal;
                                                *(cij_pos_o+1) ^= rVal;
                                                *(cij_pos_o+2) ^= rVal;
                                        }
                                }
                        }
                }
        }
        cout << _T("OK") << endl;
}

//
////////////////////////////////////////////////////////////////////////////
// Programa de confabulació a partir de dues imatges marcades
////////////////////////////////////////////////////////////////////////////
//
void IniciaConfabula(CString arxiu1, CString arxiu2)
{
        // Primer comprovem que es tracta d'imatges amb les mateixes característiques
        //  geomètriques. No s'hi val a confabular pedaços de diferents mida.
        cout << "Programa de confabulacio de dos compradors" << endl << endl;
        cout << _T("Primera Imatge : ") << (LPCSTR)arxiu1 << endl;
        // Llegim la imatge original a memòria i verifiquem què és correcta
        HDIB m_hDIB_o = CarregaImatge(arxiu1);
        if (m_hDIB_o && ImatgeValidaPerMarcar(m_hDIB_o))
        {
                cout << _T("Segona Imatge  : ") << (LPCSTR)arxiu2 << endl;
                // Llegim la imatge original a memòria  i verifiquem què és correcta
                HDIB m_hDIB_m = CarregaImatge(arxiu2);
                if (m_hDIB_m && ImatgeValidaPerMarcar(m_hDIB_m))
                {
                        // Ara comprovem que tenen les mateixes caracteristiques:
                        LPSTR lpDIBo = (LPSTR) ::GlobalLock((HGLOBAL) m_hDIB_o);
                        LPSTR lpDIBm = (LPSTR) ::GlobalLock((HGLOBAL) m_hDIB_m);

                        cout << _T("Geometria      : ");
                        if ((DIBWidth(lpDIBo) == DIBWidth(lpDIBm)) &&
                                (DIBHeight(lpDIBo) == DIBHeight(lpDIBm)))
                        {
                                // Aquest procés transforma la primera imatge en una altra
                                //   variant al.leatòriament els bits si detecta diferències
                                cout << _T("OK") << endl;
                                ExecutaConfabula(lpDIBo, lpDIBm);
                                GuardaImatge(m_hDIB_o, arxiu1);
                        }
                        else
                                MostraError(4);
                        // Esborrem l'objecte DIB de la memòria
                        GlobalFree((HGLOBAL) m_hDIB_m);
                }
                // Esborrem l'objecte DIB de la memòria
                GlobalFree((HGLOBAL) m_hDIB_o);
        }
}
```

```
//
/////////////////////////////////////////////////////////////////////////////
// Programa principal
/////////////////////////////////////////////////////////////////////////////
//
int _tmain(int argc, TCHAR* argv[], TCHAR* envp[])
{
	// initialize MFC and print and error on failure
	if (AfxWinInit(::GetModuleHandle(NULL), NULL, ::GetCommandLine(), 0))
	{
		// El programa rep sempre 2 paràmetres:
		// confabula lenna.pere.bmp lenna.maria.bmp
		if (argc == 3)
		{
			// Comprovem que existeixen els dos arxius
			if (Existeix((CString)argv[1]))
			{
				if (Existeix((CString)argv[2]))
					// Executa el procés
					IniciaConfabula((CString)argv[1], (CString)argv[2]);
				else
					MostraError(3, (CString)argv[2]);
			}
			else
				MostraError(3, (CString)argv[1]);
		}
		else
			MostraError(2);
	}
	else
		MostraError(1);

	return nRetCode;
}
```

1.3. Llibreria auxiliar de tractament d'imatges *bitmap*

Aquest codi font s'ha extret del programa diblook [DIBLOOK] i s'ha incorporat al projecte com a llibreria de funcions.

```
//  dibapi.cpp
//
//  Source file for Device-Independent Bitmap (DIB) API.  Provides
//  the following functions:
//
//  PaintDIB()          - Painting routine for a DIB
//  CreateDIBPalette()  - Creates a palette from a DIB
//  FindDIBBits()       - Returns a pointer to the DIB bits
//  DIBWidth()          - Gets the width of the DIB
//  DIBHeight()         - Gets the height of the DIB
//  PaletteSize()       - Gets the size required to store the DIB's palette
//  DIBNumColors()      - Calculates the number of colors
//                        in the DIB's color table
//  CopyHandle()        - Makes a copy of the given global memory block
//
// This is a part of the Microsoft Foundation Classes C++ library.
// Copyright (C) 1992-1998 Microsoft Corporation
// All rights reserved.
//
// This source code is only intended as a supplement to the
// Microsoft Foundation Classes Reference and related
// electronic documentation provided with the library.
// See these sources for detailed information regarding the
// Microsoft Foundation Classes product.

