Universitat de Girona
Escola Politècnica Superior

Projecte/Treball Final de Carrera

Estudi: Eng. Tècn. Informàtica de Gestió. Pla 1993

Títol: Portabilitat d'aplicacions de PC a PDA mitjançant un cas d'estudi: ScummVM

Document: Memòria

Alumne: Marc Coll Passolas

Director/Tutor: Gustavo Patow
Departament: Informàtica i Matemàtica Aplicada
Àrea: LSI

Convocatòria (mes/any): 09/2006

1 Introducció

1.1 Presentació de la problemàtica

Els assistents personals digitals, PDA i els telèfons mòbils, són cada dia més freqüents, arribant a igualar a ordinadors molt més potents en quant a velocitat i versatilitat, dissenyats originalment amb una finalitat completament diferent. Però tot i així, encara existeix una gran quantitat de *software* que no ha estat desenvolupat per aquests dispositius, però que són molt reconeguts en l'àmbit dels ordinadors personals o PC.

Aquests dispositius tenen una sèrie de limitacions en comparació als ordinadors personals. A continuació mostrem una llista de les limitacions més habituals:

- Pantalla petita i de baixa resolució.
- Pantalla amb poca profunditat de color (Escala de grisos, 16 colors, 256 colors, …)
- Absència de dispositius d'entrada i sortida habituals com són el teclat, el ratolí, disqueteres i altres dispositius d'entrada i sortida. En la majoria dels casos només trobem uns botons i una pantalla tàctil, sobre la qual l'usuari pot escriure mitjançant un llapis, propi de cada dispositiu.
- Memòria limitada, amb mides que van des de 1Mb fins a 128 Mb.
- Processadors lents en comparació als de PC.
- …

Totes aquestes diferències amb els PC's fan que el fet de portar una aplicació que originalment ha estat dissenyada per un PC, sigui una feina molt pesada i que en la majoria dels casos s'opti per refer l'aplicació començant des de zero.

Dintre del món dels dispositius mòbils, existeix una gran quantitat de dispositius diferents. Actualment la majoria funcionen amb *Win CE*, una versió específica per dispositius portàtils del conegut sistema operatiu de *Microsoft* o bé amb *Symbian*, molt utilitzat en els telèfons mòbils. Però, tot i això, la marca pionera en els dispositius mòbils anomenats PDA ha estat Palm. Nosaltres hem escollit un dispositiu amb el sistema operatiu *Palm OS* per a la realització de l'estudi. El nostre dispositiu presenta la majoria de les limitacions abans esmentades i algunes més, però per la realització del nostre estudi ens centrarem en les següents:

- Pantalla petita, de baixa resolució (160x160) i amb poca profunditat de color (escala de grisos, 16 colors, 256 colors).
- Absència de dispositius d'entrada i sortida com el teclat, el ratolí, disqueteres 6 botons i una pantalla tàctil, sobre la qual l'usuari pot escriure mitjançant un llapis.
- Aparició d'un nou tipus de fitxer anomenats "bases de dades" o fitxers "pdb" propis de *Palm OS*.

1.2 Objectiu

L'objectiu del treball és estudiar la portabilitat dels programes que han estat originalment dissenyats per un PC a un dispositiu mòbil, en concret a un PDA amb el sistema operatiu *Palm OS*. Per fer això hem escollit una aplicació que ens permet donar solucions a les limitacions que presenta el dispositiu amb *Palm OS* que utilitzarem i que ja hem esmentat en l'apartat anterior.

Els fabricants de software tenen la lliçó ben apresa i per evitar-se problemes posen sempre a la caixa dels seus productes, dues llistes, una conté la llista de requeriments mínims que necessita l'aplicació per funcionar i l'altre conté la llista de requeriments òptims amb els quals l'aplicació funcionarà de la millor manera.

Si ens fixem en aquestes llistes hi veurem les principals característiques que marquen la compatibilitat d'un programa amb un sistema solen ser les següents:

- Sistema operatiu.
- Velocitat del processador.
- Memòria RAM instal·lada.
- Targeta de vídeo necessària.
- Targeta de so.
- Espai lliure en el disc dur.

D'aquesta manera ens adonem que un processador de text no necessita la mateixa velocitat de processador i de targeta de vídeo com per exemple necessitaria un joc d'ordinador. O que un programa ha de treballar amb un volum de dades molt alt necessitarà molta més memòria RAM que un altre que no hagi de fer.

Tenint en compte tot això i el fet que cada aplicació és un món ens trobarem que hi ha aplicacions que seran portables més fàcilment entre dispositius que d'altres.

Per al nostre projecte hem intentat escollir una aplicació que ens presentés la majoria de diferències entre els seus requeriments per funcionar correctament i les prestacions d'una PALM.

L'aplicació que nosaltres utilitzarem per estudiar i intentar solucionar aquests problemes és la coneguda *ScummVM*, que es tracta d'una implementació de codi lliure del conegut motor de videojocs *SCUMM*, creat i utilitzat per *LucasArts* en les seves aventures gràfiques. El que fa l'*ScummVM* és agafar els fitxers de dades dels jocs originals i interpretar-les.

S'ha triat aquesta aplicació particular, *ScummVM*, perquè és un motor general per a un conjunt de videojocs, els que ens presenten tot tipus de problemes de compatibilitat a l'hora de portar-los a una altra plataforma, donat que tenen elevades exigències gràfiques així com texts i tot tipus d'elements de menú.

Donarem possibles solucions a la problemàtica que genera el fet que la pantalla del dispositiu mòbil utilitzat tingui una resolució de 160x160 a l'hora de mostrar imatges i textos *renderitzats* per una resolució superior, també solucionarem el problema que representa el fet de no tenir ratolí, ni altres dispositius d'entrada i sortida tradicionals. La intenció és explicar també el procés que s'hauria de seguir per poder portar aplicacions de PC a *Palm OS* i les eines que s'utilitzarien. Tot això es farà de la manera menys invasiva possible, és a dir, els canvis al codi font de l'aplicació original seran els mínims per garantir el correcte funcionament del programa en la nova plataforma. També cal dir, que per implementar l'aplicació és respectaran les guies de programació (*code formatting conventions*) del projecte *ScummVM*.

1.3 Treball previ i temporització del projecte

Per la realització del projecte he hagut de documentar-me i aprendre a utilitzar eines específiques per al desenvolupament de software per *Palm OS*, a més a més del fet d'haver d'entendre el funcionament de l'aplicació escollida pel projecte. He dividit aquestes tasques en les següents:

- Familiarització amb el sistema operatiu *Palm OS.*
- Estudi de l'*ScummVM.*
- Familiarització amb el *Palm OS* SDK i el compilador *Metrowerk's Codewarrior*.
- Disseny i implementació de l'aplicació per PC.
- Disseny i implementació de l'aplicació per *PalmOS*.

- Documentació.

A la figura següent podem veure una taula amb el temps aproximat que hem dedicat a cada apartat en dies. Tenint en compte que cada dia representa unes 8 hores. El càlcul dels dies s'ha fet de manera aproximada, ja que no s'ha seguit un control exhaustiu sobre les hores dedicades.

Tasca	**Temps dedicat**
Familiarització amb el sistema operatiu *Palm OS.*	10 dies.
Estudi de l'*ScummVM*	45 dies.
Familiarització amb el *Palm OS* SDK i el compilador *Metrowerk's Codewarrior*.	7 dies.
Disseny i implementació de l'aplicació per PC.	60 dies.
Disseny i implementació de l'aplicació per *PalmOS*.	105 dies.
Documentació.	70 dies.

Fig 1.1 Taula del temps aproximat que s'ha dedicat a cada tasca.

1.4 Metodologia utilitzada: UML

Tot i que l'UML no és una metodologia, nosaltres utilitzarem el llenguatge que ens proporciona per il·lustrar i fer més entenedor el funcionament i l'arquitectura tant de la nostra aplicació com de l'*ScummVM*.

1.4.1 Introducció a l'UML

Unified Modeling Language .- Llenguatge unificat de modelat, és un llenguatge gràfic per visualitzar, especificar, construir i documentar un sistema de software.

Objectius de l'UML

Els principals objectius d'aquest llenguatge són els següents:

- UML és un llenguatge de propòsit general que poden utilitzar tots els modeladors. No té propietari i està basat en l'acord comú de gran part de la comunitat informàtica.
- UML no pretén ser una eina de desenvolupament complet. No inclou un procés de desenvolupament pas a pas. UML inclou tots els conceptes que es consideren necessaris per utilitzar un procés modern iteratiu, basat en construir una sòlida arquitectura per resoldre requisits dirigits per casos d'ús.
- Ser tant simple com sigui possible però mantenint la capacitat de modelar tota la gamma de sistemes que es necessiti construir. UML necessita ser suficientment expressiu com per utilitzar tots els conceptes que s'originen en un sistema modern, tals com la concurrència, la distribució i també els mecanismes de l'enginyeria de software, com l'encapsulament de components.
- Ha de ser un llenguatge universal, com qualsevol llenguatge de propòsit general.
- Imposar un estàndard mundial.

Arquitectura de l'UML

Cal destacar de l'arquitectura el fet que estigui format per quatre capes, amb la finalitat de complir amb l'especificació Meta Object Facility de l'OMG (Object Management Group). Les capes són les següents:

- **Meta-metamodel.** Defineix el llenguatge per especificar metamodels.
- **Metamodel.** Defineix el llenguatge per especificar models.
- **Model.** Defineix el llenguatge per descriure un domini d'informació.
- **Objectes d'usuari.** Defineix un domini d'informació específica.

Àrees conceptuals

Els conceptes i models de l'UML es poden agrupar en les àrees conceptuals següents:

- **Estructura estàtica.** Qualsevol model precís ha de definir el seu univers en primera instància, això és els conceptes claus de l'aplicació, les seves propietats internes i les relacions entre cadascuna d'elles. Aquest conjunt de construccions és l'estructura estàtica.

Els conceptes de l'aplicació són modelats com classes, cadascuna de les quals descriu un conjunt d'objectes que emmagatzemen la informació i es comuniquen per implementar el comportament. La informació que emmagatzema és modelada com atributs. L'estructura estàtica s'expressa amb diagrames de classes i es pot utilitzar per generar la majoria de les declaracions d'estructures de dades en un programa.

- **Comportament dinàmic.** Hi ha dues formes de modelar el comportament, una és la història de la vida d'un objecte i la forma com interactua amb la resta del món i l'altre és mitjançant els patrons de comunicació d'un conjunt d'objectes connectats, és a dir, la forma en la qual interactuen entre sí. La visió de la interacció dels objectes es representa amb els enllaços entre objectes juntament amb el flux dels missatges i els enllaços entre ells. Aquest punt de vista unifica l'estructura de dades, el control de flux i el flux de dades.
- **Construccions d'implementació.** Els models UML tenen significat per l'anàlisi lògica i per la implementació física. Un component és una part física reemplaçable d'un sistema i que és capaç de respondre a les peticions descrites per un conjunt determinat d'interfícies. Un node és un recurs computacional que defineix una localització durant l'execució d'un sistema. Pot contenir components i objectes.
- **Organització del model.** La informació del model ha de ser dividida en parts coherents, perquè els equips puguin treballar en les diferents parts de manera concurrent. El coneixement humà requereix que s'organitzi el contingut del model en paquets de mida modesta. Els paquets són unitats organitzatives, jeràrquiques i de propòsit general dels models UML. Es poden utilitzar per emmagatzemar, control d'accés, gestió de la configuració i construcció de biblioteques que continguin fragments de codi reutilitzable.
- **Mecanismes d'extensió.** L'UML té una limitada capacitat d'extensió, però que és suficient per la majoria de les extensions que es requereixen durant el dia a dia sense la necessitat d'un canvi en el llenguatge Bàsic. Un estereotip és una nova classe d'element de modelat amb la mateixa estructura que un element existent però amb restriccions addicionals.

1.4.2 El perquè de la utilització de l'UML

A l'hora de programar la nostra aplicació hem utilitzat l'estàndard del llenguatge de programació UML com a guia, degut a què l'aplicació de la qual partim està dissenyada seguint aquest estàndard. Això vol dir que tots els dissenys inclosos en la memòria segueixen aquest llenguatge.

1.5 Estructura del document

Aquesta memòria està estructurada en els capítols següents:

- **Capítol 1: Introducció.** En aquest capítol s'expliquen els objectius i la temàtica del projecte. També és dóna una visió del treball previ realitzat i una descripció de la metodologia utilitzada.
- **Capítol 2: Aplicacions i eines utilitzades.** En el capítol dos de la memòria oferim una descripció de les aplicacions i les eines utilitzades per al desenvolupament del projecte. Explicant els motius de l'elecció de cada aplicació i eina.
- **Capítol 3: *Palm* / *PalmOS.*** Aquí s'explica a grans trets el funcionament de les PDA que utilitzen el *PalmOS* com a sistema operatiu, aprofundint en els aspectes més rellevants pel nostre projecte.
- **Capítol 4: *ScummVM.*** En aquest capítol expliquem què són i com funcionen l'*SCUMM* i l'*ScummVM*. Expliquem la seva arquitectura, el seu disseny i les seves estructures de dades més importants, donant una visió del seu funcionament.
- **Capítol 5: Estudi i implementació de la portabilitat.** Aquest capítol explica tota la feina realitzada en el projecte. L'estructura, disseny i estructures de dades de l'aplicació implementada. S'explica el procés realitzat per dur a terme la portabilitat i els problemes i solucions que s'han trobat durant tot el procés.
- **Capítol 6: Resultats i conclusions.** En el capítol set es donen a conèixer tots els resultats obtinguts i les conclusions que en podem treure.
- **Capítol 7: Treballs futurs i possibles millores.** Aquest capítol pretén finalitzar la memòria exposant els treballs futurs possibles i les millores que es podrien fer a l'aplicació.

2 Aplicacions i eines utilitzades

2.1 Introducció

En aquest capítol descriurem les eines i les aplicacions necessàries per al correcte desenvolupament del nostre projecte. També descriurem el procés de selecció d'aquestes eines, i els motius pels quals s'han escollit aquestes i no unes altres.

2.2 Procés de selecció

Per desenvolupar el projecte hem hagut de prendre certes decisions com la d'escollir la plataforma de l'entorn de treball així com el sistema operatiu i el compilador. Quan es tracta de desenvolupar una aplicació des de zero, per un dispositiu amb *Palm OS* trobem eines molt útils que ens permeten treballar sobre PC. Això fa que la feina de desenvolupar sigui molt més agradable que no pas fer-ho directament sobre el PDA ja que trobem un emulador i un simulador proporcionat pels propis creadors de *Palm OS* que ens emula/simula la màquina original de manera molt fiable. Per tant, l'elecció de plataforma ha estat senzilla, hem escollit treballar sobre PC. També existia la possibilitat de desenvolupar sobre *Mac OS,* però el fet de no disposar de cap màquina amb aquest sistema operatiu en el moment de començar el projecte va anul·lar aquesta possibilitat.

Un cop escollida la plataforma de treball havíem d'escollir quin sistema operatiu utilitzàvem. Dintre dels més coneguts per PC, trobem Linux i Windows. L'empresa que desenvolupa el *Palm OS* ens proporciona els SDK's (*Software Development Kit*) necessaris per treballar tant en Linux com en Windows, també existeix una versió de l'emulador per cada sistema operatiu, així doncs per decidir-nos vàrem mirar l'altre part del projecte, l'*ScummVM*.

Ens vàrem trobar que estava preparat per ser compilat pels dos sistemes operatius. Degut a això en un primer moment vàrem escollir el *Microsoft Visual Studio 6.0*, ja que el teníem instal·lat a les aules de la universitat, i era un dels compiladors amb els que jo personalment estava més familiaritzat. Tot i que en un primer moment creiem que aquest seria el compilador utilitzat durant tot el desenvolupament, ens vàrem trobar, que si hi havia una versió de l'SDK de *PalmOS* que s'instal·lava automàticament en el *Codewarrior*. També sabíem que els desenvolupadors de l'*ScummVM* havien estat treballant amb aquest compilador, i que el codi font incorporava el projecte per treballar-hi. Un cop descobert això vàrem consultar els compiladors oferts per l'empresa que ha creat el *Codewarrior* i ens vàrem trobar que hi havia una versió del compilador

pel *Palm OS* i que aquest mateix compilador es comunicava directament amb l'emulador de *Palm OS* oferint a la vegada un sistema de *debug* que vàrem trobar molt útil.

Tot això, juntament amb el fet que jo personalment em sentia més còmode desenvolupant sobre Windows i amb un compilador amb entorn gràfic, ens va convèncer que simplificaria el procés si es desenvolupava sobre Windows amb el *Codewarrior* per *Palm OS* i utilitzant l'emulador que la pròpia empresa Palm oferia del seu hardware. Vaig escollir treballar amb l'emulador en comptes d'un PDA perquè no en tenia cap a la meva disposició que pogués utilitzar.

Tot i així encara existia la possibilitat de fer servir el simulador, però en el moment de començar a desenvolupar aquest projecte no tenia suport de targetes de memòria, el que era indispensable per poder carregar els jocs que l'*ScummVM* utilitzava per funcionar.

Només quedava l'elecció del llenguatge de programació a utilitzar, que també va ser fàcil ja que l'*ScummVM* estava programat en C/C++ i l'SDK de Palm també, ens vàrem veure gairebé obligats a escollir aquest llenguatge.

Així que les eines utilitzades per desenvolupar el projecte són:

- Emulador de *Palm OS*
- *Palm OS* 5.0 SDK
- *Microsoft Visual Studio 6.0*
- *Metrowerk's Codewarrior 8.0* per *Palm OS*
- C/C++

2.3 Emulador de *Palm OS* (*POSE*) versió 3.5

Un emulador és un programa que permet executar programes en una plataforma diferent per la qual van ser escrits originalment. A diferència d'un simulador, que només reprodueix el comportament d'un programa, l'emulador intenta modelar de manera precisa al dispositiu que està emulant. El que fa és reproduir de manera exacta el funcionament del *hardware* original.

El nostre emulador de *Palm OS,* altrament anomenat *POSE* (*Palm OS Emulator*), és un programa que reprodueix el funcionament exacte del hardware d'una PDA de la marca Palm, d'arquitectura 68K i amb el sistema operatiu *Palm OS* instal·lat.

Aquest emulador funciona amb un sistema de ROMs. El que fa l'emulador és executar aquestes ROM. Físicament les ROM són arxius binaris que es poden carregar a la memòria. En el cas de la *Palm* les ROMs són còpies exactes del *hardware* d'una *Palm* en concret.

Existeixen ROMs per la majoria de *Palm* que funcionaven amb l'arquitectura esmentada anteriorment.

Quan executem l'emulador, ens dóna la possibilitat d'escollir entre els següents paràmetres del *hardware* de la màquina emulada:

- Aparença, mitjançant l'elecció d'una carcassa (*skin*).
- Versió de la ROM.
- Dispositiu o màquina.
- Quantitat de memòria RAM del dispositiu.

En la figura següent es poden apreciar aquests aspectes.

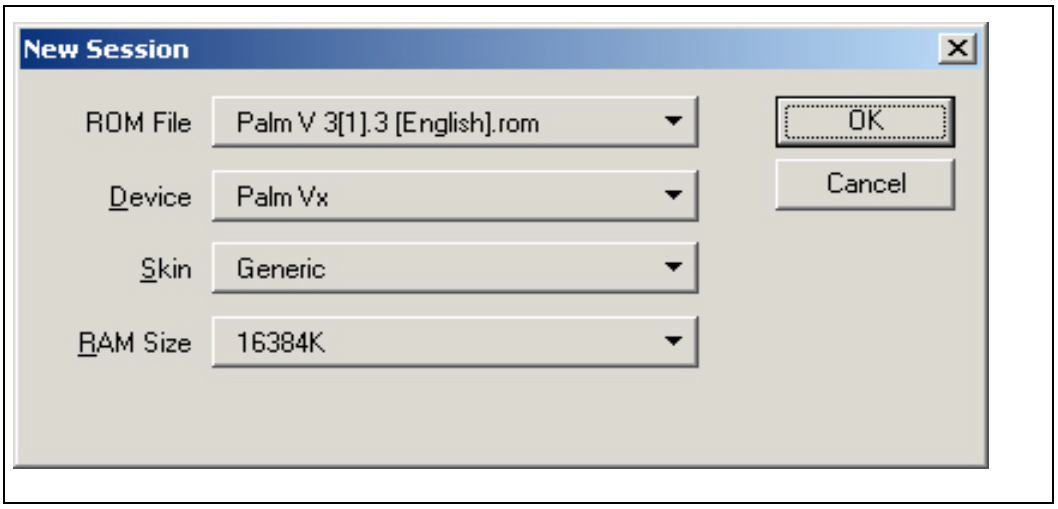


Fig 2.1 Detall de les opcions de hardware que l'emulador deixa escollir del dispositiu a emular.