#include "stdafx.h"
#include "dibapi.h"
#include <io.h>
#include <errno.h>

/****************************************************************************
 *
 * PaintDIB()
 *
 * Parameters:
 *
 * HDC hDC          - DC to do output to
 *
 * LPRECT lpDCRect  - rectangle on DC to do output to
 *
 * HDIB hDIB        - handle to global memory with a DIB spec
 *                    in it followed by the DIB bits
 *
 * LPRECT lpDIBRect - rectangle of DIB to output into lpDCRect
 *
 * CPalette* pPal   - pointer to CPalette containing DIB's palette
 *
 * Return Value:
 *
 * BOOL             - TRUE if DIB was drawn, FALSE otherwise
 *
 * Description:
 *   Painting routine for a DIB.  Calls StretchDIBits() or
 *   SetDIBitsToDevice() to paint the DIB.  The DIB is
 *   output to the specified DC, at the coordinates given
 *   in lpDCRect.  The area of the DIB to be output is
 *   given by lpDIBRect.
 *
 ****************************************************************************/

BOOL WINAPI PaintDIB(HDC     hDC,
```

```
BOOL WINAPI PaintDIB(HDC     hDC,
                                     LPRECT  lpDCRect,
                                     HDIB    hDIB,
                                     LPRECT  lpDIBRect,
                                     CPalette* pPal)
{
        LPSTR    lpDIBHdr;            // Pointer to BITMAPINFOHEADER
        LPSTR    lpDIBBits;           // Pointer to DIB bits
        BOOL     bSuccess=FALSE;      // Success/fail flag
        HPALETTE hPal=NULL;           // Our DIB's palette
        HPALETTE hOldPal=NULL;        // Previous palette

        /* Check for valid DIB handle */
        if (hDIB == NULL)
                return FALSE;

        /* Lock down the DIB, and get a pointer to the beginning of the bit
         *  buffer
         */
        lpDIBHdr  = (LPSTR) ::GlobalLock((HGLOBAL) hDIB);
        lpDIBBits = ::FindDIBBits(lpDIBHdr);

        // Get the DIB's palette, then select it into DC
        if (pPal != NULL)
        {
                hPal = (HPALETTE) pPal->m_hObject;

                // Select as background since we have
                // already realized in forground if needed
                hOldPal = ::SelectPalette(hDC, hPal, TRUE);
        }

        /* Make sure to use the stretching mode best for color pictures */
        ::SetStretchBltMode(hDC, COLORONCOLOR);

        /* Determine whether to call StretchDIBits() or SetDIBitsToDevice() */
        if ((RECTWIDTH(lpDCRect)  == RECTWIDTH(lpDIBRect)) &&
           (RECTHEIGHT(lpDCRect) == RECTHEIGHT(lpDIBRect)))
                bSuccess = ::SetDIBitsToDevice(hDC,                     // hDC
                                                        lpDCRect->left,             // DestX
                                                        lpDCRect->top,              // DestY
                                                        RECTWIDTH(lpDCRect),        //
nDestWidth
                                                        RECTHEIGHT(lpDCRect),       //
nDestHeight
                                                        lpDIBRect->left,            // SrcX
                                                        (int)DIBHeight(lpDIBHdr) -
                                                              lpDIBRect->top -
                                                              RECTHEIGHT(lpDIBRect),   //
SrcY
                                                        0,                          //
nStartScan
                                                        (WORD)DIBHeight(lpDIBHdr),  //
nNumScans
                                                        lpDIBBits,                  //
lpBits
                                                        (LPBITMAPINFO)lpDIBHdr,     //
lpBitsInfo
                                                        DIB_RGB_COLORS);            //
wUsage
   else
         bSuccess = ::StretchDIBits(hDC,                        // hDC
                                                lpDCRect->left,                 // DestX
                                                lpDCRect->top,                  // DestY
                                                RECTWIDTH(lpDCRect),            //
nDestWidth
                                                RECTHEIGHT(lpDCRect),           //
nDestHeight
                                                lpDIBRect->left,                // SrcX
                                                lpDIBRect->top,                 // SrcY
                                                RECTWIDTH(lpDIBRect),           //
wSrcWidth
```