La ROM utilitzada pel projecte és el d'una màquina que té una pantalla de 160x160 píxels, una paleta de 256 colors i que funciona amb el sistema operatiu *Palm OS* 5.0. Aquest dispositiu disposa de només 6 botons i de la pantalla tàctil, a part dels botons físics. A la pantalla tàctil hi trobem 6 botons més accessibles només amb el llapis del dispositiu (Veure fig 2.2).

Fig 2.2 Emulador de Palm OS amb el sistema operatiu Palm OS 5.0

Aquesta versió de l'emulador disposa d'un programa anomenat "HostFS Emulator" que un cop carregat a l'emulador dóna la possibilitat d'utilitzar directoris del disc dur del PC com a targetes de memòria de la màquina original, de manera que podem carregar els jocs de *LucasArts* utilitzats per l'*ScummVM* en memòria.

Palm OS va deixar de donar suport a l'emulador amb la desaparició de l'arquitectura *ARM* de 68K. El que els desenvolupadors utilitzen en l'actualitat per provar les seves aplicacions és el simulador.

El simulador el que fa és reproduir el comportament del sistema operatiu de la màquina simulada. El simulador de *Palm OS*, és una extensió de l'emulador de manera que l'inclou per poder executar les aplicacions de 68k. La principal diferència entre un i altre, és que el fet d'haver d'emular tot el hardware original de la màquina, en el cas de l'emulador, fa que depenent de la màquina on l'executem vagi molt més lent que la màquina original. Mentre que el simulador és més fidel en aquest aspecte.

2.4 *Palm OS* 5.0 SDK

Un SDK (*Software Development Kit*) és un conjunt de les eines/aplicacions necessàries per desenvolupar una aplicació en un llenguatge determinat o bé en un entorn determinat.

L'SDK de *Palm OS* ens ofereix una sèrie de llibreries, documentació i exemples que ens permeten desenvolupar aplicacions de manera més o menys senzilla.

L'SDK utilitzat en el nostre projecte ha estat el *Palm OS 5 SDK 68K R3*. Aquest paquet de desenvolupament ens vé donat en 2 formats. Un d'instal·lable, que consta d'un binari que detecta automàticament la versió de *Codewarrior* que tenim instal·lat, i ens integra l'SDK amb el compilador. I un altre comprimit que ens permet seleccionar els elements de l'SDK i instal·lar-los manualment sobre qualsevol compilador.

El contingut de l'SDK ens vé donat seguint la següent estructura:

- Codewarrior support
- *Palm documentation*
- *Palm OS support*
- *Palm tools*
- Licenses

2.4.1 *Codewarrior support*

El *Codewarrior support agrupa* una sèrie d'eines que donen a qualsevol versió del *Codewarrior,* la possibilitat de crear i compilar aplicacions per *Palm OS.* Concretament, hi trobem el suport per desenvolupar aplicacions en llenguatge C/C++ i per desenvolupar aplicacions amb múltiples segments (veure l'apèndix 2).

També hi trobem un conjunt d'aplicacions genèriques d'exemple de les funcionalitats que ens aporta l'SDK.

Els *plugins* necessaris perquè l'SDK s'integri amb la interfície gràfica els podem trobar en aquest directori. Aquest *plugins* agrupen els *linkadors* i les llibreries, que serveixen per afegir les opcions de l'SDK al panell de control del compilador.

2.4.2 *Palm documentation*

En aquest directori hi trobem tots els documents necessaris per implementar aplicacions per *Palm OS*, mitjançant l'SDK.

Els documents més rellevants són:

- ***Palm OS Companion***: Proporciona una guia de la *Palm* i de com programar aplicacions per *Palm OS*. Explica detalladament les parts del sistema operatiu i del maquinari. També explica els mètodes proporcionats per l'SDK que controlen i gestionen el sistema operatiu.
- ***Palm OS Reference***: Conté la descripció de totes les crides als mètodes i les funcions del *Palm OS*, juntament amb la descripció i crides a les estructures de dades més importants.
- ***Development Tools guide***: Descriu una sèrie d'eines i el seu funcionament, que es poden utilitzar per desenvolupar aplicacions per *Palm OS*. La més interessant de totes és el *Debugger*.

2.4.3 *Palm OS Support*

El contingut d'aquest directori ens proporciona les llibreries que el *Codewarrior* necessita per la generació automàtica de codi quan creem una aplicació nova. Juntament amb les llibreries que ha d'incloure el projecte per permetre la comunicació entre l'emulador i el compilador.

També ens proporciona les llibreries que necessitem per desenvolupar aplicacions per *Palm OS*, seran totes aquelles que haurem d'incloure en el nostre codi, en funció de les necessitats del nostre programa. Per acabar hi trobem també, les eines per poder fer crides al sistema.

2.4.4 *Palm tools* i *licences*

El directori *Palm tools*, ens dóna una sèrie d'eines en forma d'aplicació, que podem utilitzar per desenvolupar les nostres aplicacions.

Aquestes eines o aplicacions serien el *debugger* per *Palm OS* que ens permet analitzar el codi que s'està executant, donant-nos la possibilitat d'aturar l'execució del programa en qualsevol punt, i analitzar les variables i el funcionament d'aquest.

Una altra eina necessària a l'hora de crear aplicacions i que trobem en aquest directori és el *Constructor*.

El *Constructor* és una aplicació que va lligada molt lligada amb el compilador. És un editor d'entorn visual, per poder construir interfícies d'usuari. Disposa d'un gestor de projectes, un catàleg d'objectes, un dissenyador de formularis i d'un petit editor gràfic que ens permet crear icones i imatges. El constructor genera un fitxer de recursos que posteriorment es pot integrar en el projecte creat amb el compilador.

I per acabar podem trobar el suport per *Codewarrior* de caràcters japonesos.

En el directori de *Licences*, hi trobem la llicència i les normes d'ús del paquet SDK.

2.5 *Metrowerk's Codewarrior 8.0* per *Palm OS*

Aquest és el compilador escollit per desenvolupar la nostra aplicació. El *Codewarrior* ofereix una interfície sensiblement diferent a les interfícies dels compiladors gràfics que existeixen per Windows. Això és degut a què va ser desenvolupat per usuaris de Machintosh. Aquest fet fa que requereixi un cert període d'adaptació al nou entorn (veure figura 2.3).

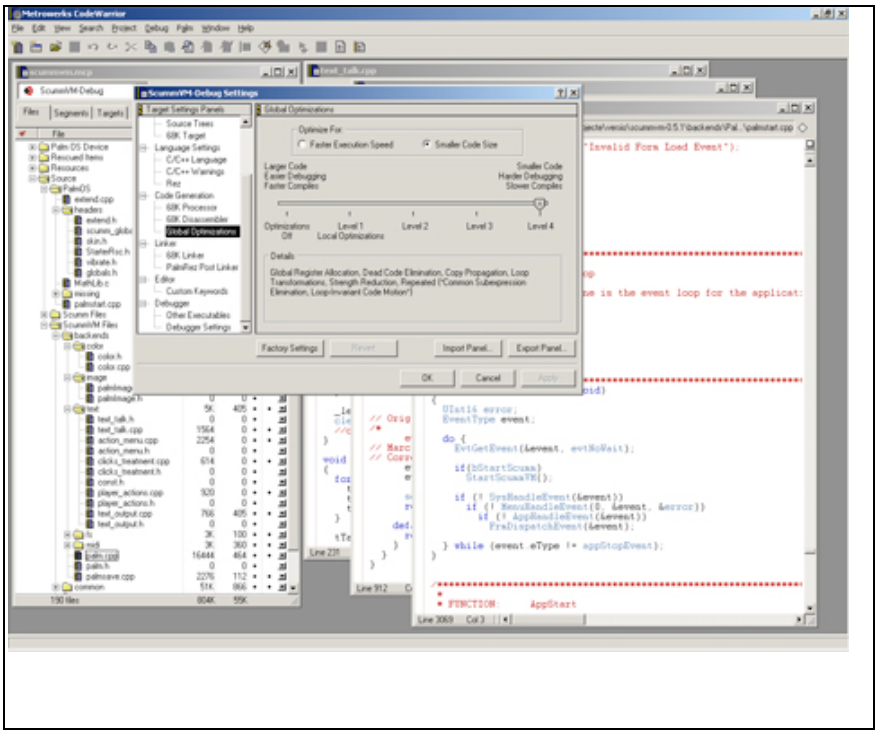


Fig 2.3 Captura de pantalla del Codewarrior 8.0 for Palm OS

Els avantatges que presenta aquest entorn de desenvolupament sobre *Palm OS* són els següents:

- Possibilitat de treballar amb l'emulador de Palm agilitant el procés de debug de les aplicacions. El fet de comunicar-se directament amb l'emulador fa que, mentre treballem en la modalitat de debug, podem tenir controlada l'execució del programa així com les dades d'aquest.
- Connexió entre el compilador i l'emulador de *PalmOS*. Aquest fet simplifica el procés de càrrega de l'aplicació sobre l'emulador per provar i *debugar* els avenços de la nostra aplicació.
- Controlar aspectes de la programació per Palm com la segmentació del codi (veure annex 2).
- Configuracions específiques per a la compilació en l'entorn de *Palm OS*, és a dir, és un compilador gràfic específic per compilar aplicacions destinades a funcionar sobre *Palm OS* les opcions dels menús i les configuracions del compilador estan encarades exclusivament a la generació de codi per aquest sistema.
- Escollir entre tota una sèrie d'optimitzacions de compilació. Discriminant si volem optimitzar, per aconseguir una major velocitat d'execució, o bé, si volem optimitzar per aconseguir que el binari ocupi poc espai. Ens dóna fins a quatre nivells d'optimització de codi.

Com hem dit abans, el *Codewarrior*, ens ofereix la possibilitat de connectar el compilador amb l'emulador de *PalmOS* (veure figura 2.4). Això es fa mitjançant el menú de configuració del compilador, podem assignar l'emulador com l'aplicació que s'utilitzarà per executar el nostre programa. És a dir, utilitzant aquesta opció, quan tant si escollim l'opció *debug*, com si escollim l'opció *run*, del nostre navegador, un cop compilat el projecte el carregarà i l'executarà sobre l'emulador.

Aquest fet, té un parell de conseqüències importants. La primera és que ens estalvia molt de temps a l'hora de provar l'aplicació, ja que no hem d'estar agafant el binari i carregant-lo manualment a l'emulador i la segona, i encara més important, és que permet executar l'aplicació en mode *debug*, permetent d'aquesta manera, un control de l'execució del programa a l'hora de buscar i arreglar els possibles errors de programació.

Cal dir que també ens ofereix la possibilitat de connectar el compilador a una PDA amb *PalmOS*, en el cas de disposar-ne d'una i treballar directament sobre la màquina i no sobre l'emulador.

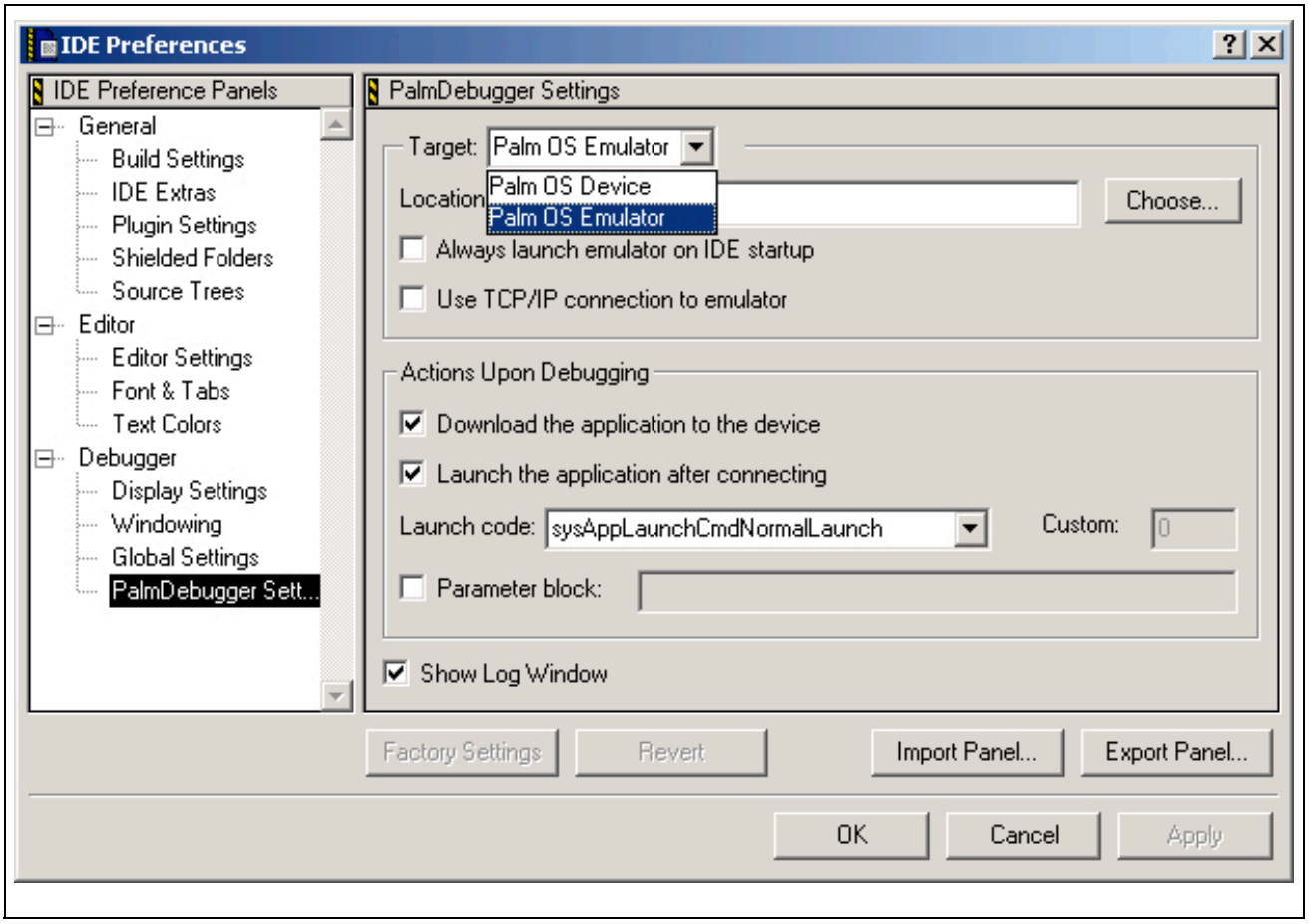


Fig 2.4 Pantalla de la configuració que permet assignar l'emulador o un dispositiu Palm al debugger del compilador.

2.6 *Microsoft Visual Studio 6.0*

El *Visual Studio*, és una *suite* comercial de programació per desenvolupar sobre *Microsoft Windows*. Està formada per varis llenguatges de programació, entre ells el *Visual C++*, que és amb el que treballarem en el nostre projecte. Aquest editor/compilador visual ens permet, editar, compilar i *debuggar*, aplicacions programades amb C i C++. Inclou moltes eines visuals que no utilitzarem, ens limitarem a utilitzar el compilador i el *debugger*.

Aquest és el complidor amb el que treballarem durant una bona part del projecte. L'*ScummVM*, porta el projecte de *Visual Studio,* en el codi font preparat per poder treballar amb la versió per PC. Però tot i estar preparat la compilació no va ser fàcil, és per això vaig documentar tots els passos necessaris per compilar la versió 0.5.1 de l'*ScummVM* a l'apèndix 1.

3 Palm / *Palm OS*

En aquest capítol explicarem els trets més característics que s'han de tenir en compte en el moment de voler desenvolupar una aplicació per una PDA que funcionen amb *PalmOS*. També explicarem l'estructura d'un programa, la gestió de la memòria que fa el *PalmOS.* Parlarem també dels tipus de fitxers anomenats *databases*.

3.1 Disseny d'aplicacions per Palm

A l'hora de desenvolupar una aplicació per un dispositiu Palm s'han de tenir en compte varis aspectes rellevants a causa de les característiques reduïdes dels dispositius. Aquests aspectes són els següents:

- S'ha de tenir en compte que l'aplicació s'executarà en una pantalla petita.
- S'ha de limitar l'entrada de text al dispositiu.
- S'ha de tenir en compte que el més probable és que el programa s'hagi de sincronitzar amb un altre programa que estarà en un ordinador de sobretaula.
- El programa ha d'ocupar poc espai en el disc, ha de ser petit.
- El programa ha de ser ràpid, hi ha d'estar molt optimitzat degut a les limitades prestacions del processador de la Palm.

A continuació aprofundirem més en aquestes limitacions que presenta *PalmOS*:

Pantalla petita

La mida de la pantalla de la Palm és de 160x160 píxels que es visualitzen en una àrea d'uns 6x6 centímetres. Per tant hem de tenir en compte que les dades que es volen mostrar s'han de poder visualitzar correctament dins d'aquesta àrea. A causa d'això, si les dades no hi caben, s'hauran de dividir en parts i agrupar-les en grups. Evidentment aquesta feina no és trivial i segurament el disseny s'haurà de refer moltes vegades fins a aconseguir el bo.

S'ha de tenir en compte l'existència dels anomenats *scrolls*, tant verticals com horitzontals, que seran essencials per distribuir la informació que volem mostrar a la pantalla, en el cas que en aquesta no hi càpiga. Però tot i poder fer servir els *scrolls* no hem d'oblidar que la missió és facilitar la vida a l'usuari i que el fet d'haver d'utilitzar molt les barres de desplaçament acaba cansant-lo.

No hem d'oblidar que en aquest cas, és la mida de la pantalla el que condiciona el disseny i no el contrari.

Limitar l'entrada de text al dispositiu

Hem de tenir en compte que l'absència de teclat fa que el fet d'introduir text a la Palm per part de l'usuari sigui una feina més pesada del que hauria de ser. S'ha de recordar que la Palm no és un dispositiu d'entrada de text. Per fer això, l'usuari disposa de la possibilitat de connectar-se amb un ordinador de sobretaula que li permet fer-ho amb molta més facilitat. Hem de recordar que les dades s'introdueixen mitjançant l'escriptori virtual que ens proporciona la tecnologia *HotSync*, que utilitzem des d'un ordinador de sobretaula, i que aquestes dades seran visualitzades a la Palm.

Evidentment que tot i així s'ha de permetre l'entrada de text per part de l'usuari directament a la Palm, però la quantitat de text ha de ser la mínima i necessària. Perquè l'usuari pugui entrar text la Palm disposa de:

- El teclat projectat a la pantalla.
- El *Graffiti.*
- Altres eines com el copia i enganxa.

Tots aquests elements hauran de ser accessibles per l'aplicació.

Fig 3.1 Detall dels diferents tipus de teclats que ens ofereix el PalmOS. Lletres, numèric, internacional.

A la figura 3.1 podem observar els diferents tipus de teclats que ens proporciona el *Palm OS*, via *software*. Tenim els dos teclats amb les lletres de l'alfabet amb accents i sense accents, i el teclat numèric i de símbols.

L' altra manera d'introduir dades a la *Palm* més utilitzada, és mitjançant el que s'anomena *Graffiti*.

El *Graffiti* és el sistema de reconeixement d'escriptura utilitzat per *Palm*. Aquest sistema es basa en una nova manera, molt simple d'escriure els caràcters en majúscules. S'ha ideat un sistema, d'escriptura que permet escriure tots els caràcters necessaris en majúscules, sense haver d'aixecar el llapis de la superfície sensible al tacte del llapis de la *Palm*. S'eliminen les parts complexes de les quatre lletres més difícils (A, F, K , T). Podem veure un exemple d'aquest mètode d'escriptura a la figura 3.2. En el diagrama de la figura, s'observen tres tipus diferents de caràcters que es poden escriure mitjançant les seves *gestures* (en aquest cas descripció del moviment que ha de fer el llapis), amb la seva equivalència a la taula dels caràcters alfanumèrics. Concretament tenim, lletres majúscules, símbols de puntuació i caràcters amb accents.

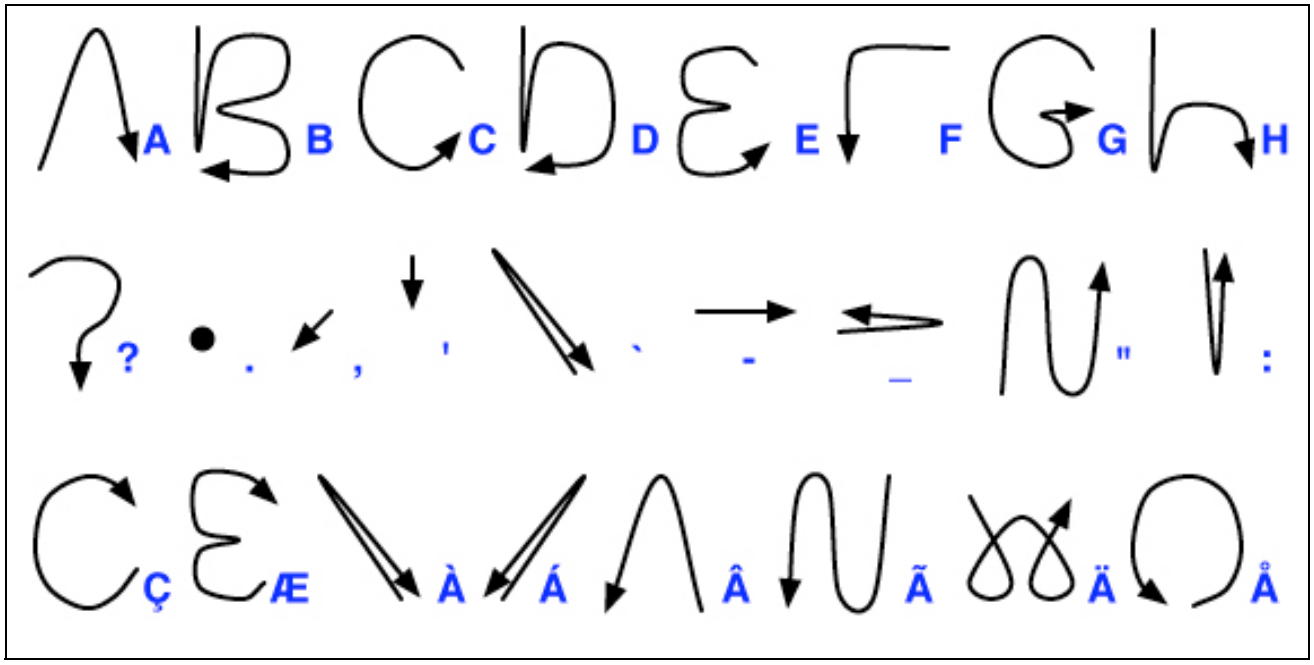


Fig 3.2 Exemple d'escriptura utilitzant el mètode del Graffiti.

També s'anomena *Graffiti* a la superfície rectangular sensible al llapis que trobem a la part inferior central de la *Palm* (veure figura 3.3).

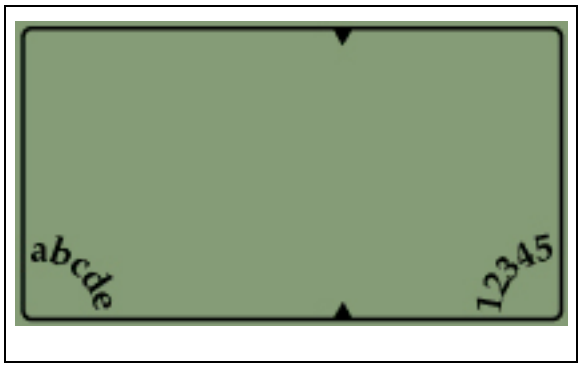


Fig 3.3 Detall de l'àrea del Graffiti a la Palm.

Sincronització de dades

Un altre factor important és el fet que la nostra aplicació s'haurà de sincronitzar amb el PC, ja que els dissenyadors de Palm van pensar aquests dispositius, perquè coexistissin de manera simbiòtica amb altres. El resultat d'això és la tecnologia *HotSync* que permet sincronitzar les dades des de l'ordinador cap a la Palm i viceversa.

Aplicació petita

L'aplicació ha d'ocupar tant poc espai a memòria com sigui possible, perquè no es disposa d'una mida de pila gaire gran i tampoc hi ha molt d'espai per emmagatzemar dades.

Aplicació ràpida

Els usuaris de PDAs mesuren el temps de manera diferent als usuaris d'ordinadors personals, per començar un s'està movent i l'altre està assegut. És molt probable que un usuari de PDA estigui fent més d'una cosa alhora de manera que no disposa de molt de temps. Això fa que no es pugui estar esperant molta estona a què l'aplicació s'engegui. Amb tot això volem dir que la velocitat de l'aplicació és molt important, ja que l'usuari d'un PDA espera un temps de resposta molt ràpid a cada acció que fa. Quan hem de dissenyar l'aplicació hem de tenir molt en compte que la mida de la pila és realment molt petit, per això hem de vigilar amb les rutines recursives i amb voler guardar grans quantitats d'informació a la pila. La memòria dinàmica és tant petita que hem de limitar molt

la quantitat de variables globals. També s'ha de limitar el fet de guardar grans quantitats de dades a la pila dinàmica.

Tots aquests aspectes són molt rellevants quan volem desenvolupar una aplicació nova per la Palm, però encara ho són molt més si es vol fer el que intentem fer en aquest treball, és a dir, quan es tracta de la portabilitat d'aplicacions que ja existeixen per PC. Hem de tenir ben clars cadascun d'aquests punts, perquè seran els que ens marcaran els objectius i els problemes a superar.

3.2 Tractament d'events i programa principal

El funcionament d'una aplicació en un sistema d'events com és el *Palm OS*, és bastant simple: l'aplicació s'executa en forma de bucle, escoltant els events provocats en el sistema (tant per l'usuari com pel mateix sistema operatiu) i actua en conseqüència.

Per familiaritzar-nos amb el *Palm OS* i entendre millor el funcionament del sistema d'events i l'estructura del programa principal, explicarem el codi i l'estructura que ens genera per defecte el *Codewarrior* quan creem un projecte nou.

Abans de poder treballar, hem d'instal.lar el *Codewarrior* 8.0 i el *Palm SDK*, tasca que no és gens complicada a causa que, com ja hem esmentat en apartats anteriors, el *Codewarrior* ja vé preparat per poder desenvolupar sobre Palm, és a dir, ja porta incorporat l'SDK. Però cal mencionar que la versió que porta és la 3.0, que en el nostre cas l'actualitzarem a la 5.0. Un cop tenim el compilador apunt ja podem crear el projecte.

Per crear-lo utilitzarem mitjançant el menú "*File > New Project*" el compilador ens pregunta llavors quin tipus de projecte volem i quin llenguatge. Nosaltres li diem que volem desenvolupar una aplicació per Palm i que la volem desenvolupar en C++ (veure figura 3.4).

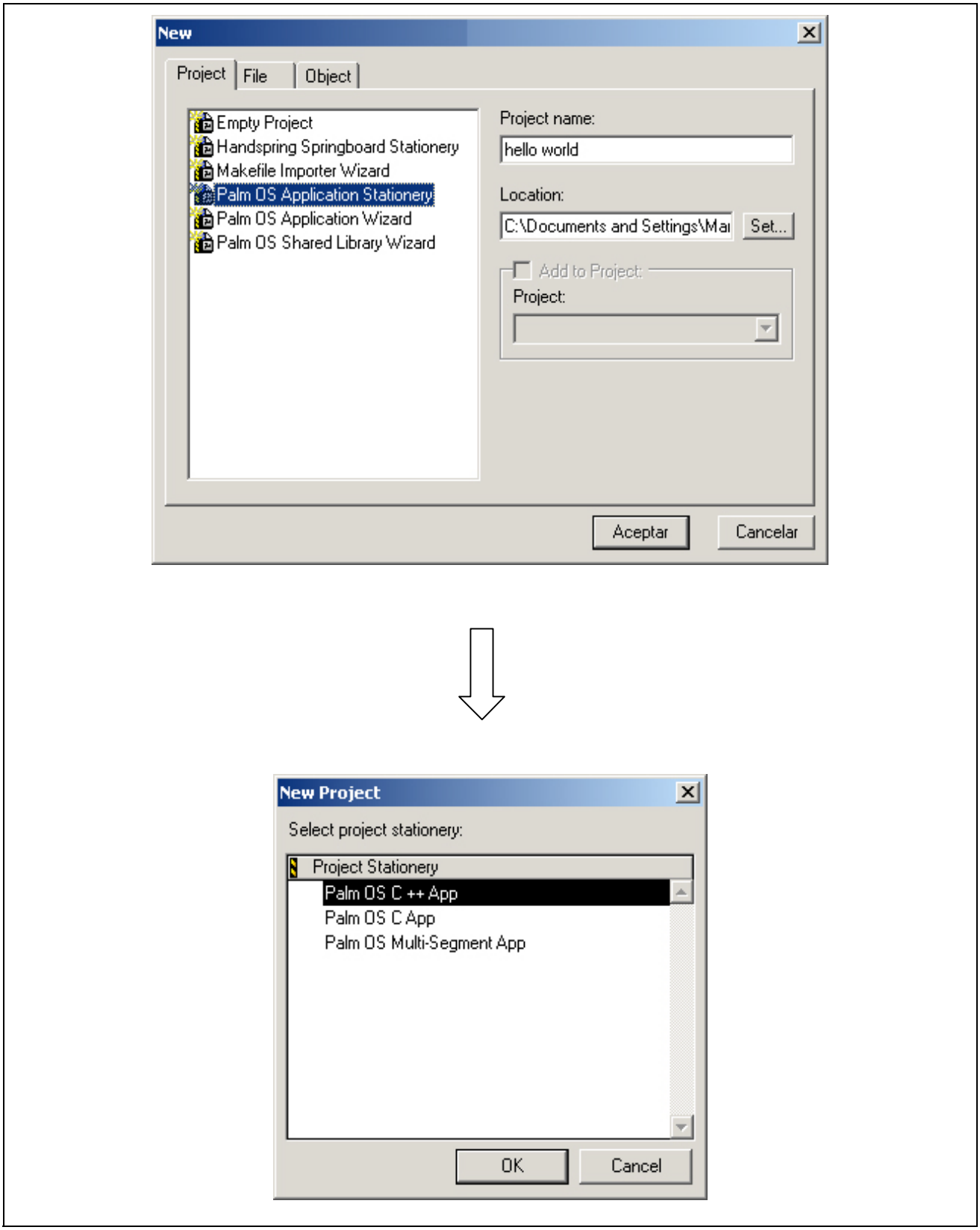


Fig 3.4 Procés de selecció de tipus de projecte i llenguatge de programació.

Un cop fet això, el compilador ens crea tota l'estructura de directoris del projecte, incloent les llibreries per defecte. Ens trobem que el compilador també ens crea el fitxer "*Starter.cpp*", amb el

programa principal o *main,* i una sèrie de mètodes que ens permeten controlar tant els events com l'inici i el fi de l'aplicació. Doncs, ens trobem que el compilador ens estalvia una part de la nostra feina, facilitant-nos l'aprenentatge de com desenvolupar una aplicació per *Palm OS*.

Aquest projecte és pot compilar directament: ho fem mitjançant el menú "*Project* > *Make*", així veiem què és el que fa exactament. Ens trobem que senzillament ens genera un binari de *Palm OS* que podem carregar a l'emulador i que ens mostra un formulari amb un menú d'opcions on hi ha els apartats *About, Power off* i *Find* (veure figura 3.5). El primer ens descriu una mica la utilitat de l'*starter.cpp*, mentre que el segon apaga el *PDA* i l'últim executa l'aplicació nativa del sistema operatiu de cerca.

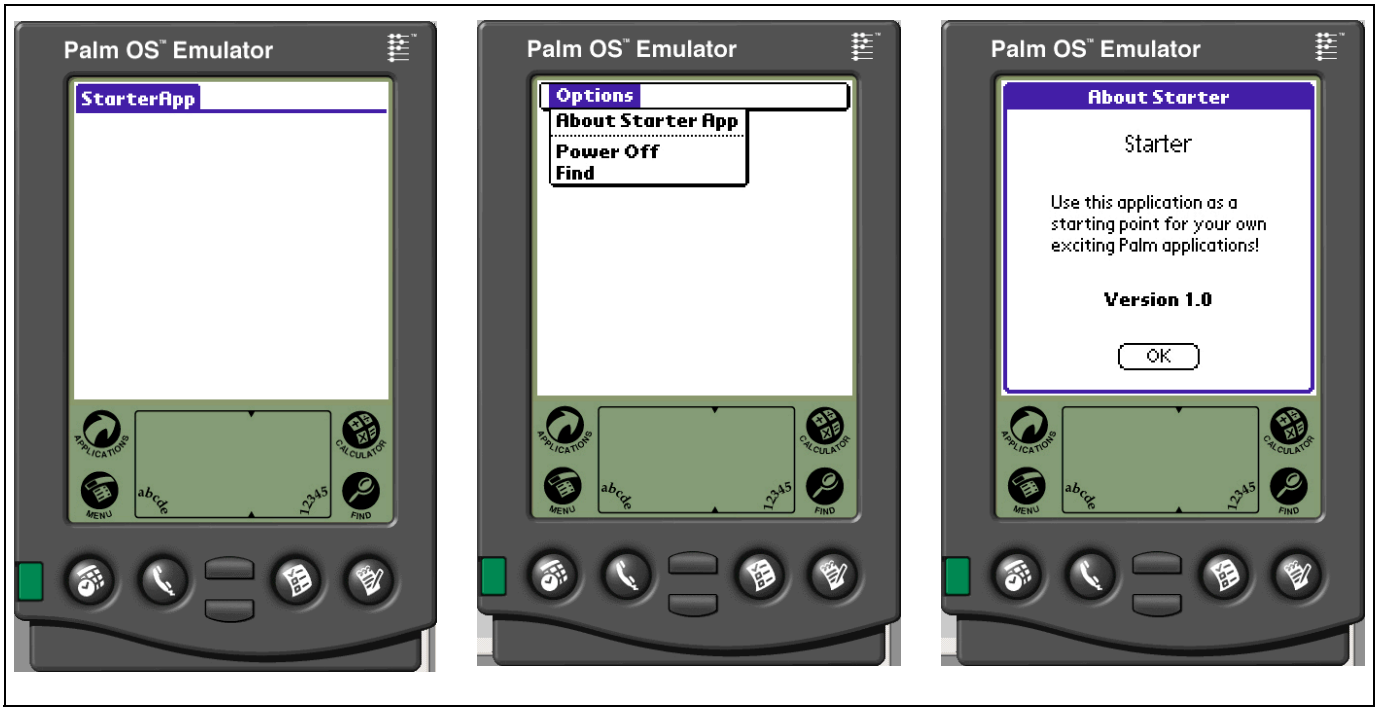


Fig 3.5 Aplicació starter.cpp. Execució. Menú d'opcions. About.

Si entrem a mirar el codi de l'*starter.cpp* veiem exactament com es tracten els events i el funcionament principal d'un programa de Palm.

```
static UInt32 StarterPalmMain(UInt16 cmd, MemPtr /*cmdPBP*/, UInt16 launchFlags)
{
 Err error;
 error = RomVersionCompatible (kOurMinVersion, launchFlags);

 if (error) return (error);
  switch (cmd) {
   case sysAppLaunchCmdNormalLaunch:
      error = AppStart();
      if (error) return error;
      FrmGotoForm(MainForm);
      AppEventLoop();
      AppStop();
      break;
   default:
      break;
  }
 return errNone;
}
```

Fig 3.6 Detall del codi de la funció StarterPalmMain

Per començar, ens trobem amb la funció anomenada *PilotMain*, que és l'equivalent a la funció *main* de qualsevol programa. Aquesta, ens envia a l'*StarterPalmMain* (veure figura 3.6) que serà l'encarregada de fer les comprovacions de versió de la *Rom* del PDA i així poder detectar incompatibilitats. En el cas que tot hagi anat correctament, es carregaran les preferències del programa i s'inicialitzaran les variables. Un cop fet això mostrarà el formulari principal de l'aplicació i engegarà el bucle d'events que s'estarà executant fins que es decideixi aturar el programa, ja sigui, perquè l'usuari l'atura voluntàriament o perquè el sistema operatiu decideix tancar-lo. Un cop es decideixi tancar el programa executarà la funció que guarda totes les dades necessàries i finalitza el programa correctament.

Tot i que la funció *main* del programa seria com ja hem dit (el *PilotMain*), podem veure que el que realment ens interessa és la funció *StarterPalmMain*, ja que serà la que realment engegarà la nostra aplicació. Un cop hem vist això a grans trets, veurem com tractem els events. Per fer-ho analitzarem una mica més d'aprop la funció *AppEventLoop*, que podem veure a la figura 3.7.

Dins d'aquesta funció ens trobem amb el bucle principal d'events. Aquest, s'executarà indefinidament fins que rebi l'event de tancar l'aplicació. Els events que rep el programa

procedeixen d'una cua del tipus *FIFO* (*First in First out*), de manera que els events es van tractant a mesura que es van provocant i en el mateix ordre en el que es generen. Aquests events poden ser provocats tant pel sistema operatiu com per l'usuari de l'aplicació.

El que fa el bucle principal, és treballar desencuant events i decidir com tractar-los. Per fer-ho, distingeix entre els següents tipus d'events:

- **Event de sistema**: si l'event rebut és un event provocat pel sistema, dona el control al sistema, perquè el tracti.
- **Event de menú**: en el cas que sigui un event de menú, serà el sistema el que actuarà mostrant o amagant el menú.
- **Event de l'aplicació**: els events de l'aplicació són els que el programador en qüestió decideix tractar. En aquest cas es donarà pas a la funció del programador, perquè tracti l'event.

```
static void AppEventLoop(void) {
 UInt16 error;
 EventType event;
 do {
    EvtGetEvent(&event, evtWaitForever);
    if (! SysHandleEvent(&event))
       if (! MenuHandleEvent(0, &event, &error))
         if (! AppHandleEvent(&event))
            FrmDispatchEvent(&event);
 } while (event.eType != appStopEvent);
}
```

Fig 3.7 Detall del codi de la funció AppEventLoop

Ens centrarem en els events de l'aplicació, que són els que realment interessen al programador, ja que seran aquests els que el programador podrà decidir com tractar i controlar. Un cop s'han descartat els events de sistema i els events de menú, el que ha de fer l'aplicació és donar pas a la funció que tractarà aquest event. S'ha de dir que cada formulari ha de tenir la seva funció de tractament d'events, ja que es canvia de formulari també es canvia de *GUI* (*Graphic user interface*), i que això pot fer que apareguin nous elements que generen events diferents als que es podien estar tractant en el formulari anterior.