```
                                                        RECTHEIGHT(lpDIBRect),          //
wSrcHeight
                                                        lpDIBBits,                      // lpBits
                                                        (LPBITMAPINFO)lpDIBHdr,         //
lpBitsInfo
                                                        DIB_RGB_COLORS,                 // wUsage
                                                        SRCCOPY);                       // dwROP

   ::GlobalUnlock((HGLOBAL) hDIB);

       /* Reselect old palette */
       if (hOldPal != NULL)
       {
               ::SelectPalette(hDC, hOldPal, TRUE);
       }

   return bSuccess;
}

/*************************************************************************
 *
 * CreateDIBPalette()
 *
 * Parameter:
 *
 * HDIB hDIB        - specifies the DIB
 *
 * Return Value:
 *
 * HPALETTE         - specifies the palette
 *
 * Description:
 *
 * This function creates a palette from a DIB by allocating memory for the
 * logical palette, reading and storing the colors from the DIB's color table
 * into the logical palette, creating a palette from this logical palette,
 * and then returning the palette's handle. This allows the DIB to be
 * displayed using the best possible colors (important for DIBs with 256 or
 * more colors).
 *
 *************************************************************************/
BOOL WINAPI CreateDIBPalette(HDIB hDIB, CPalette* pPal)
{
       LPLOGPALETTE lpPal;      // pointer to a logical palette
       HANDLE hLogPal;          // handle to a logical palette
       HPALETTE hPal = NULL;    // handle to a palette
       int i;                   // loop index
       WORD wNumColors;         // number of colors in color table
       LPSTR lpbi;              // pointer to packed-DIB
       LPBITMAPINFO lpbmi;      // pointer to BITMAPINFO structure (Win3.0)
       LPBITMAPCOREINFO lpbmc;  // pointer to BITMAPCOREINFO structure (old)
       BOOL bWinStyleDIB;       // flag which signifies whether this is a Win3.0 DIB
       BOOL bResult = FALSE;

       /* if handle to DIB is invalid, return FALSE */

       if (hDIB == NULL)
         return FALSE;

   lpbi = (LPSTR) ::GlobalLock((HGLOBAL) hDIB);

   /* get pointer to BITMAPINFO (Win 3.0) */
   lpbmi = (LPBITMAPINFO)lpbi;

   /* get pointer to BITMAPCOREINFO (old 1.x) */
   lpbmc = (LPBITMAPCOREINFO)lpbi;

   /* get the number of colors in the DIB */
   wNumColors = ::DIBNumColors(lpbi);

   if (wNumColors != 0)
   {
               /* allocate memory block for logical palette */
```

```
                /* allocate memory block for logical palette */
                hLogPal = ::GlobalAlloc(GHND, sizeof(LOGPALETTE)
                                                        + sizeof(PALETTEENTRY)
                                                        * wNumColors);

                /* if not enough memory, clean up and return NULL */
                if (hLogPal == 0)
                {
                        ::GlobalUnlock((HGLOBAL) hDIB);
                        return FALSE;
                }

                lpPal = (LPLOGPALETTE) ::GlobalLock((HGLOBAL) hLogPal);

                /* set version and number of palette entries */
                lpPal->palVersion = PALVERSION;
                lpPal->palNumEntries = (WORD)wNumColors;

                /* is this a Win 3.0 DIB? */
                bWinStyleDIB = IS_WIN30_DIB(lpbi);
                for (i = 0; i < (int)wNumColors; i++)
                {
                        if (bWinStyleDIB)
                        {
                                lpPal->palPalEntry[i].peRed = lpbmi->bmiColors[i].rgbRed;
                                lpPal->palPalEntry[i].peGreen = lpbmi->bmiColors[i].rgbGreen;
                                lpPal->palPalEntry[i].peBlue = lpbmi->bmiColors[i].rgbBlue;
                                lpPal->palPalEntry[i].peFlags = 0;
                        }
                        else
                        {
                                lpPal->palPalEntry[i].peRed = lpbmc->bmciColors[i].rgbtRed;
                                lpPal->palPalEntry[i].peGreen = lpbmc->bmciColors[i].rgbtGreen;
                                lpPal->palPalEntry[i].peBlue = lpbmc->bmciColors[i].rgbtBlue;
                                lpPal->palPalEntry[i].peFlags = 0;
                        }
                }