Per definir la funció de tractament d'events que s'ha de fer servir en cada moment hem d'utilitzar la funció *FrmSetEventHandler* que ens subministra l'SDK del *Palm OS* i que, donat un identificador de formulari, assignarà una funció per tractar els events al controlador d'events de l'aplicació. Un cop definida aquesta funció, el *FrmDispatchEvent* serà l'encarregat d'executar-la.

La funció *FrmSetEventHandler* la trobem dins de l'*AppHandleEvent* i la podem examinar més detalladament a la figura 3.8. Aquesta funció serà la que mirarà si l'event que rep és l'event d'obrir un formulari i carregarà la funció destinada a tractar els events d'aquest, al controlador d'events.

```
static Boolean AppHandleEvent(EventPtr eventP) {
 UInt16 formId;
 FormPtr frmP;

 if (eventP->eType == frmLoadEvent) {
    // carreguem el formulari.
    formId = eventP->data.frmLoad.formID;
    frmP = FrmInitForm(formId);
    FrmSetActiveForm(frmP);

    // Defineix el controlador d'events pel formulari. Aquest controlador
    // serà cridat quan el formulari estigui actiu i rebi un event.
    switch (formId) {
      case MainForm:
            FrmSetEventHandler(frmP, MainFormHandleEvent);
            break;
      default:
//          ErrFatalDisplay("Invalid Form Load Event");
      break;
     }
     return true;
 }
 return false;
}
```

Fig 3.8 Detall del codi de la funció AppHandleEvent

En definitiva, és des de la funció *AppHandleEvent* des d'on podrem definir les funcions a utilitzar per tractar els events de cada formulari de la nostra aplicació.

En el cas de l'*starter.cpp* ens trobem amb la funció *MainFormHandleEvent*, aquesta funció fa que s'obri la finestra d'informació de l'aplicació quan l'usuari fa clic sobre l'opció *AboutStarterApp* que hem vist a la figura 3.4.

3.3 Gestió de la memòria

Com ja hem dit anteriorment la quantitat de memòria disponible en un PDA del tipus Palm és escassa, per tant, s'ha d'anar amb molt de compte a l'hora de programar una aplicació per aquests dispositius.

3.3.1 Estructura general

Per entendre millor tot això començarem fent una ullada a l'arquitectura de la memòria. La memòria RAM està dividida en 2 grans blocs (veure figura 3.6):

- **Emmagatzematge**. Aquesta part està gestionada pel gestor de les anomenades bases de dades, ja que és la part de la memòria on es guarda la informació estàtica.
- **Dinàmica**. En aquesta part de la memòria és on trobem les dades de les aplicacions que s'estan executant, així com també del sistema operatiu. El gestor de memòria és l'encarregat de gestionar la informació d'aquesta part de la memòria.

Ens centrarem en la memòria dinàmica, que és en la que intervé el gestor de memòria.

En *Palm OS* la memòria dinàmica s'utilitza per guardar-hi de manera temporal:

- Les dades globals de sistema (buffer de pantalla, interfície d'usuari, referències a bases de dades, …).
- Memòria dinàmica de sistema (IrDA, finestra del *find*, dades temporals, …).
- Memòria dinàmica d'aplicacions i globals de les aplicacions (estructures dinàmiques, variables globals i estàtiques).
- La pila de les aplicacions (la pila i les variables locals).
- La pila del protocol TCP/IP.

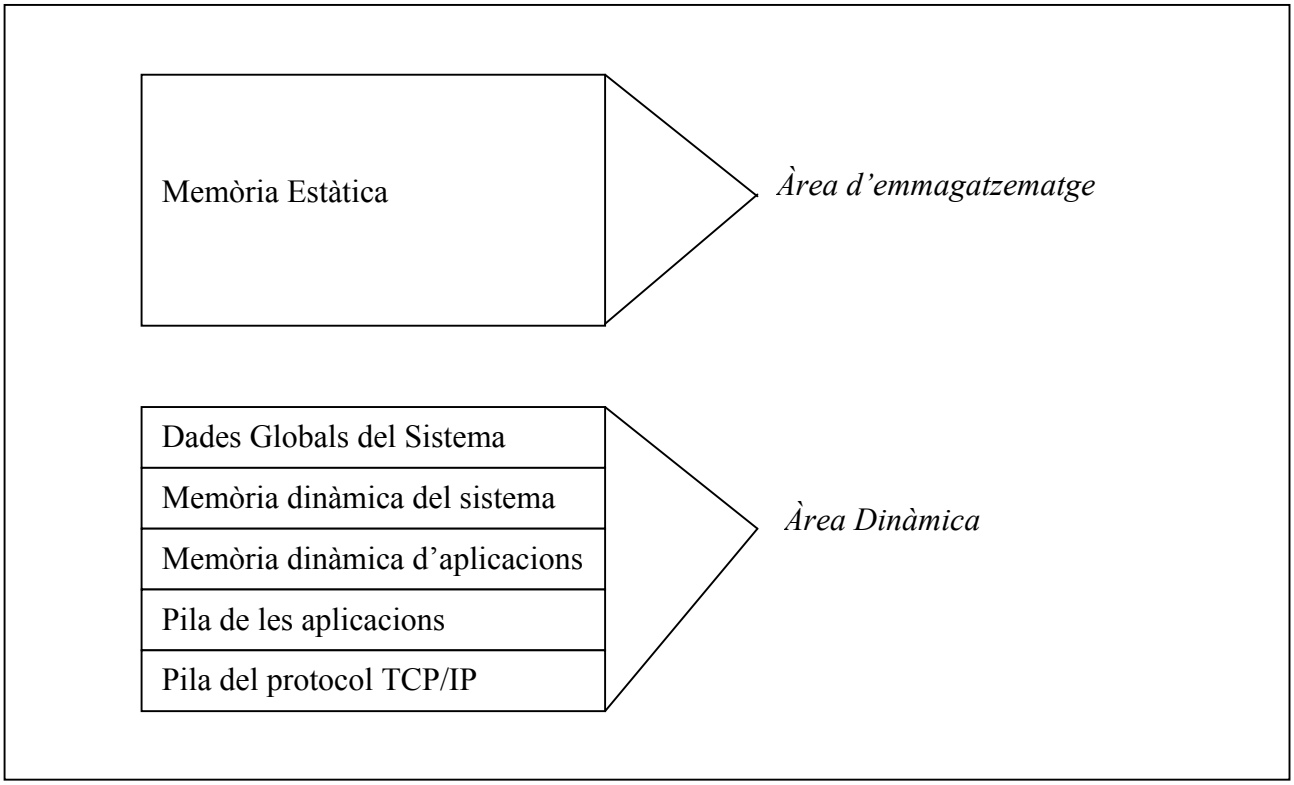


Fig 3.6 Diagrama de la memòria.

3.3.2 *Dynamic Heap*

Un *heap* és una àrea de la memòria contigua utilitzada per gestionar un o més trossos petits de memòria, per tant el *dynamic heap*, és una part de la memòria on hi ha dades dinàmiques. Quan les aplicacions treballen amb la memòria normalment utilitzen trossos petits de memòria, també anomenats *chunks*. El conjunt de *dynamic heaps* forma l'àrea de memòria dinàmica del sistema.

Els desenvolupadors d'aplicacions poden treballar amb *chunks* que no es poden recol·locar anomenats punters o bé treballar amb chunks que si que es poden recol·locar. És preferible treballar sempre amb aquests últims, ja que així el gestor de memòria, que és l'encarregat de tractar aquests trossets petits d'informació i recol·locar-los, podrà fer la seva feina i recol·locar els *chunks* de manera que tot l'espai de memòria lliure estigui contigu per intentar reduir la fragmentació (veure la figura 3.7).

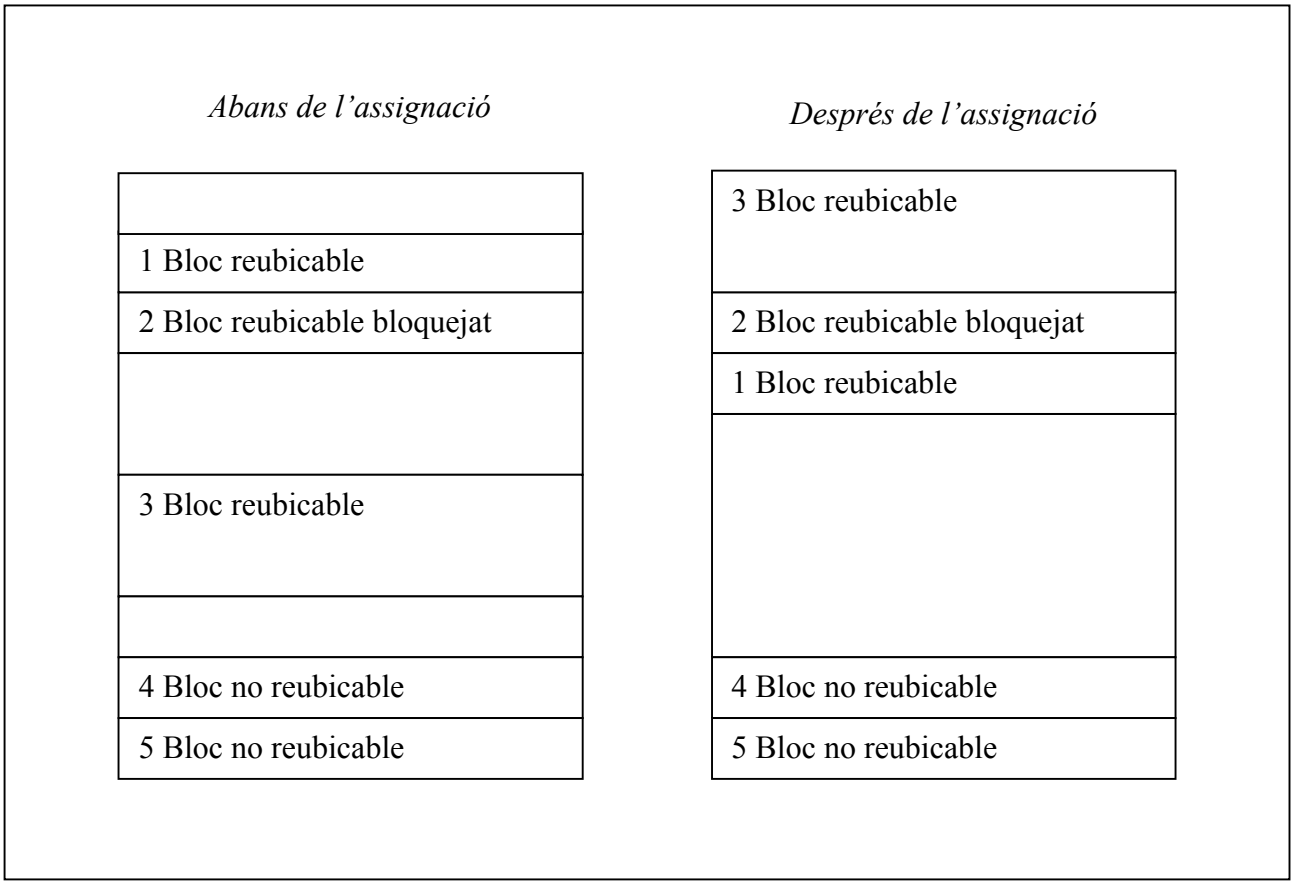


Fig 3.7 Diagrama de la gestió de memòria

3.3.3 *Static Heap*

L'*static heap* és una part petita de la memòria on es guarden les dades estàtiques, com podrien ser els fitxers de dades en un PC. El conjunt d'aquests *heaps* és el que anomenem l'àrea estàtica de la memòria. Aquesta part de la memòria és gestionada íntegrament pel gestor de dades (*Data Manager*).

En aquesta part de la memòria hi podem trobar un o més d'aquests dos components:

- Bases de dades (*database*).
- Registres (*record*).

La relació entre aquests dos components és molt forta ja que una base de dades agrupa un conjunt de registres. Aquests registres són blocs de memòria reubicables. S'ha de tenir molt en compte que un registre, per limitacions del *PalmOS*, no pot excedir mai la mida dels 64Kb.

Per poder mantenir la integritat de l'àrea de dades, aquest està protegit contra escriptura. Això assegura que una aplicació no pugui destruir accidentalment dades d'una altra aplicació o, fins i tot, destruir altres aplicacions. Tots els canvis a les bases de dades i als registres es fan mitjançant el *Data Manager API*, que és l'encarregat d'assegurar-se de què no s'escrigui en zones de la memòria que no pertoca.

3.4 Bases de dades

Com acabem de dir una base de dades és una col·lecció de registres, per saber quins registres pertanyen a una base de dades. Aquesta, guarda la següent informació:

- La posició del registre a memòria.
- Un identificador únic de 3 bytes. Aquest ID és assignat directament pel gestor de dades quan es crea un registre.
- Un atribut d'un byte que conté els flags d'esborrat, arxivat, ocupat i privat d'un bit cadascun i 4 bits de categoria.

Una base de dades ha d'estar necessàriament emmagatzemada en un sol *heap*, mentre els registres que la formen poden estar distribuïts a varis *heaps* (veure figura 3.8).

A més a més de la informació sobre els registres que formen una base de dades aquestes poden contenir els tipus d'informació següent:

- **Bloc d'informació d'aplicació:** sol contenir els noms de les categories o bé altres dades respectives a la base de dades.
- **Bloc d'informació d'ordenació:** Aquí és on es guarda una llista amb l'ordenació que es vol donar als registres de la base de dades.
- **Nom, tipus i creador:** Les bases de dades s'han de crear amb un nom, que ha de ser únic, un tipus i un creador. Quan un usuari esborra una aplicació, el sistema operatiu esborra automàticament totes les bases de dades que comparteixen el mateix creador. Perquè aquesta neteja es pugui portar a terme correctament és important que les bases de dades tinguin com a creador el mateix que l'aplicació.

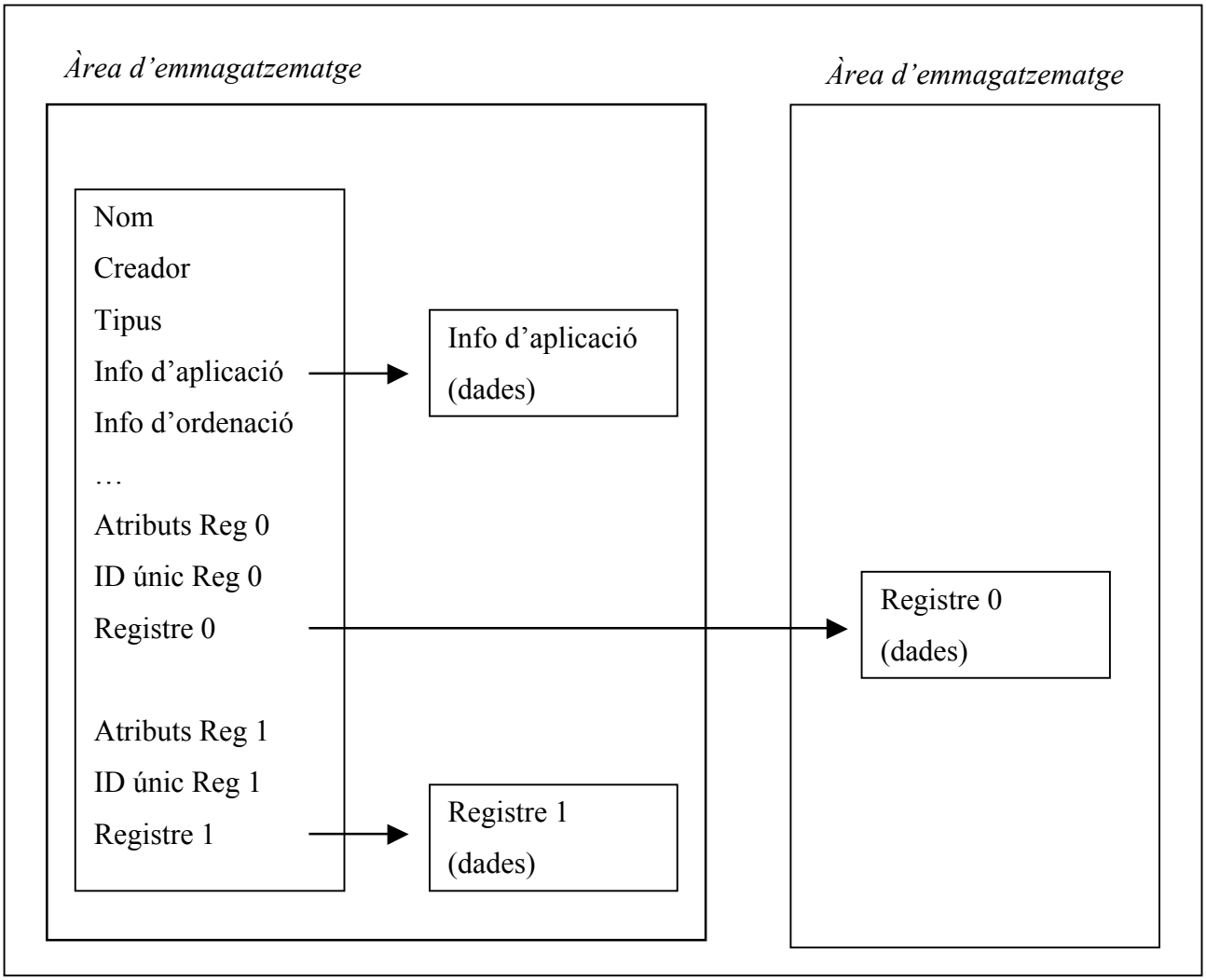


Fig 3.8 Diagrama de les bases de dades

Com ja hem dit les bases de dades es guarden en l'àrea de la memòria dedicada a l'emmagatzematge. Aquesta àrea estaria repartida entre l'àrea de la memòria del dispositiu i les targetes d'expansió de memòria. En els dispositius actuals que funcionen amb *Palm OS* tenen només una targeta de memòria, tot i que el *Palm OS* és capaç de suportar-ne múltiples. Per accedir a la targeta s'utilitza un número o identificador, la part de la memòria interna portaria el número 0 i la primera targeta portaria l'1 i així successivament.

Quan es crea una base de dades nova s'ha d'especificar a quina targeta s'emmagatzemarà. Per accedir-hi des d'una aplicació haurem d'utilitzar els *handles* (manegadors de memòria). L'accés es fa igual que si es tractés de la memòria interna, però sempre utilitzant l'identificador de targeta, que ens marcarà l'adreça base i així el sistema operatiu s'adoni que volem accedir a la part de la memòria d'una targeta.

4 *ScummVM*

4.1 Introducció

En aquest capítol explicarem què és, què fa i com funciona l'*ScummVM*. Per fer-ho explicarem la seva estructura i les parts més rellevants de l'aplicació per així poder entendre millor la portabilitat de l'aplicació.

4.2 Què és l'*SCUMM*?

Script Creation Utility for Maniac Mansion .- Utilitat de creació de guions pel *Maniac Mansion*.

El sistema *SCUMM* neix de la idea que la millor manera de realitzar una tasca complexa és començar per crear una eina que t'ajudi a resoldre-la. D'aquesta manera, uns programadors de *LucasArts* van decidir que en comptes de fer un sol programa molt complicat per cada nova aventura gràfica, era millor fer un motor genèric que fos capaç d'executar qualsevol aventura gràfica, sempre i quan se li subministressin les dades adients. Això els permetia concentrar-se en el disseny dels jocs, en comptes de preocupar-se pels detalls de la programació. Cal mencionar, que l'SCUMM no és de programari lliure, és una utilitat registrada per *LucasArts*.

La manera de funcionar és la següent: només existeix un sol fitxer executable, anomenat l'intèrpret que treballa amb les dades dels fitxers de dades que estan implementats utilitzant el sistema *SCUMM*. Aquests fitxers contenen les imatges, els diàlegs, detalls del comportament dels objectes, els personatges, les pel·lícules d'introducció,… L'intèrpret recull totes aquestes dades dibuixant-t'ho tot, animant els caràcters, processant la interacció amb l'usuari i tota la resta de detalls necessaris perquè una aventura gràfica funcioni. Gràcies al fet que no hi ha codi executable en els fitxers de dades, el fet de portar un joc a una altra plataforma és molt simple. Només cal portar l'intèrpret i fer servir els mateixos fitxers de dades.