                /* create the palette and get handle to it */
                bResult = pPal->CreatePalette(lpPal);
                ::GlobalUnlock((HGLOBAL) hLogPal);
                ::GlobalFree((HGLOBAL) hLogPal);
        }

        ::GlobalUnlock((HGLOBAL) hDIB);

        return bResult;
}

/*************************************************************************
 *
 * FindDIBBits()
 *
 * Parameter:
 *
 * LPSTR lpbi       - pointer to packed-DIB memory block
 *
 * Return Value:
 *
 * LPSTR            - pointer to the DIB bits
 *
 * Description:
 *
 * This function calculates the address of the DIB's bits and returns a
 * pointer to the DIB bits.
 *
 *************************************************************************/


LPSTR WINAPI FindDIBBits(LPSTR lpbi)
{
        return (lpbi + *(LPDWORD)lpbi + ::PaletteSize(lpbi));
}
```

```
/***************************************************************************
 *
 * DIBWidth()
 *
 * Parameter:
 *
 * LPSTR lpbi       - pointer to packed-DIB memory block
 *
 * Return Value:
 *
 * DWORD            - width of the DIB
 *
 * Description:
 *
 * This function gets the width of the DIB from the BITMAPINFOHEADER
 * width field if it is a Windows 3.0-style DIB or from the BITMAPCOREHEADER
 * width field if it is an other-style DIB.
 *
 ***************************************************************************/
DWORD WINAPI DIBWidth(LPSTR lpDIB)
{
        LPBITMAPINFOHEADER lpbmi;  // pointer to a Win 3.0-style DIB
        LPBITMAPCOREHEADER lpbmc;  // pointer to an other-style DIB

        /* point to the header (whether Win 3.0 and old) */

        lpbmi = (LPBITMAPINFOHEADER)lpDIB;
        lpbmc = (LPBITMAPCOREHEADER)lpDIB;

        /* return the DIB width if it is a Win 3.0 DIB */
        if (IS_WIN30_DIB(lpDIB))
                return lpbmi->biWidth;
        else  /* it is an other-style DIB, so return its width */
                return (DWORD)lpbmc->bcWidth;
}

/***************************************************************************
 *
 * DIBHeight()
 *
 * Parameter:
 *
 * LPSTR lpbi       - pointer to packed-DIB memory block
 *
 * Return Value:
 *
 * DWORD            - height of the DIB
 *
 * Description:
 *
 * This function gets the height of the DIB from the BITMAPINFOHEADER
 * height field if it is a Windows 3.0-style DIB or from the BITMAPCOREHEADER
 * height field if it is an other-style DIB.
 *
 ***************************************************************************/
DWORD WINAPI DIBHeight(LPSTR lpDIB)
{
        LPBITMAPINFOHEADER lpbmi;  // pointer to a Win 3.0-style DIB
        LPBITMAPCOREHEADER lpbmc;  // pointer to an other-style DIB

        /* point to the header (whether old or Win 3.0 */

        lpbmi = (LPBITMAPINFOHEADER)lpDIB;
        lpbmc = (LPBITMAPCOREHEADER)lpDIB;

        /* return the DIB height if it is a Win 3.0 DIB */
        if (IS_WIN30_DIB(lpDIB))
                return lpbmi->biHeight;
        else  /* it is an other-style DIB, so return its height */
                return (DWORD)lpbmc->bcHeight;
}
```