4.3 Què és l'*ScummVM*?

Script Creation Utility for Maniac Mansion Virtual Machine .- Màquina virtual de la utilitat de creació de guions pel *Maniac Mansion*.

L'*ScummVM* és una aplicació que pertany al programari lliure, que agrupa intèrprets de molts dels jocs del popular gènere de les aventures gràfiques. Inicialment va ser creat per suportar els jocs creats amb el sistema *SCUMM* de *LucasArts*.

El sistema *SCUMM*. Ha sofert canvis i modificacions d'un joc a un altre. Això passa, perquè al llarg dels anys, els jocs han necessitat noves funcionalitats i millores que s'han anat incorporant al format dels fitxers. Com hem dit abans, l'SCUMM no va ser dissenyat per fer-se públic. Això vol dir que l'*ScummVM* no ha estat creat a partit del codi font original de l'*SCUMM*. Això repercuteix en el fet que l'*ScummVM* estigui actualitzant-se constantment per poder contemplar tots aquests canvis i variacions que hi ha d'un joc a l'altre, i per arreglar els errors que apareixen amb aquests canvis.

Cal dir que els creadors i desenvolupadors de l'*ScummVM* han arribat a un acord amb *LucasArts* que els permet seguir desenvolupant l'aplicació després d'haver estat sota sospita de plagi.

La funcionalitat de l'*ScummVM* és fer d'intèrpret dels fitxers de dades originals dels jocs, per així no haver de dependre de l'executable. S'encarrega d'interpretar aquests fitxers de dades, reemplaça així l'executable i dóna la possibilitat de jugar als jocs en plataformes diferents per les que havien estat dissenyats. Algunes de les quals detallem a continuació:

- Windows
- Windows CE
- Linux
- Mac OS X
- AmigaOS
- MorphOS
- BeOS
- Dreamcast
- GP32
- PalmOS
- UNIX

A la figura 4.1 podem veure el menú de la versió actual de l'*ScummVM* (0.8.2) per Mac OS X.

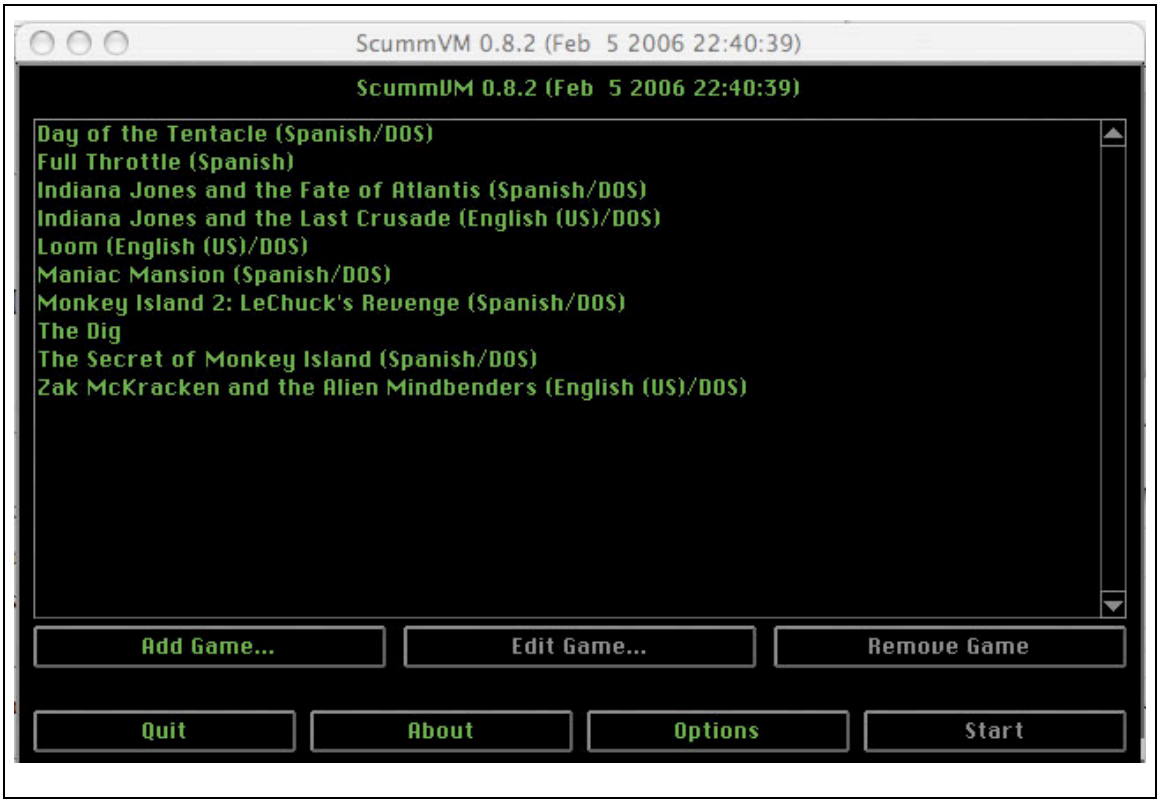


Fig 4.1 Captura de pantalla del menú d'opcions de la versió 0.8.2 de l'ScummVM per Mac OS X

Els jocs suportats per l'*ScummVM* són tots aquells fitxers de dades que segueixen el sistema *SCUMM*. Alguns dels jocs suportats són els següents (veure figura 4.2):

- *Maniac Mansion* (1988).
- Zak McKracken and the Alien Mindbenders (1988)
- *Indiana Jones and the Last Crusade* (1989) versions EGA i VGA.
- *Loom* (1990).
- *The Secret of Monkey Island* (1990) versions EGA i VGA.
- *Monkey Island 2: LeChuck's revenge* (1991)
- *Day of the Tentacle* (1993)
- *Sam and Max* (1993)
- …

Fig 4.2 Captures de pantalla del Maniac Mansion i del Zak McKracken and the Alien Mindbenders

Més endavant, els creadors de *l'ScummVM es* van adonar que moltes aventures gràfiques de l'època havien estat desenvolupades seguint la mateixa filosofia de *LucasArts*. Utilitzaven per tant, un sistema semblant (mitjançant un executable que interpretava els fitxers de dades dels jocs). Evidentment no utilitzaven el sistema *SCUMM*, que com ja hem dit, que era propietat de *LucasArts*, però si que utilitzaven la mateixa filosofia. Aquest fet va fer que els creadors de l'*ScummVM*, decidint implementar el motor capaç d'interpretar aquests jocs i ampliar així, les possibilitats de l'*ScummVM*. Alguns dels jocs suportats són els següents:

- *Beneath a Steel Sky* de *Revolution Software* (1994)
- *Simon the Sorcerer 1* de *Adventure Soft* (1993)
- *Simon the Sorcerer 2* de *Adventure Soft* (1995)
- …

4.4 Estructura

El llenguatge de programació utilitzat és el C/C++, a causa de l'antiguitat del projecte, moltes parts estan desenvolupades en C, mentre que les parts més noves ja les podem trobar en C++. Per tant, és una aplicació que ha estat programada utilitzant l'orientació a objectes.

Com ja hem dit en el punt anterior, l'*ScummVM* és una aplicació multiplataforma que està implementada i dissenyada tenint en compte aquest aspecte des d'un bon principi: fer que la feina del programador que vol portar l'aplicació a una altra plataforma sigui relativament senzilla.

L'*ScummVM* està estructurat en vàries capes que agrupen funcionalitats de l'aplicació. Inicialment es distingeixen 2 grans capes:

- *SCUMM*.
- Màquina Virtual.

A la figura 4.3 podem veure l'esquema principal de l'*ScummVM*.

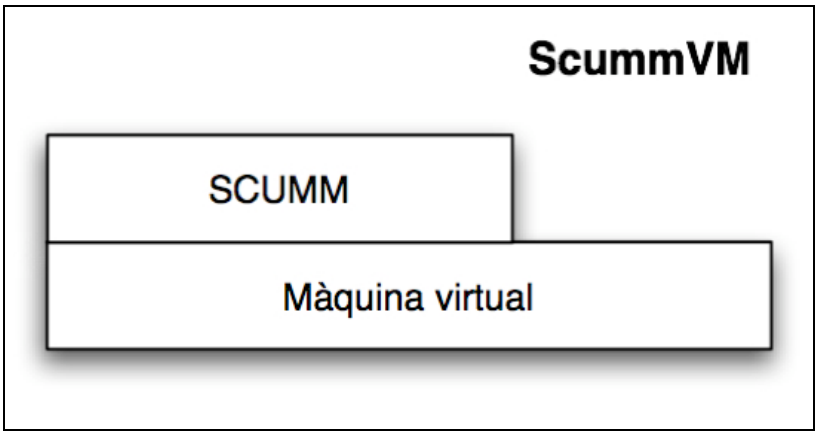


Fig 4.3 Diagrama de capes de l'ScummVM

4.4.1 *SCUMM*

Aquesta capa és la dedicada únicament i exclusivament a la interpretació dels fitxers de dades dels jocs i de l'emulació del sistema *SCUMM* de *LucasArts*, bàsicament es tracta del motor de l'aventura gràfica.

L'*SCUMM* està format per un conjunt de subcapes que engloben tot els submotors que fan que els jocs funcionin. Aquestes capes són les següents:

- ***SPUTM*:** *SCUMM Presentation Utility* o utilitat de presentació de l'*SCUMM*, és el nom real del motor.
- ***SCUMM*:** aquest, és realment el llenguatge de guions utilitzat per crear les històries dels jocs.
- ***IMUSE*:** *Interactive Music Streaming Engine*, és el sistema de control dinàmic dels fitxers musicals MIDI.
- ***SMUSH*:** és un format de compressió i reproducció de vídeo

- ***INSANE*:** és el sistema de control d'events.
- ***MMUCAS*:** és el sistema d'assignació de memòria utilitzat al joc *The Curse of Monkey Island*.

Tots aquests submotors o subsistemes, formen el conjunt del que anomenem *SCUMM* i son els encarregats d'interactuar conjuntament i interpretar les dades dels fitxers dels jocs per així poder-les enviar a la part de la màquina virtual. A la figura 4.4 podem veure un diagrama d'aquests motors.

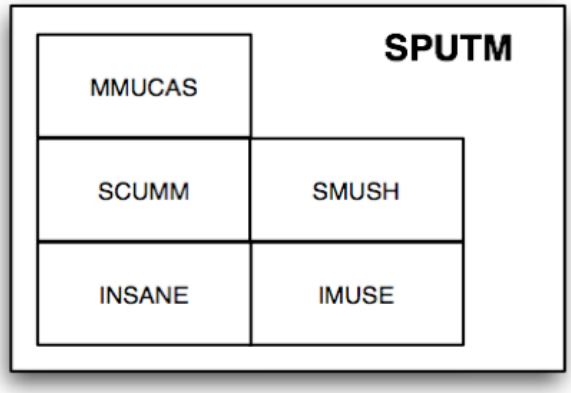


Fig 4.4 Diagrama d'elements de motors que formen part de l'SCUMM

4.4.1 Màquina virtual

Aquesta és la capa que engloba totes les diferents màquines virtuals de l'aplicació per cadascuna de les diferents plataformes que l'*ScummVM* suporta.

Cada màquina virtual reimplementa els diferents mètodes que utilitza l'*SCUMM* per fer les crides al sistema operatiu, utilitzant les crides apropiades de cada plataforma. A part d'això també implementa la gestió de l'entrada i sortida de dades per part de l'usuari i de l'aplicació.

És a la màquina virtual de cada plataforma on trobem la implementació de la interfície d'usuari de l'*ScummVM*. Això ho podem observar a la figura 4.5.

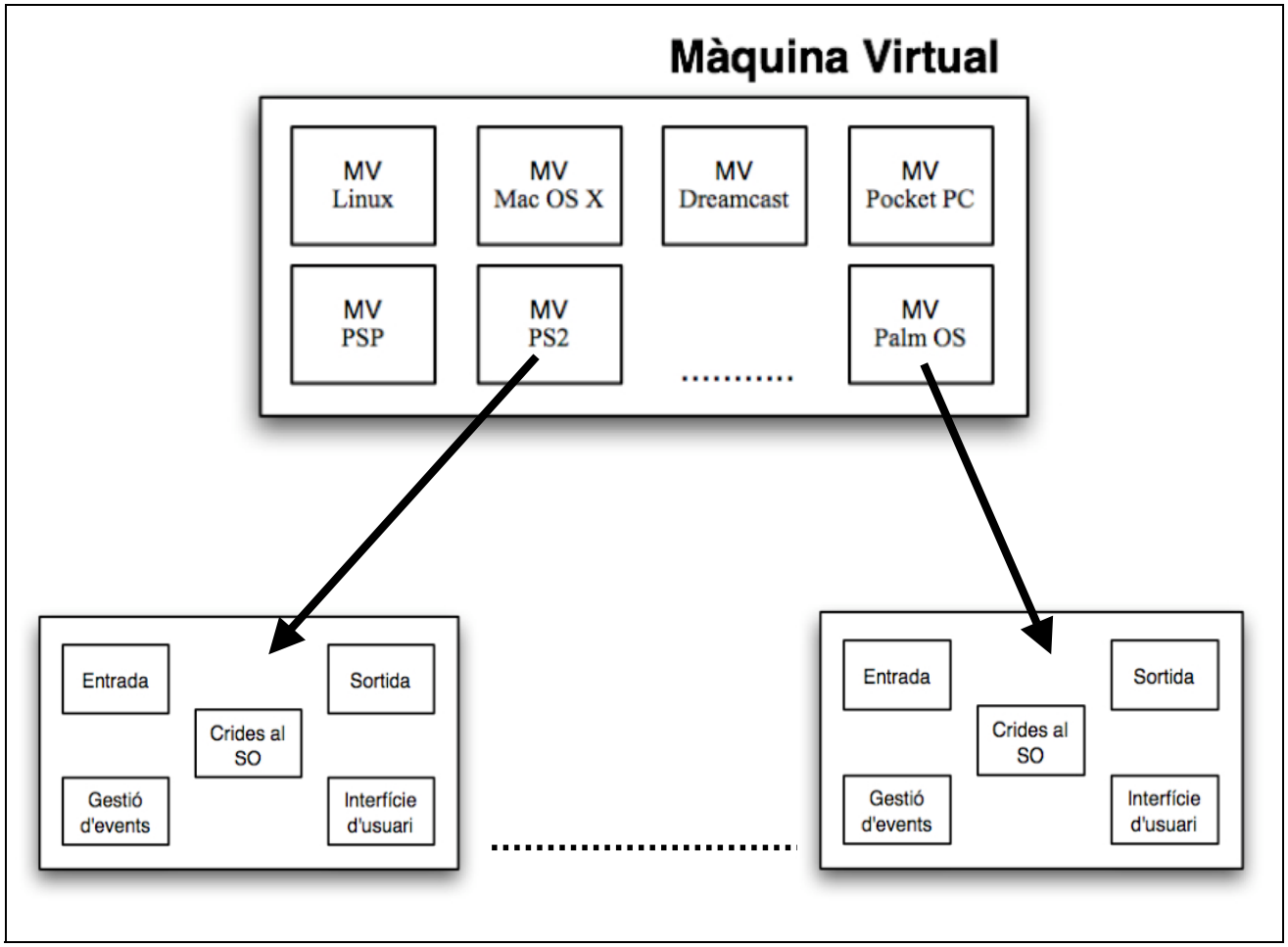


Fig 4.5 Diagrama d'elements de la màquina virtual

A dins de *l'ScummVM,* trobem que, per referir-se a les diferents màquines virtuals, utilitzen la paraula *backend*. En termes generals, les paraules *front end* i *back end* es refereixen als estats inicial i final del flux d'un procés. Aquests termes prenen significats més especialitzats quan els utilitzem en àrees més particulars. La idea general seria que el *frontend* és el responsable de recollir l'entrada de dades de l'usuari, que pot ser de moltes maneres diferents, i processar-la de tal manera que el *backend* la pugui utilitzar.

En el disseny d'aplicacions, un *frontend* és la part del programa que interacciona amb l'usuari i el *backend* és la part que processa les dades que li envia el *frontend*. Aquesta distinció dins del disseny de l'aplicació, és un tipus d'abstracció que ajuda a mantenir les diferents parts del programa separades.

Moltes aplicacions estan separades conceptualment, utilitzant aquests termes, però en la majoria dels casos el *backend* s'amaga a l'usuari.

Com acabem de veure, el sistema de capes de l'*ScummVM* separa totes les crides al sistema i entrada i sortida del veritable motor del joc

En el cas de *ScummVM,* ens trobem que els termes de *frontend* i *backend* s'utilitzen al revés, és a dir, tota la part que controla la interacció amb l'usuari l'anomenen *backend*, mentre que la part de l'aplicació que s'encarrega de rebre les dades que envia en aquest cas el *backend* i actuar en conseqüència ni tant sols l'anomenen, però si ho fessin segurament l'anomenarien *frontend*.

No se sap ben bé perquè han capgirat els termes, però tenint en compte que tot el que envolta el món de les aventures gràfiques de *LucasArts* està dotat de grans dosis d'humor, sembla que no és més que una manera més graciosa de fer referència al que realment són *frontends*.

4.5 Arquitectura de l'*ScummVM*

Per analitzar l'arquitectura, es va recórrer a la web dels autors (http://www.scummvm.org/), per buscar tota la documentació del projecte. Però ens vam trobar que la documentació de l'*ScummVM*, era pràcticament inexistent, únicament vàrem trobar els documents de:

- Instruccions d'instal·lació del programa.
- Llista dels jocs suportats.
- Mínima documentació de la història del projecte.
- Estàndard de programació que han d'utilitzar els desenvolupadors del projecte.
- Manual d'usuari de l'aplicació.

L'única documentació del codi font disponible, era la que es podia generar automàticament amb el programa *Doxygen*.

Un cop generada la documentació vam veure que l'*ScummVM*, està format per aproximadament unes 300 classes. A causa del volum d'informació, hem de dir que en aquest capítol només analitzarem les classes que creiem que són les més importants.

L'objectiu és poder entendre millor el funcionament de l'*ScummVM*, per poder atacar la problemàtica del nostre projecte.

4.5.1 Diagrama de classes

A la figura 4.6 hi trobem representades les classes més rellevants, amb la serva herència (també més rellevant) per poder entendre mínimament el funcionament de l'*ScummVM*.