```
/*************************************************************************
 *
 * PaletteSize()
 *
 * Parameter:
 *
 * LPSTR lpbi       - pointer to packed-DIB memory block
 *
 * Return Value:
 *
 * WORD             - size of the color palette of the DIB
 *
 * Description:
 *
 * This function gets the size required to store the DIB's palette by
 * multiplying the number of colors by the size of an RGBQUAD (for a
 * Windows 3.0-style DIB) or by the size of an RGBTRIPLE (for an other-
 * style DIB).
 *
 *************************************************************************/
WORD WINAPI PaletteSize(LPSTR lpbi)
{
   /* calculate the size required by the palette */
   if (IS_WIN30_DIB (lpbi))
         return (WORD)(::DIBNumColors(lpbi) * sizeof(RGBQUAD));
   else
         return (WORD)(::DIBNumColors(lpbi) * sizeof(RGBTRIPLE));
}
/*************************************************************************
 *
 * DIBNumColors()
 *
 * Parameter:
 *
 * LPSTR lpbi       - pointer to packed-DIB memory block
 *
 * Return Value:
 *
 * WORD             - number of colors in the color table
 *
 * Description:
 *
 * This function calculates the number of colors in the DIB's color table
 * by finding the bits per pixel for the DIB (whether Win3.0 or other-style
 * DIB). If bits per pixel is 1: colors=2, if 4: colors=16, if 8: colors=256,
 * if 24, no colors in color table.
 *
 *************************************************************************/
DWORD WINAPI DIBNumColors(LPSTR lpbi)
{
       DWORD wBitCount;  // DIB bit count

       /*  If this is a Windows-style DIB, the number of colors in the
        *  color table can be less than the number of bits per pixel
        *  allows for (i.e. lpbi->biClrUsed can be set to some value).
        *  If this is the case, return the appropriate value.
        */

       if (IS_WIN30_DIB(lpbi))
       {
              DWORD dwClrUsed;

              dwClrUsed = ((LPBITMAPINFOHEADER)lpbi)->biClrUsed;
              if (dwClrUsed != 0)
                     return (WORD)dwClrUsed;
       }

       /*  Calculate the number of colors in the color table based on
        *  the number of bits per pixel for the DIB.
        */
       if (IS_WIN30_DIB(lpbi))
              wBitCount = ((LPBITMAPINFOHEADER)lpbi)->biBitCount;
       else
```

```
        else
                wBitCount = ((LPBITMAPCOREHEADER)lpbi)->bcBitCount;

        /* return number of colors based on bits per pixel */
        switch (wBitCount)
        {
                case 1:
                        return 2;

                case 4:
                        return 16;

                case 8:
                        return 256;

                case 16:
                        return 65536;

                case 24:
                        return 16777216;

                default:
                        return 0;
        }
}

DWORD WINAPI DIBNumBPP(LPSTR lpbi)
{
        DWORD wBitCount;  // DIB bit count

        /*  If this is a Windows-style DIB, the number of colors in the
         *  color table can be less than the number of bits per pixel
         *  allows for (i.e. lpbi->biClrUsed can be set to some value).
         *  If this is the case, return the appropriate value.
         */

        if (IS_WIN30_DIB(lpbi))
        {
                DWORD dwClrUsed;

                dwClrUsed = ((LPBITMAPINFOHEADER)lpbi)->biClrUsed;
                if (dwClrUsed != 0)
                        return (WORD)dwClrUsed;
        }

        /*  Calculate the number of colors in the color table based on
         *  the number of bits per pixel for the DIB.
         */
        if (IS_WIN30_DIB(lpbi))
                wBitCount = ((LPBITMAPINFOHEADER)lpbi)->biBitCount;
        else
                wBitCount = ((LPBITMAPCOREHEADER)lpbi)->bcBitCount;

        /* return bits per pixel */
        return wBitCount;
}

////////////////////////////////////////////////////////////////////////////
//// Clipboard support

//-------------------------------------------------------------------------
//
// Function:   CopyHandle (from SDK DibView sample clipbrd.c)
//
// Purpose:    Makes a copy of the given global memory block.  Returns
//             a handle to the new memory block (NULL on error).
//
//             Routine stolen verbatim out of ShowDIB.
//
// Parms:      h == Handle to global memory to duplicate.
//
// Returns:    Handle to new global memory block.
//
```

```
//-----------------------------------------------------------------------
HGLOBAL WINAPI CopyHandle (HGLOBAL h)
{
        if (h == NULL)
                return NULL;

        DWORD dwLen = ::GlobalSize((HGLOBAL) h);
        HGLOBAL hCopy = ::GlobalAlloc(GHND, dwLen);

        if (hCopy != NULL)
        {
                void* lpCopy = ::GlobalLock((HGLOBAL) hCopy);
                void* lp     = ::GlobalLock((HGLOBAL) h);
                memcpy(lpCopy, lp, dwLen);
                ::GlobalUnlock(hCopy);
                ::GlobalUnlock(h);
        }

        return hCopy;
}

//  myfile.cpp
//
//  Source file for Device-Independent Bitmap (DIB) API.  Provides
//  the following functions:
//
//  SaveDIB()           - Saves the specified dib in a file
//  ReadDIBFile()       - Loads a DIB from a file
//
//
// This is a part of the Microsoft Foundation Classes C++ library.
// Copyright (C) 1992-1998 Microsoft Corporation
// All rights reserved.
//
// This source code is only intended as a supplement to the
// Microsoft Foundation Classes Reference and related
// electronic documentation provided with the library.
// See these sources for detailed information regarding the
// Microsoft Foundation Classes product.

#include "stdafx.h"
#include <math.h>
#include <io.h>
#include <direct.h>
#include "dibapi.h"

/*
 * Dib Header Marker - used in writing DIBs to files
 */
#define DIB_HEADER_MARKER   ((WORD) ('M' << 8) | 'B')

/*************************************************************************
 *
 * SaveDIB()
 *
 * Saves the specified DIB into the specified CFile.  The CFile
 * is opened and closed by the caller.
 *
 * Parameters:
 *
 * HDIB hDib - Handle to the dib to save
 *
 * CFile& file - open CFile used to save DIB
 *
 * Return value: TRUE if successful, else FALSE or CFileException
 *
 *************************************************************************/
BOOL WINAPI SaveDIB(HDIB hDib, CFile& file)
{
        BITMAPFILEHEADER bmfHdr; // Header for Bitmap file
        LPBITMAPINFOHEADER lpBI;   // Pointer to DIB info structure
        DWORD dwDIBSize;
```

```
        DWORD dwDIBSize;

        if (hDib == NULL)
                return FALSE;

        /*
         * Get a pointer to the DIB memory, the first of which contains
         * a BITMAPINFO structure
         */
        lpBI = (LPBITMAPINFOHEADER) ::GlobalLock((HGLOBAL) hDib);
        if (lpBI == NULL)
                return FALSE;

        if (!IS_WIN30_DIB(lpBI))
        {
                ::GlobalUnlock((HGLOBAL) hDib);
                return FALSE;       // It's an other-style DIB (save not supported)
        }

        /*
         * Fill in the fields of the file header
         */

        /* Fill in file type (first 2 bytes must be "BM" for a bitmap) */
        bmfHdr.bfType = DIB_HEADER_MARKER;  // "BM"

        // Calculating the size of the DIB is a bit tricky (if we want to
        // do it right).  The easiest way to do this is to call GlobalSize()
        // on our global handle, but since the size of our global memory may have
        // been padded a few bytes, we may end up writing out a few too
        // many bytes to the file (which may cause problems with some apps).
        //
        // So, instead let's calculate the size manually (if we can)
        //
        // First, find size of header plus size of color table.  Since the
        // first DWORD in both BITMAPINFOHEADER and BITMAPCOREHEADER conains
        // the size of the structure, let's use this.

        dwDIBSize = *(LPDWORD)lpBI + ::PaletteSize((LPSTR)lpBI);  // Partial Calculation

        // Now calculate the size of the image

        if ((lpBI->biCompression == BI_RLE8) || (lpBI->biCompression == BI_RLE4))
        {
                // It's an RLE bitmap, we can't calculate size, so trust the
                // biSizeImage field

                dwDIBSize += lpBI->biSizeImage;
        }
        else
        {
                DWORD dwBmBitsSize;  // Size of Bitmap Bits only

                // It's not RLE, so size is Width (DWORD aligned) * Height

                dwBmBitsSize = WIDTHBYTES((lpBI->biWidth)*((DWORD)lpBI->biBitCount)) * lpBI-
>biHeight;

                dwDIBSize += dwBmBitsSize;

                // Now, since we have calculated the correct size, why don't we
                // fill in the biSizeImage field (this will fix any .BMP files which
                // have this field incorrect).

                lpBI->biSizeImage = dwBmBitsSize;
        }


        // Calculate the file size by adding the DIB size to sizeof(BITMAPFILEHEADER)

        bmfHdr.bfSize = dwDIBSize + sizeof(BITMAPFILEHEADER);
        bmfHdr.bfReserved1 = 0;
        bmfHdr.bfReserved2 = 0;
```