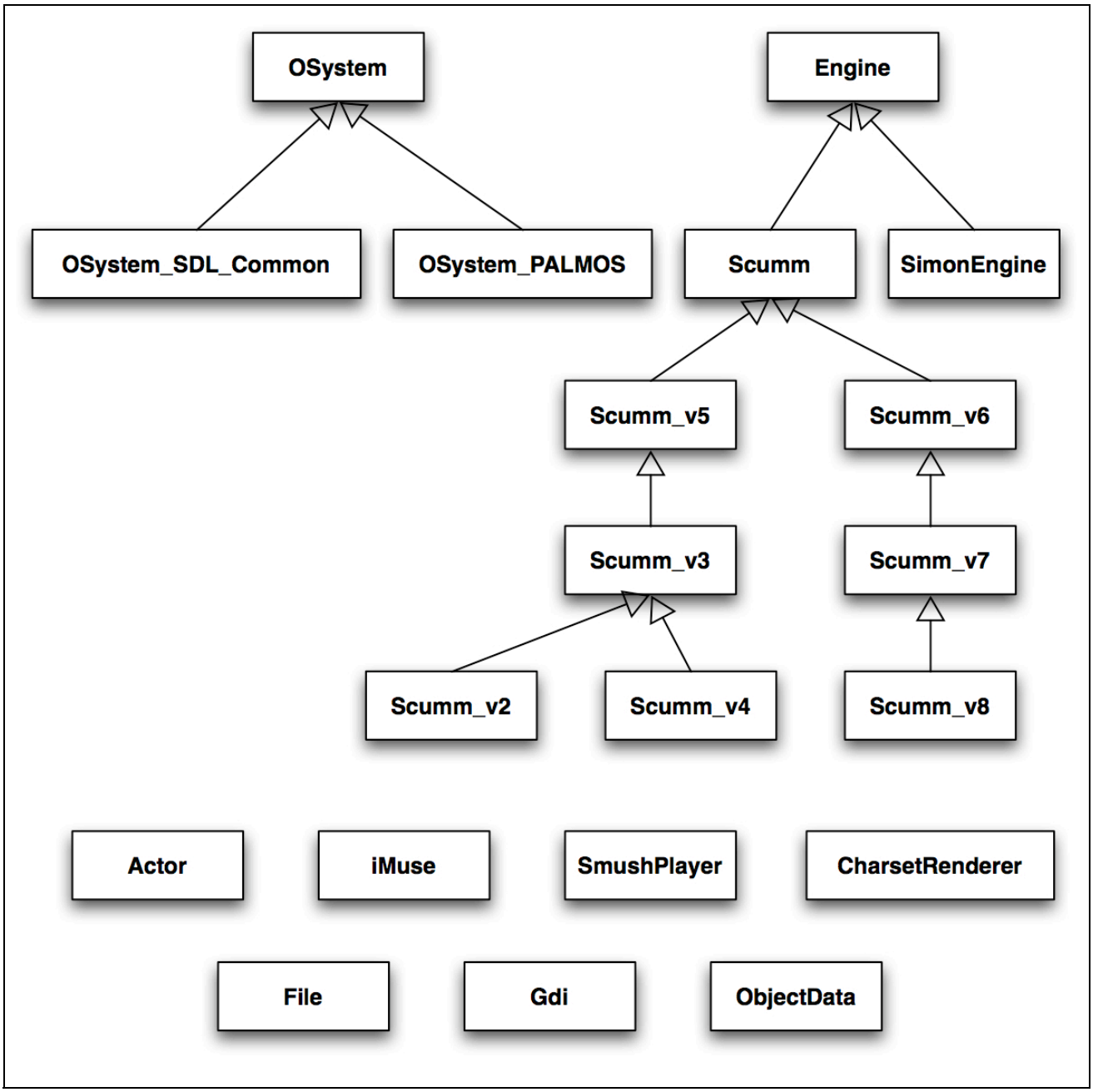


Fig 4.6 Diagrama de classes (herència).

Per complementar el diagrama de la figura anterior, hem decidit afegir també un diagrama de col·laboració d'aquestes classes més importants, per poder veure així les connexions entre classes. Aquest diagrama el trobem a la figura 4.7.

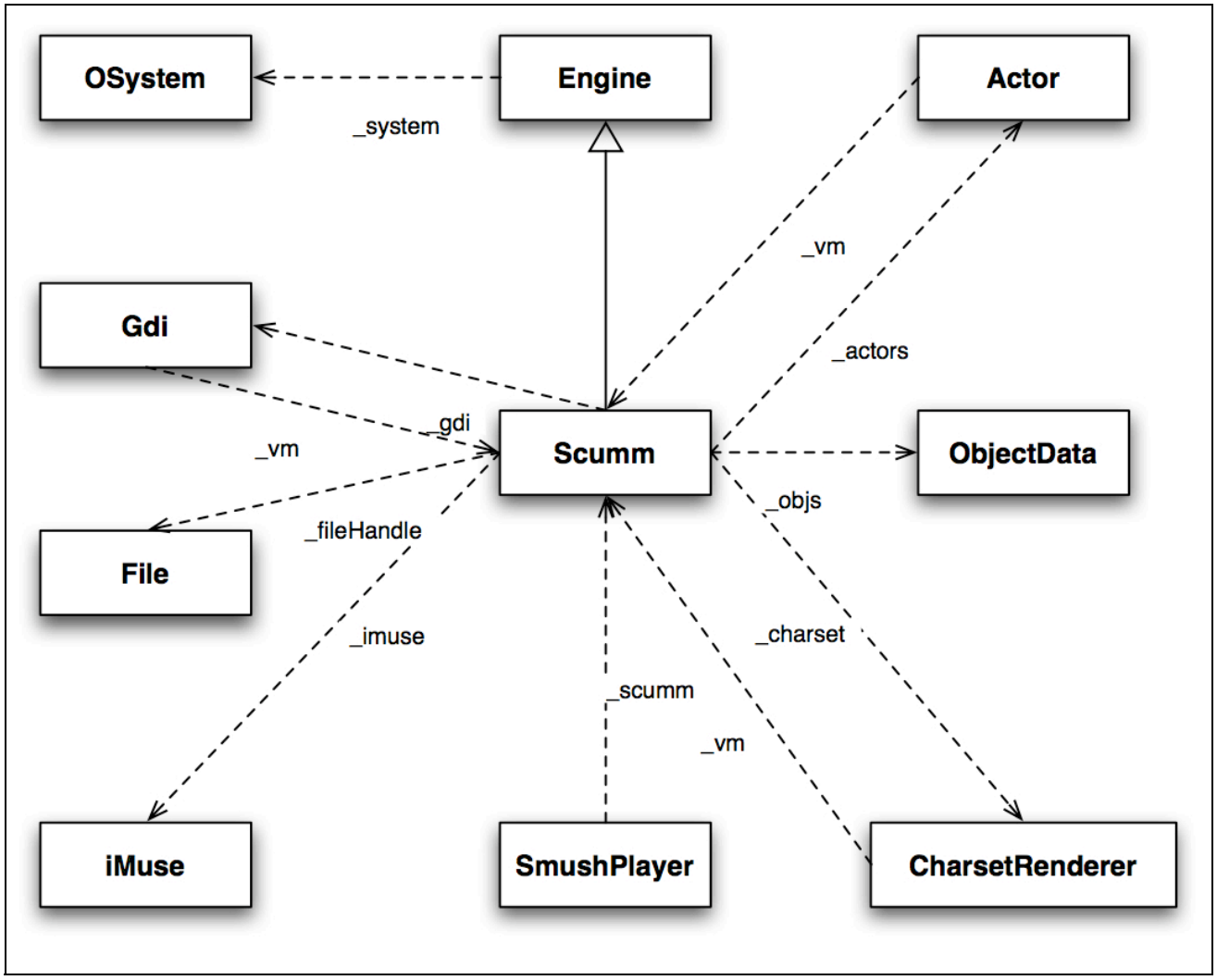


Fig 4.7 Diagrama de classes (col·laboració).

A continuació explicarem, classe per classe, quina funció realitza a dins de l'*ScummVM*.

Com podem observar en la figura anterior, la classe *Scumm* hereta de la classe *Engine*. Per això, explicarem primer totes les classes que interactuen amb la classe *Scumm* i després les de la classe *Engine.*

Classe *Scumm*

Podríem dir que aquesta, juntament amb la classe *Engine*, és la més important dins de l'*ScummMV*, ja que, com podem veure en el diagrama de col·laboració, es relaciona amb tota la resta de classes del sistema. Rep informació de tota la resta i la gestiona. Tota l'aplicació funciona al seu voltant.

Aquesta classe controla, entre d'altres, les classes següents:

- **Classe *Actor*:** Aquesta classe és la que emmagatzema totes les característiques referents als personatges dels jocs. La posició, la mida, el factor d'escalat, el color del text quan parla, etc. En definitiva tots els aspectes que defineixen un personatge d'un joc.
- **Classe Gdi:** La classe *Gdi* és l'encarregada de dibuixar un *bitmap*. Té, entre d'altres, implementats els mètodes de descompressió de *bitmaps*, i de dibuixar-los.
- **Classe *iMuse*:** La classe *iMuse*, és la que controla el so i la música dels jocs. Controla elements com el volum, quan ha de començar i acabar una música o un so, i accedeix al *driver Midi*. Aquesta classe interactua amb altres classes i subclasses que no surten al diagrama. Aquestes, s'encarreguen de l'accés i descompressió dels fitxers de so, i moltes altres funcionalitats, sempre referents al so.
- **Classe *SmushPlayer*:** És la classe encarregada de reproduir les escenes de vídeo i animacions que apareixen durant el joc. Té mètodes per accedir als elements de so i de vídeo i mesclar-los, reproduint les escenes animades. Igual que la classe anterior, *iMuse*, aquesta també interacciona amb altres classes que surten al diagrama, però que també tenen relació amb els vídeos i animacions dels jocs.
- **Classe *CharsetRenderer*:** Aquesta és la classe encarregada de *renderitzar* tot el text que surt per pantalla, és a dir, agafa el text que serà escrit a la pantalla i l'associa amb la seva imatge corresponent. De manera que el text que veiem per pantalla és una imatge. Independentment del sistema on s'utilitzi l'aplicació, el text sempre es veurà igual.
- **Classe File:** És la classe que gestiona l'accés als fitxers de dades dels jocs. Conté mètodes per obrir, llegir i guardar fitxers.
- **Classe ObjectData:** Aquesta classe, és la que defineix un objecte del joc, com pot ser una taula, un llibre, etc. Conté atributs com ara si es pot desplaçar, la seva mida, si els personatges hi poden interactuar, etc.

Classe *Engine*

Com podem veure a la figura 4.6, la classe *Engine* té dues subclasses:

- **Classe *Scumm:*** aquesta classe que acabem d'explicar, és la superclasse d'unes subclasses anomenades *Scumm_vx* on x és un número.

  Com ja hem explicat a l'apartat 4.3, el motor original de l'*SCUMM* ha anat variant amb cada joc. Aquestes subclasses implementen els mètodes de la superclasse *Scumm* necessaris per

adaptar l'intèrpet dels jocs a aquestes noves versions. Hi ha fins a 8 variacions de l'intèrpret. En funció del joc al que es vol jugar l'*ScummVM*, utilitza una versió o una altra de l'intèrpret.

- **Classe *SimonEngine*:** Com ja hem dit abans (veure apartat 4.3), l'*ScummVM* no només és capaç d'interpretar els jocs de *LucasArts* sinó que també és capaç d'interpretar altres jocs de l'època. Aquest és el cas del joc *Simon the Sorcerer*. La classe *SimonEngine*, reimplementa els mètodes necessaris perquè l'*ScummVM* sigui capaç d'interpretar-lo.

El conjunt format per totes aquestes subclasses i la superclasse *Scumm* agrupa els mètodes necessaris per a què la superclasse *Engine* faci la seva feina, que és la de motor que interpreta els fitxers de dades del joc, i que crea el joc a partir d'aquestes dades. Aquesta classe utilitza la implementació del motor adequada per cada joc, llegeix les dades dels fitxers dels jocs i crea les escenes, col·loca els personatges, executa les animacions, etc. En definitiva crearà l'entorn en el qual l'usuari podrà interactuar i actuarà en conseqüència utilitzant els paràmetres establerts per cada joc.

Aquesta classe, és per tant, l'encarregada de fer de motor de l'*ScumnmVM*. Per una banda controla el procés de creació de cada escena i control de personatges del joc i, per l'altre, controla les crides al sistema operatiu.

Classe *OSystem*: Aquesta és la superclasse que agrupa totes les crides al sistema operatiu i que gestiona tota l'entrada i sortida d'imatges, so i interacció amb l'usuari i amb el sistema mitjançant els events. Com podem veure en la figura 4.5, aquesta classe té una per cada sistema operatiu pel que funciona l'*ScummVM*, tot i que, només explicarem els dos amb els que hem treballat al projecte:

- **Classe *OSystem_PalmOS*:** Classe que implementa les crides al sistema operatiu *PalmOS*.
- **Classe *OSystem_SDL_Common*:** Utilitza les llibreries *SDL* (*Simple DirectMedia Layer*), per implementar les crides a la majoria dels sistemes operatius disponibles per PC.

4.5.2 Llibreries que utilitza l'*ScummVM*

L'*ScummVM*, a part d'implementar les seves classes i mètodes, utilitza una sèrie de llibreries que també són programari lliure. Les llibreries que necessita són les següents:

- ***SDL* (*Simple DirectMedia Layer*).** Aquestes llibreries multimèdia estan dissenyades per proporcionar al programador accés a baix nivell a l'àudio, al teclat, al ratolí i al *hardware* 3D mitjançant l'*OpenGL*. S'utilitzen en molts emuladors i reproductors *MPEG*. Aquestes llibreries suporten molts sistemes operatius diferents. Entre els més coneguts trobem:
    - Linux
    - Windows
    - MacOS X
- ***MAD* (*MPEG Audio Decoder*).** El *MAD*, és un descodificador d'audio *MPEG de* gran qualitat.
- ***LibMPEG2*.** És una llibreria per descodificar vídeo dels tipus *MPEG-2* i *MPEG-1*.
- ***Mathlib.*** És una llibreria de *PalmOS* conté un conjunt de funcions matemàtiques. Conté totes les funcions del *math.h* del llenguatge *C* per PC, i hi afegeix funcions de trigonometria, logarítmica, exponenciació i altres.

5 Estudi i implementació de la portabilitat

5.1 Introducció

En aquest capítol explicarem els passos que hem seguit per aconseguir portar l'*ScummVM* a *PalmOS*. Explicarem de manera detallada els problemes que ens hem trobat i les solucions que hem implementat per cadascun d'ells, aconseguint al final de tot, un *frontend* de l'*ScummVM* que ens permet fer funcionar l'aplicació en els dispositius *Palm* que hem descrit en els objectius del projecte.

5.2 Aspectes generals

Primer de tot s'han de detectar els problemes que representa portar una aplicació com l'*ScummVM,* dissenyada i desenvolupada per un PC, a una PDA amb sistema operatiu *Palm OS.* Com ja hem dit en la primera part d'aquest document, les limitacions d'una màquina d'aquestes característiques en comparació amb un PC són moltes i aquestes varien en funció de l'aplicació que volem utilitzar.

En el nostre cas d'estudi donarem solucions a:

- La problemàtica que genera el fet que la pantalla del PDA tingui una resolució de 160x160 a l'hora de mostrar imatges i textos renderitzats per a una resolució superior.
- La problemàtica que genera el fet del canvi de resolució per a la detecció d'events del tipus clic amb el ratolí.
- La problemàtica generada pel fet de no tenir ratolí, ni altres dispositius d'entrada i sortida tradicionals.
- Passos a seguir per poder portar aplicacions de PC a *Palm OS*.

Tot això s'ha de fer de la manera menys invasiva possible, per no desvirtuar ni canviar el funcionament de l'*ScummVM* original.

Un cop decidits els problemes a solucionar, hem dividit tot el que s'ha de fer per aconseguir l'objectiu final en un seguit de fases amb petits objectius, per així poder atacar millor aquest objectiu final.

Les fases són les següents:

1. Analitzar en detall el codi de l'*ScummVM* per PC.

   a. Mirar d'entendre a grans trets el funcionament de l'*ScummVM*.
   b. Localitzar i analitzar el funcionament de les funcions i/o mètodes encarregades de l'entrada/sortida de:
      i. Imatges.
      ii. Text.
      iii. Control d'events de teclat i ratolí.
   c. Localitzar les crides a aquestes funcions per poder establir els punts d'entrada de la nostra aplicació.
2. Reducció d'imatge sobre la versió de PC:
   a. Sense filtrat.
   b. Amb filtres.
3. Separar i controlar l'entrada/sortida en PC.
   a. Separar el text de les imatges.
   b. Controlar els events de ratolí i teclat.
4. Definir l'aplicació, els objectes, els mètodes i les funcions que s'implementaran.
5. Portar a *PalmOS*.
   a. Instal·lar el compilador i les llibreries necessàries per desenvolupar per *PalmOS*.
   b. Compilar l'*ScummVM* per *PalmOS*.
   c. Definir funcionament i interfície de la nostra aplicació.
   d. Adaptar el codi generat per PC i implementar la part de Palm, per a què l'aplicació funcioni d'acord amb la definició del punt 4.

Com es pot observar en l'esquema anterior, veiem que hi ha tota una part de feina que es desenvolupa sobre la versió per PC de l'*ScummVM*. Si observem bé els punts de l'esquema, veiem que el que s'ha de fer en els quatre primers és completament independent de la plataforma per la qual treballem. Dit això, vàrem escollir treballar primer amb la versió per PC, perquè jo personalment m'hi sentia més còmode.

5.3 Anàlisi del funcionament de l'*ScummVM*

Com ja hem dit a l'apartat anterior, després d'entendre el funcionament a grans trets de l'*ScummVM*, hem de localitzar i analitzar el funcionament dels mètodes relacionats amb:

- **Tractament de les imatges dels jocs:** Necessitem saber quins mètodes i quines classes treballen amb les imatges dels jocs i buscar des de quins objectes s'hi accedeix i es manipulen les imatges. També hem de saber a quina resolució treballen els jocs, per poder decidir quins mètodes de reducció d'imatge hem d'implementar i quin és el ràtio de reducció necessària.
- **Tractament del text dels jocs:** Hem d'esbrinar com ho fa l'*ScummVM* per gestionar el text que apareix als jocs, en quin format trobem aquest text i quines classes i mètodes que el tracten.
- **Control d'events generats per l'usuari:** Hem de descobrir quins són i com es gestionen els events que genera l'usuari per poder interactuar amb els jocs. Quins són els mètodes que els tracten i com és tracten.

5.3.1 Tractament de les imatges dels jocs

Pel que fa a les imatges, el primer que ens interessa és la seva resolució original. Ens trobem que *l'ScummVM* dóna l'opció d'escollir entre vàries resolucions i varis mètodes de filtrat d'imatge. Per obtenir aquesta informació no cal analitzar el codi, la podem trobar al fitxer *readme.txt* que vé amb el paquet de l'executable de l'*ScummVM*.

A continuació mostrem unes quantes de les opcions de resolució implementades a l'*ScummVM*:

- ***Normal* – 320x200:** aquesta és la resolució nativa de les imatges dels jocs originals. Aquest mode no implementa filtrat d'imatge.
- **2x – 640x480 píxels:** aquest mode duplica la resolució original sense aplicar cap filtre d'imatge. És el mode de funcionament per defecte de l'*ScummVM*.
- **3x – 960x600 píxels:** aquest mode triplica la resolució original de les imatges sense aplicar cap filtre d'imatge.
- ***2xsai* – 640x480:** el mode 2xSaI filtra les imatges mitjançant aquest algorisme de filtrat i duplica la resolució original. Aquest algorisme utilitza tècniques d'*antialiasing* i obté millors resultats que el filtrat bilinear estàndard.

- ***supereagle* – 640x480:** aquesta modalitat fa el mateix que l'anterior, duplica la imatge i aplica un filtre amb *antialiasing*. Cal dir que és més lent.
- ***admame2x* – 640x480:** aquesta opció també duplica la mida de la imatge i aplica un mètode de filtrat d'*antialiasing*.
- ***admame3x* – 960x600:** aquest mode utilitza el mateix mètode de filtrat que l'anterior però en aquest cas per triplicar la mida de la imatge.
- ***tv2x* – 640x480:** el mode tv2x dobla la resolució original de la imatge, amb un filtrat mitjançant el sistema d'*horizontal scanlines*.
- ***dotmatrix* – 640x480:** l'escalat del *dotmatrix* és el mateix que el de la majoria, duplica la imatge original i ofereix un efecte de matriu de punts.

Per entendre millor els filtrats dels diferents modes de pantalla que ofereix *l'ScummVM,* podem veure la figura 5.1 que vé a continuació. Per aquest exemple hem escollit tots els modes que treballen al doble de la resolució.

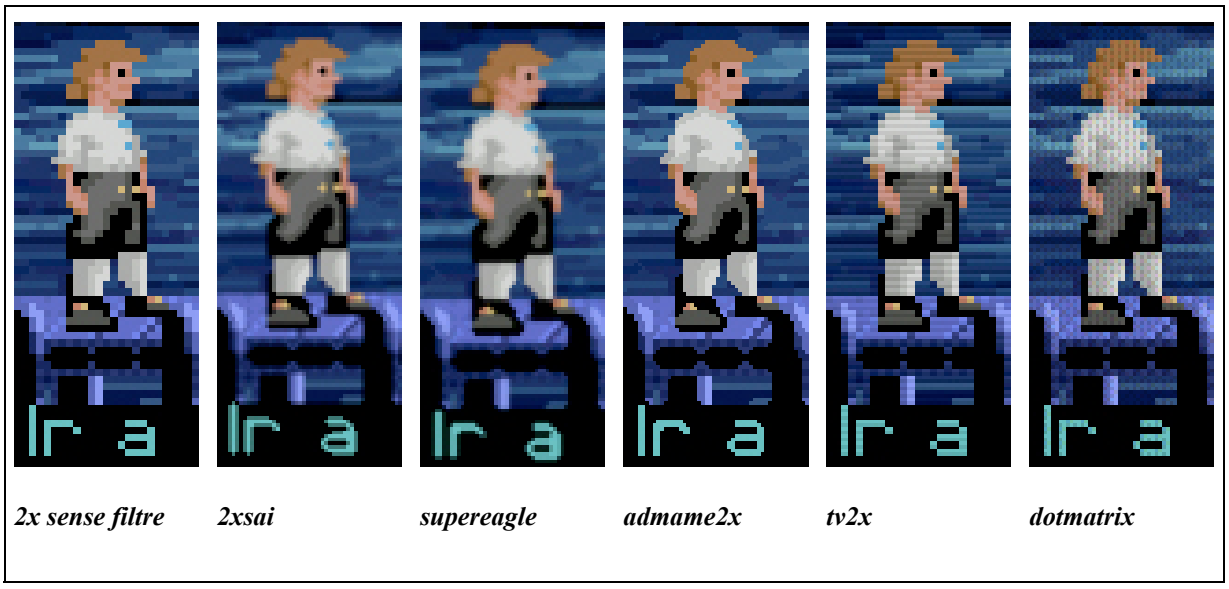


Fig 5.1 Captures de pantalla dels diferents filtres utilitzats pels modes gràfics de l'ScummVM.

Com acabem de veure, la resolució original de les imatges dels jocs és 320x200 píxels sense cap mena de filtrat. Per tant, la nostra tasca serà reduir la imatge a la meitat de la seva resolució. Per poder fer-ho, hem hagut de buscar al codi de l'*ScummVM* quin és el mètode encarregat de dibuixar les imatges a la pantalla.

El mètode en qüestió és *update_screen()* i funciona de la següent manera:

- Recull la informació necessària per generar l'adreça de memòria on hi ha emmagatzemada la imatge.
- Llegeix la mida de la imatge que es dibuixarà per pantalla.
- Genera l'adreça de memòria de la targeta de vídeo on s'enviarà la imatge per a què es dibuixi.
- Per acabar, envia al processador d'escalat totes aquestes dades, per a què enviï la imatge a la memòria de vídeo.

El processador d'escalat és una funció que varia depenent de quin és el mètode de renderitzat escollit per l'usuari en el moment de carregar un joc. La funció que realitza el renderitzat de la imatge és l'encarregada d'aplicar a cada imatge els filtres que hem explicat abans i després dibuixar-la. El funcionament de les funcions de renderitzat és bàsicament el mateix.

- Reben l'adreça de memòria d'on hi ha la imatge a dibuixar.
- Apliquen les transformacions del filtratge escollit.
- Finalment, envien la imatge a l'adreça de vídeo, d'on el sistema operatiu recollirà la imatge i la dibuixarà per pantalla.

El mètode *update_screen()* forma part de la classe *OSystem*. Com ja hem dit abans, aquesta classe és l'encarregada de tota l'entrada i sortida i de la comunicació amb el sistema operatiu. Aquesta és una superclasse, per tant el mètode *update_screen()* el trobem reimplementat a dins de cada *backend* de les diferents plataformes. En el cas de la implementació per PC, el trobem a dins del *OSystem_SDL*.

Per entendre millor el procés, hem fet un diagrama de seqüència que podem veurem a la figura 5.2.

En el diagrama es pot veure el procés que segueix per dibuixar la pantalla des de l'inici del programa.

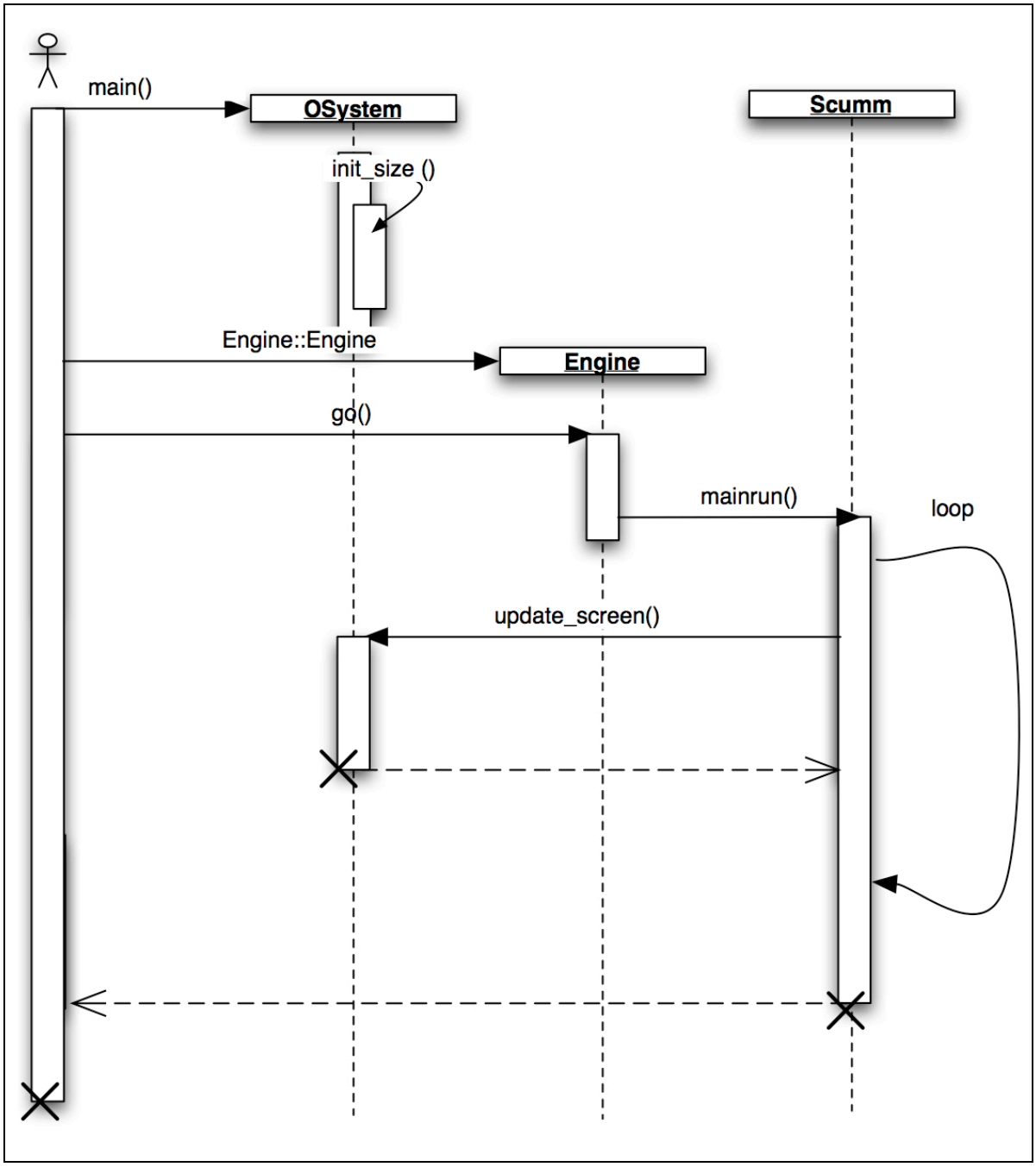


Fig 5.2 Diagrama de seqüència.

A l'inici de l'execució de *ScummVM*, trobem que aquest executa el mètode *main()*, aquest mètode identifica el sistema i instancia la classe *OSystem* d'aquest sistema, creant així l'objecte *OSystem*. Un cop creat l'objecte, inicialitza les variables de pantalla mitjançant l'*init_size()*.

Un cop fet això, l'execució del programa segueix, i es crea l'objecte *Engine*. Un cop creat aquest objecte, el *main* n'executa la rutina *go()*. Aquesta rutina inicialitza els atributs de l'objecte *Engine* i executa el mètode *mainrun()* de l'objecte *Scumm*. Aquest mètode és el bucle principal del

programa, i va cridant els mètodes necessaris dels diferents objectes en funció de l'execució del joc. S'executa contínuament fins al final de l'execució del programa.

Un d'aquests mètodes és l'*update_screen()*. El que fa l'*ScummVM* dins dels dos bucles principals, és recollir tota la informació necessària i generar la imatge de l'estat actual del joc, que es dibuixarà a la pantalla. Aquesta imatge s'emmagatzema en la memòria. Un cop generada tota la imatge, s'executa l'*update_screen()*.

Aquest mètode rep un punter que inidica la posició inicial de la memòria on està guardada la imatge, la mida de la imatge i l'adreça de memòria de la memòria de vídeo. *L'update_screen()*, copiarà els w*h bytes (w: número de bytes d'amplada de la imatge, i h:número de bytes d'alçada de la imatge) des de l'adreça d'origen a l'adreça de la memòria de vídeo, d'on la targeta de vídeo la llegirà per dibuixar-la.

El mètode *update_screen()* s'executa un cop per cada volta del bucle *mainrun()* fins al final del programa, de manera que la pantalla es va actualitzant contínuament amb la imatge corresponent.

5.3.2 Text dels jocs

Pel que fa al text, el primer que hem de dir és que *l'ScummVM* fa la distinció de quatre tipus diferents:

- Text de les converses entre personatges, o del narrador.
- Text dels menús d'acció que hi ha per cada joc.
- Text escollit del menú d'acció o de les converses interactives.
- Text interactiu de les converses.

5.3.2.1 Tipus de text a l'*ScummVM*

Per entendre millor quins són els tipus de text exactament, a continuació farem una petita explicació amb una captura de pantalla on es marca el text del que estem parlant.

Text de les converses entre personatges, o del narrador:

És tot aquell text que apareix a les presentacions dels jocs o a les escenes animades dins dels jocs. També és el text que apareix en la pantalla durant les converses dels personatges (veure figura 5.3).

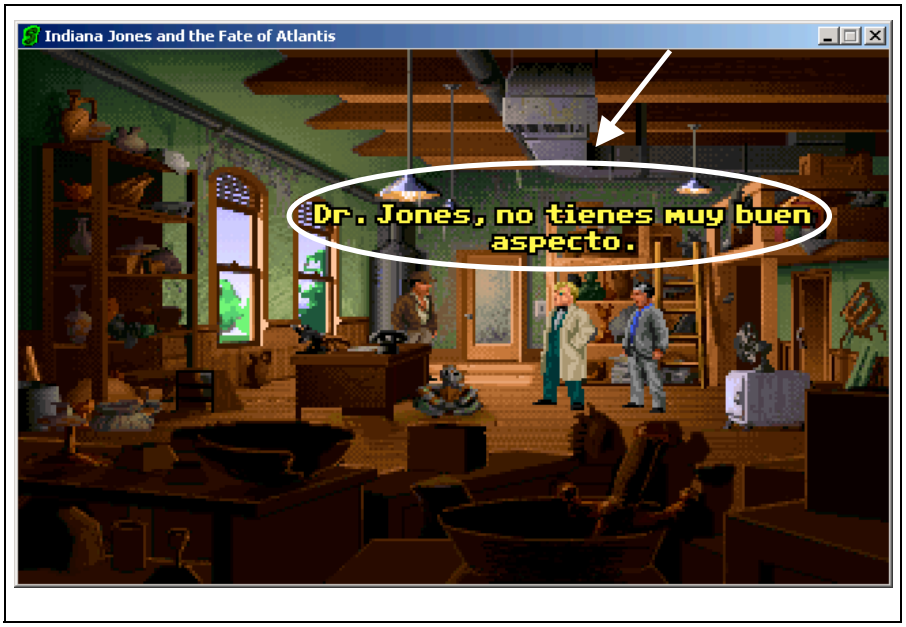


Fig 5.3 Text de les converses entre personatges.

Text dels menús d'acció que hi ha per cada joc:

És tot el text de les accions que pot realitzar l'usuari amb el personatge del joc (veure figura 5.4). Accions com *abrir, coger, cerrar, usar*, etc.

Fig 5.4 Text del menú d'acció del joc.

Text de l'opció escollida del menú d'acció:

És el text que apareix entre la imatge del joc i les accions del menú interactiu i que ens indica quina de les opcions del menú d'accions tenim seleccionada en cada moment (veure figura 5.5).

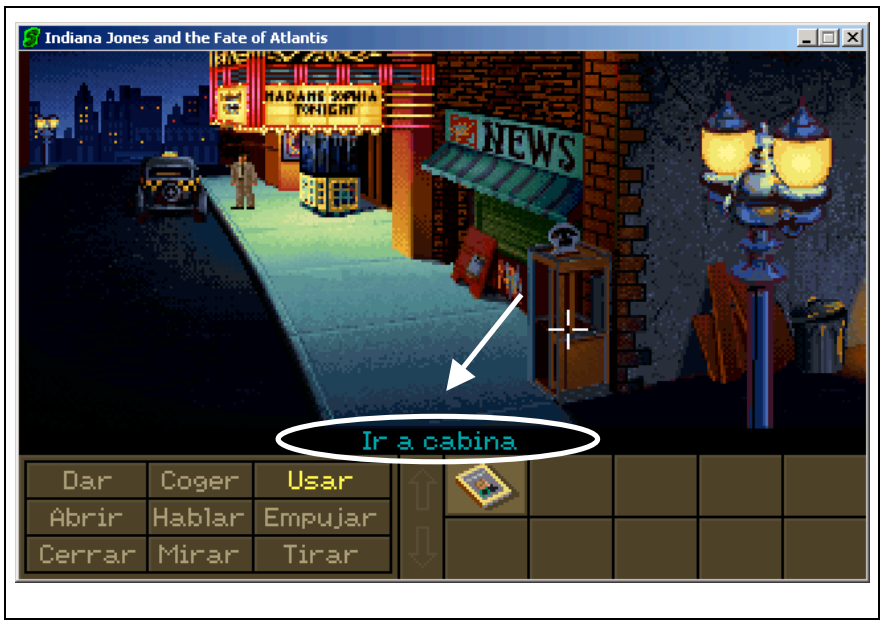


Fig 5.5 Text de l'opció escollida del menú d'accions.

Text interactiu de les converses:

És el text que apareix a la part inferior de la pantalla quan el jugador escull l'acció de parlar amb algun personatge del joc. Són totes les frases que apareixen en aquest moment i que el joc ens permet escollir a l'hora de parlar amb algun personatge (veure figura 5.6).

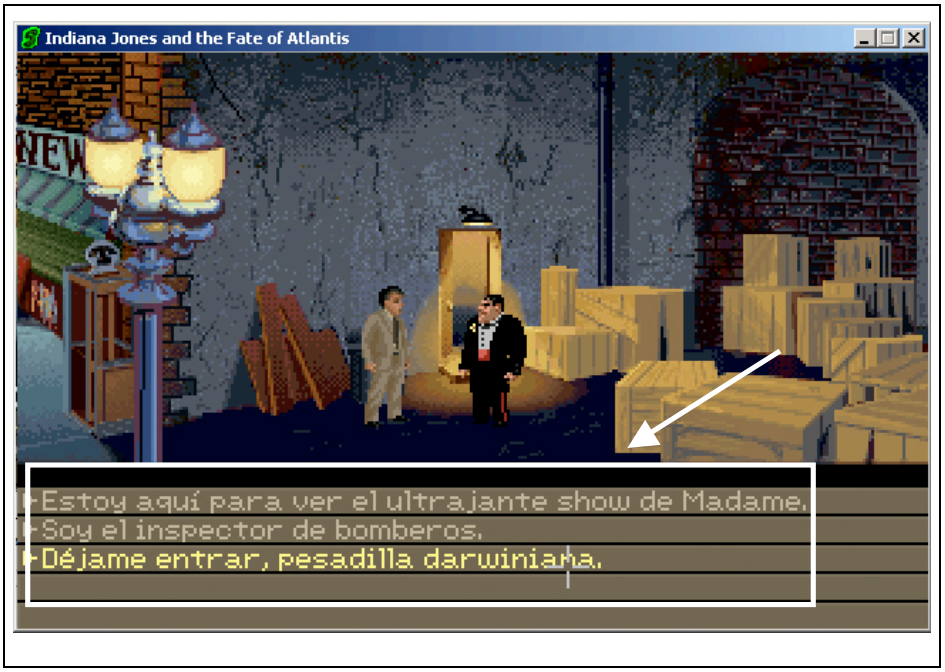


Fig 5.6 Text interactiu de les converses.

5.3.2.2 Funcionament del text

Un cop explicats els diferents tipus de text, hem de dir que l'*ScummVM* utilitza uns quants mètodes dins del codi font per tractar cada tipus de text. Aquests mètodes seran els candidats per establir els punts d'entrada de la nostra aplicació i així poder tractar el text per separat. Tot el text que apareix per la pantalla es llegeix dels fitxers de dades dels jocs en format text, i es tracta d'aquesta forma tota l'estona fins al moment de mostrar-lo per pantalla. És en aquest moment quan el text deixa de ser text pur i es converteix en imatge.

El fet que el text s'acabi convertint en imatge fa que nosaltres haguem de tractar el text pel nostre compte, ja que si no ho fem així, quan reduïm les imatges del joc, el text es tornarà il·legible (veure figura 5.7). La solució que proposem és capturar el text tal com ens vé donat dels fitxers de dades i utilitzar la tipologia de text del PC en un primer moment, i més endavant la tipologia de *PalmOS*. D'aquesta manera tenim que el text deixa de ser dependent de la resolució de les imatges. Més endavant, en aquest mateix capítol, expliquem més detalladament aquest tractament.

consola de l'ScummVM

Monkey Island 2: Le Chuck's Revenge

Fig 5.7 Captures que mostren com queda el text il·legible després d'una reducció sense filtres.

Pel que fa als mètodes que tracten el text dins de *l'ScummVM,* ens trobem que els textos de les converses entre els personatges i de l'opció escollida són tractats pel mètode *CHARSET_1()*. Aquest mètode és l'encarregat d'omplir l'estructura de dades amb la informació del text (mida de lletra, color, tipografia, etc.) que es dibuixarà a la pantalla. Un cop fet tot això, crida per cada lletra de la frase que s'ha de dibuixar al mètode *printChar()*. Aquest és l'encarregat de transformar cada lletra amb la seva imatge corresponent.

La conversió de text a imatge es fa a partir d'un conjunt d'imatges que representen les lletres i els símbols. Aquest conjunt d'imatges s'anomena *charset* i és diferent per cada joc, i dins de cada joc poden haver-hi varis conjunts d'imatges en funció del tipus de text. Pel que fa al text del menú d'accions i al text interactiu de les converses, trobem que el tractament que se'ls hi dóna és el mateix, però que el mètode que els tracta és un altre. Aquest mètode és el *drawString()*. Com acabem de dir, el sistema és el mateix que l'anterior, el mètode *drawString()* rep la informació del text que es dibuixarà i omple l'estructura de dades pertinent, per després fer la crida al mètode *printchar()*, que serà l'encarregat de transformar els caràcters en imatges.

Els mètodes *drawString()* i *CHARSET_1()* formen part de la classe *Scumm*, mentre que el mètode *printChar()* forma part de la classe *CharsetRenderer*, que és la que conté tots els mètodes i funcions necessaris per transformar el text en imatge, com ja hem comentat anteriorment (veure secció 4.5.1).