```
        /*
         * Now, calculate the offset the actual bitmap bits will be in
         * the file -- It's the Bitmap file header plus the DIB header,
         * plus the size of the color table.
         */
        bmfHdr.bfOffBits = (DWORD)sizeof(BITMAPFILEHEADER) + lpBI->biSize
                                                                                +
PaletteSize((LPSTR)lpBI);
        TRY
        {
                // Write the file header
                file.Write((LPSTR)&bmfHdr, sizeof(BITMAPFILEHEADER));
                //
                // Write the DIB header and the bits
                //
                file.WriteHuge(lpBI, dwDIBSize);
        }
        CATCH (CFileException, e)
        {
                ::GlobalUnlock((HGLOBAL) hDib);
                THROW_LAST();
        }
        END_CATCH

        ::GlobalUnlock((HGLOBAL) hDib);
        return TRUE;
}

/*************************************************************************
  Function:  ReadDIBFile (CFile&)
   Purpose:  Reads in the specified DIB file into a global chunk of
                     memory.
   Returns:  A handle to a dib (hDIB) if successful.
                     NULL if an error occurs.
  Comments:  BITMAPFILEHEADER is stripped off of the DIB.  Everything
                     from the end of the BITMAPFILEHEADER structure on is
                     returned in the global memory handle.
*************************************************************************/
HDIB WINAPI ReadDIBFile(CFile& file)
{
        BITMAPFILEHEADER bmfHeader;
        DWORD dwBitsSize;
        HDIB hDIB;
        LPSTR pDIB;

        /*
         * get length of DIB in bytes for use when reading
         */

        dwBitsSize = file.GetLength();

        /*
         * Go read the DIB file header and check if it's valid.
         */
        if (file.Read((LPSTR)&bmfHeader, sizeof(bmfHeader)) != sizeof(bmfHeader))
                return NULL;

        if (bmfHeader.bfType != DIB_HEADER_MARKER)
                return NULL;

        /*
         * Allocate memory for DIB
         */
        hDIB = (HDIB) ::GlobalAlloc(GMEM_MOVEABLE | GMEM_ZEROINIT, dwBitsSize);
        if (hDIB == 0)
        {
                return NULL;
        }
        pDIB = (LPSTR) ::GlobalLock((HGLOBAL) hDIB);

        /*
         * Go read the bits.
         */
```

```
        if (file.ReadHuge(pDIB, dwBitsSize - sizeof(BITMAPFILEHEADER)) !=
                dwBitsSize - sizeof(BITMAPFILEHEADER) )
        {
                ::GlobalUnlock((HGLOBAL) hDIB);
                ::GlobalFree((HGLOBAL) hDIB);
                return NULL;
        }
        ::GlobalUnlock((HGLOBAL) hDIB);
        return hDIB;
}
```

Annex 2. Edició i compilació dels programes

En aquest annex es descriu la manera com s'ha organitzat en disc el desenvolupament dels programes, on s'obtenen els executables finals i el procés de compilació i obtenció dels mateixos.

1.1. Estructura del projecte

Els productes s'ha desenvolupat en un únic projecte de Microsoft Visual C 6.0, on hi ha dos espais de treball. Un correspon al programa que genera i recupera la marca, i l'altre correspon al programa de les confabulacions.

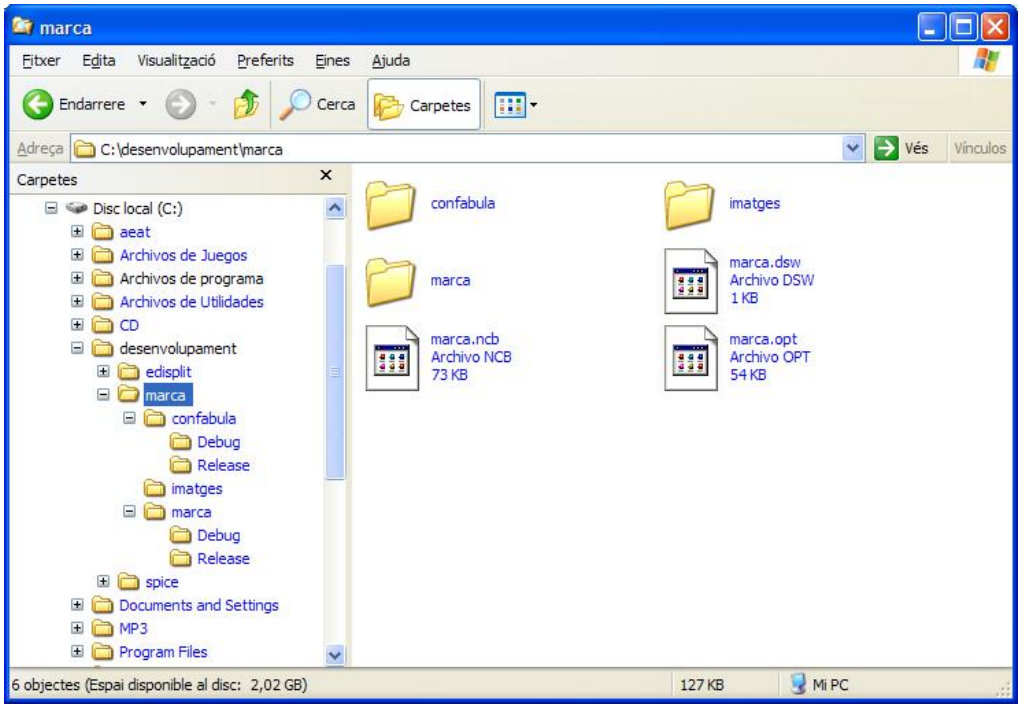
En l'entorn gràfic del Visual C s'observen els programes font que composen l'aplicació. Es pot efectuar l'edició del codi, la compilació i el procés de prova pas a pas o amb execució complerta.

En el projecte es poden especificar els paràmetres que es asaran el programa en el moment de la seva execució, si es vol comprovar el funcionament pas a pas. Per fer-ho s'ha d'anar a *Project* >> *Settings* >> seleccionar el producte >> *debug* >> *parameters*

Es tracta d'una aplicació nativa de 32 bits, que segons les especificacions del projecte funciona en mode consola. Incorpora el suport de les *Microsoft Foundation Classes* (el *framework* de Microsoft per a desenvolupar aplicacions que funcionen en Windows).

1.2. Distribució del desenvolupament al disc

La distribució dels arxius a disc ha estat la següent:



Imatge 12: Distribució del desenvolupament en directoris al disc

Dins el directori de desenvolupament hi trobem el subdirectori marca, on hi ha el projecte de Visual C. Dins aquest directori hi ha els dos subdirectoris que contenen els *workspace* de Visual C corresponents als dos productes (marca i confabula). Hi ha un tercer directori (imatges) on es guarden totes les imatges originals i creades en les proves (sense marcar i marcades).

Dins els directoris dels produtes trobem les dues versions que crea habitualment el Visual C: *Debug* (conté el producte amb compilació per a fer proves, amb informació addicional i només executable en l'entorn de desenvolupament) i *Release* (que conté la versió definitiva i executable a qualsevol sistema Windows).

Els productes definitius es troben, per tant, als directoris:

- c:\desenvolupament\marca\marca\release\marca.exe i
- c:\desenvolupament\marca\confabula\release\confabula.exe

Annex 3. Informació sobre les aplicacions auxiliars

En el transcurs del procés de creació s'han usat les aplicacions auxiliars que es relacionen a continuació:

Aplicació	**Versió**	**Utilitat**
Microsoft Visual C	6.0	Desenvolupament del producte
Microsoft Paint	5.1	Edició i escalat de les imatges
Adobe Photodeluxe	3.0	Compressió JPEG amb % de qualitat
Microsoft Word	10.6612.6626	Edició de la memòria
Microsoft PowerPoint	10.6501.6626	Presentació en diapositives
Microsoft Windows	XP	Proves del producte i edició de la memòria
Microsoft Windows	2000	Desenvolupament del producte
Microsoft DibLook	1.0	Aplicació d'exemple per a la visualització d'imatges bitmap (codi font reutilitzable)
Microsoft WinDiff	5.1	Aplicació que permet comparar arxius visualment. Utilitat del Visual C.

Taula 9: Taula d'aplicacions auxiliars