Per entendre millor el procés de mostrar el text, hem realitzat un diagrama de seqüència que es pot veure a la figura 5.8

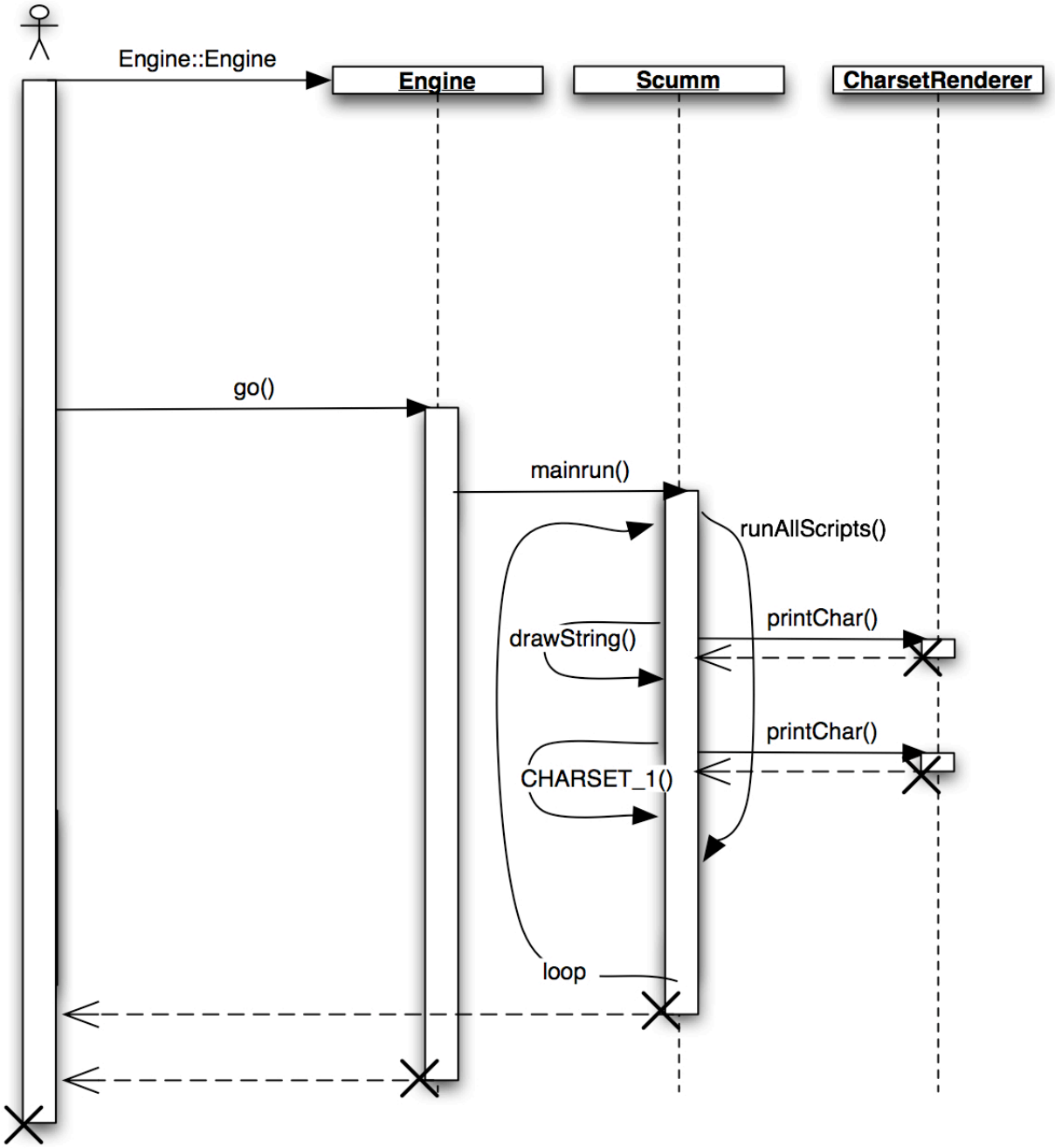


Fig 5.8 Diagrama de seqüència del text

A l'inici de l'execució de *ScummVM*, es crea l'objecte *Engine*. Un cop creat aquest objecte, el *main* n'executa la rutina *go()*. Aquesta rutina inicialitza els atributs de l'objecte *Engine* i executa el mètode *mainrun()* de l'objecte *Scumm*. Aquest mètode és el bucle principal del programa. Va cridant els mètodes necessaris dels diferents objectes en funció de l'execució del joc i s'executa contínuament fins al final de l'execució del programa.

Un d'aquests mètodes és l'*runAllscripts()*, que pertany a l'objecte *Scumm*. Aquest mètode identifica la versió del joc i executa els *scripts* o subprogrames que s'encarreguen d'interpretar les dades dels fitxers dels jocs en cada etapa de l'execució. Aquest mètode està a dins del bucle i per tant s'executa contínuament. Quan el mètode *runAllscripts()* llegeix de les dades dels jocs que s'ha d'escriure algun text per la pantalla, l'identifica i utilitza un dels dos mètodes següents en funció de si és el text de les accions del joc o bé si és el text de les converses entre personatges.

Aquests mètodes són el *CHARSET_1* i el *drawString()*. Tan un com l'altre donen format al text que s'ha d'escriure i acaben cridant al mètode *printChar()* de l'objecte *CharsetRenderer*. Aquest mètode dibuixa a la pantalla cada caràcter del text.

5.3.3 Control d'events generats per l'usuari

Quan parlem del control d'events generats per l'usuari, ens referim a tots aquells events resultat de la interacció de l'usuari amb el joc. Després de carregar un joc a l'*ScummVM*, aquest queda aturat esperant que l'usuari controli el personatge del joc.

Les aventures gràfiques de *LucasArts* són jocs desenvolupats per PC, per tant no és d'estranyar el fet que mitjançant el ratolí i/o del teclat puguem controlar els jocs. Cal dir però, que el dispositiu més utilitzat és el ratolí, però els jocs oferien la possibilitat de ser controlats mitjançant certes tecles del teclat.

Mitjançant aquests dispositius, l'usuari pot controlar els moviments del personatge (caminar cap a un costat o altre de la pantalla), les seves accions (obrir, tancar, parlar, …), o bé el menú d'opcions de cada joc (guardar partida, treure el so, sortir, …).

Dit això, queda clar que ens hem de centrar en gestionar de manera correcta els events provocats per aquests dispositius.

Com hem definit en els objectius del projecte, la PDA amb la que treballem no disposa de teclat ni de ratolí. Per tant, hem de buscar altres dispositius d'entrada i sortida que ens facin el servei d'aquests. Els dispositius escollits han estat els següents:

- Els botons de la PDA per substituir el teclat.
- El llapis òptic de la PDA per substituir el ratolí.

Per poder capturar i tractar com volguéssim els events de teclat i els events de ratolí, hem de localitzar en el codi de l'*ScummVM* quin mètode era l'encarregat de gestionar-los. El mètode en

qüestió és el *waitForTimer()*. Aquest mètode, que forma part de la classe *Scumm*, està escoltant els events durant tota l'execució del programa. Per capturar els events, utilitza el mètode *poll_event()*, que forma part de l'objecte de sistema. Per tant s'haurà d'implementar un mètode *poll_event()* diferent per cada plataforma. Aquest mètode *poll_event()* recull i tracta els events de ratolí directament, però pels events de teclat utilitza el mètode *processKbd()*, que també pertany a la classe *Scumm*.

A continuació, podem veure la figura 5.9, que és un diagrama de seqüència del control dels events, per entendre millor aquest procés.

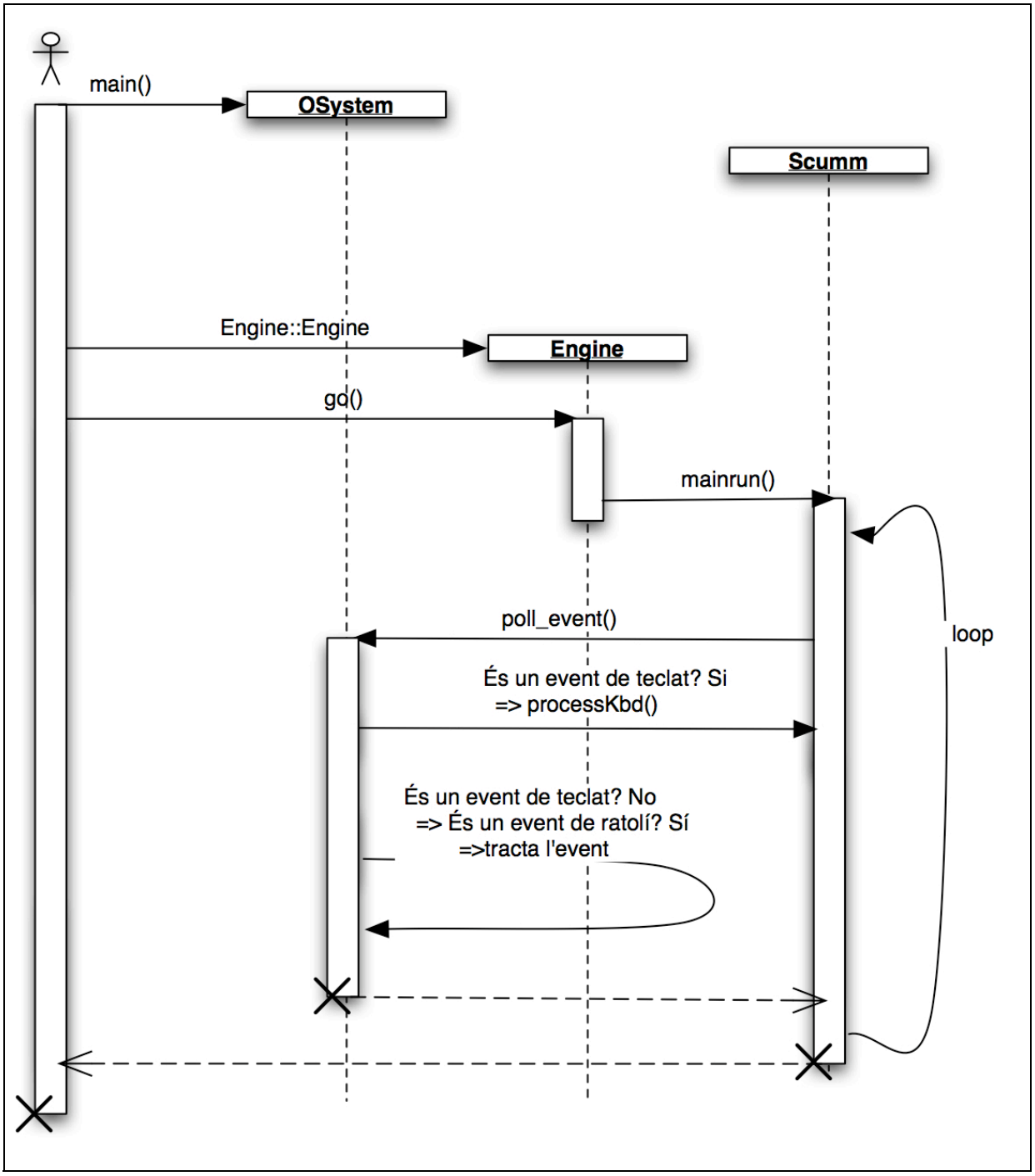


Fig 5.9 Diagrama de seqüència del tractament d'events

A l'inici de l'execució de *ScummVM*, trobem que aquest executa el mètode *main()*, aquest mètode identifica el sistema i instancia la classe *OSystem* d'aquest sistema, creant així l'objecte *OSystem*.

Un cop fet això l'execució del programa segueix, i es crea l'objecte *Engine*. Un cop creat aquest objecte, el *main* n'executa la rutina *go()*. Aquesta rutina inicialitza els atributs de l'objecte *Engine* i executa el mètode *mainrun()* de l'objecte *Scumm*. Aquest mètode és el bucle principal del

programa. Va cridant els mètodes necessaris dels diferents objectes en funció de l'execució del joc i s'executa contínuament fins al final de l'execució del programa.

Per tractar els events, *l'ScummVM* utilitza el mètode *poll_event()*. Però aquest mètode no s'encarrega de tractar tots els events possibles, si no que només tracta els que són rellevants pels jocs que interpreta l'*ScummVM*. El que fa és rebre l'event, identificar-lo i actuar en conseqüència. Si és un event de teclat, crida al mètode *processKbd()* de l'objecte *Scumm*. Mentre que si és un event de ratolí el tracta el mateix *poll_event()*.

El mètode *poll_event()* s'executa un cop per cada volta del bucle *mainrun()* de manera que està rebent i tractant els events contínuament.

Com ja hem dit, nosaltres ens centrarem en els de teclat i ratolí. Però, sobretot, amb els de ratolí, perquè, a causa de la reducció d'imatge, s'haurà de reajustar la posició dels clics de l'usuari, ja que si no l'*ScummVM* rebrà coordenades errònies, ja que el sistema de coordenades intern està preparat per treballar a 320 x 200. Amb els events de teclat no tindrem aquest problema de les coordenades, però si que haurem de controlar-ne uns quants per poder assignar la seva funcionalitat als botons de la Palm. Més endavant expliquem exactament com tractem els events de teclat i ratolí (veure apartat 5.5.2).

5.4 Reducció d'imatge

Una imatge digital està formada per un conjunt de píxels (unitat mínima en la qual podem descompondre una imatge digital). Mitjançant la resolució d'una imatge podem saber el número de píxels que hi ha. Per tant si diguéssim que una imatge té 2x2 píxels de resolució, estaríem dient que la imatge conté 4 píxels. El primer número de la resolució ens indica l'alçada de la imatge i el segon número ens indica l'amplada. Amb aquesta alçada i amplada podem generar una matriu de píxels (veure figura 5.10). On a cada intersecció de fila i columna hi tenim el color d'un píxel.

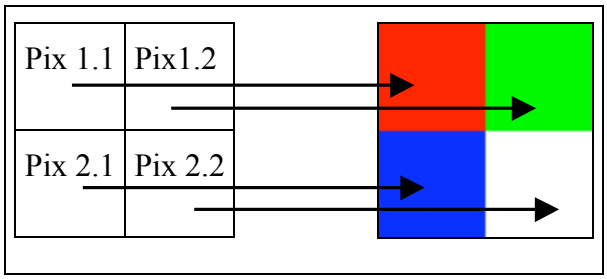


Fig 5.10 Detall d'una matriu de píxels.

Com ja hem dit en punts anteriors, per aconseguir els objectius del nostre projecte hem d'aplicar una reducció d'imatge en temps real a les imatges del joc. Aquesta reducció s'ha de fer de 320x200 píxels a 160x100 píxels, per tant, nosaltres hem de reduir les imatges del joc a la meitat de la seva resolució original.

La reducció d'imatge no és un fet trivial, ja que comporta la pèrdua de dades de la imatge original. Aquesta pèrdua d'informació, ocasiona distorsió i pèrdua de definició. L'efecte més habitual de la reducció és l'anomenat *aliasing*.

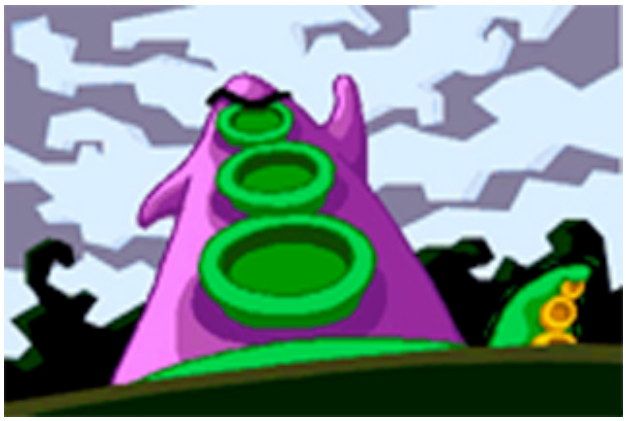
L'*aliasing,* és un efecte indesitjable que fa que senyals contínues diferents entre sí, es tornin indistingibles quan es mostregen digitalment. En l'àmbit dels gràfics per computador, l'*aliasing* es pot observar quan veiem que algunes corbes o línies inclinades de la imatge presenten un efecte de serra o d'esglaó. El fet de reduir la imatge a la meitat, fa que la freqüència de mostreig es redueixi a la meitat, i aquest fet provoca l'aparició de l'*aliasing* (veure figura 5.11).

Imatge original a 320x200 ***Imatge reduïda a 160x100 sense filtre antialiasing***

Fig 5.11 Exemple de l'aparició d'aliasing quan reduïm una imatge.

Per poder evitar això, la freqüència de mostreig ha de ser com a mínim el doble de la freqüència màxima que es pot trobar en el senyal original, tal com diu el teorema de *Nyquist*. Això vol dir, que si volem reduir aquest efecte, no ens podem limitar a descartar un de cada dos píxels, sinó que el que hem de fer és utilitzar la informació que ens proporcionen els píxels del voltant al que volem descartar, i crear a partir d'aquesta informació un nou píxel. Existeixen molts mètodes de filtrat diferent, cadascun dels quals aplica les seves fórmules algebraiques per aconseguir calcular el valor del nou píxel. L'objectiu d'aplicar aquests filtres és el d'obtenir una imatge el més semblant a l'original possible (veure figura 5.12) i reduir aquest efecte de serra que apareix a la imatge.

Imatge original a 320x200



Imatge reduïda a 160x100 utilitzant un filtre antialiasing

Fig 5.12 Exemple de reducció d'imatge utilitzant un filtre antialiasing.

El procés de reducció d'imatge utilitzant mètodes de filtrat per aconseguir reduir l'*aliasing* pot arribar a ser molt costós. Tot depèn de la complexitat dels càlculs que apliquem al conjunt de píxels seleccionats per aconseguir obtenir el nou píxel.

Existeixen molts tipus mètodes de filtrat per la reducció d'imatge, però a causa de les limitacions en quant a memòria i velocitat del processador del nostre PDA, hem decidit implementar mètodes de reducció poc costosos però que donen uns resultats bastant bons.

Els mètodes de reducció d'imatge que hem decidit implementar són els següents:

- Reducció 2:1 sense filtrat.
- Reducció 2:1 amb matriu de pesos.
    - Matriu de 2x2.
    - Matriu de 3x3.

Excepte el mètode de la reducció sense filtrat, els altres dos estan basats en matrius de pesos. Aquesta matriu també és anomenada màscara. Aquest sistema, serveix per fer una ponderació del valor de cada píxel, en funció de la seva posició dins la màscara. S'assigna un valor numèric que marca la importància del píxel per cada posició de la matriu de pesos (veure apartat 5.4.2).

5.4.1 Procés de reducció sense filtrat

El procés de reducció sense filtres té l'avantatge de ser el més senzill i ràpid que hi ha, però amb l'inconvenient de ser el que pitjors resultats dóna, ja que la pèrdua d'informació resulta considerable i l'aparició d'*aliasing* en aquest cas està assegurada.

Com hem explicat anteriorment, les imatges tenen una amplada (W) i una alçada (H) expressada en número de píxels.

El sistema és molt simple, el primer que hem de fer és calcular el ràtio de reducció. El ràtio ens indicarà el número de píxels que hem de descartar. El càlcul per fer això és molt senzill. Suposem que tenim una imatge de resolució W_1xH_1, i la volem reduir a una imatge de mides W_2xH_2. Si sabem que $W_2 < W_1$ i que $H_2 < H_1$, el càlcul del ràtio serà el següent:

- Ràtio $_{vertical} = W_2/W_1$
- Ràtio $_{horitzontal} = H_2/H_1$

En el nostre cas obtenim un ràtio de dos. Això vol dir que hem de recórrer la imatge i que de cada dos píxels en sentit vertical i dos en sentit horitzontal ens n'hem de quedar només amb un. Com ja havíem dit abans, aquest sistema ofereix un resultat bastant diferent de la imatge original, ja que en el nostre cas estem descartant tres píxels de cada quatre. Per entendre millor aquest procés es pot veure la figura 5.13.

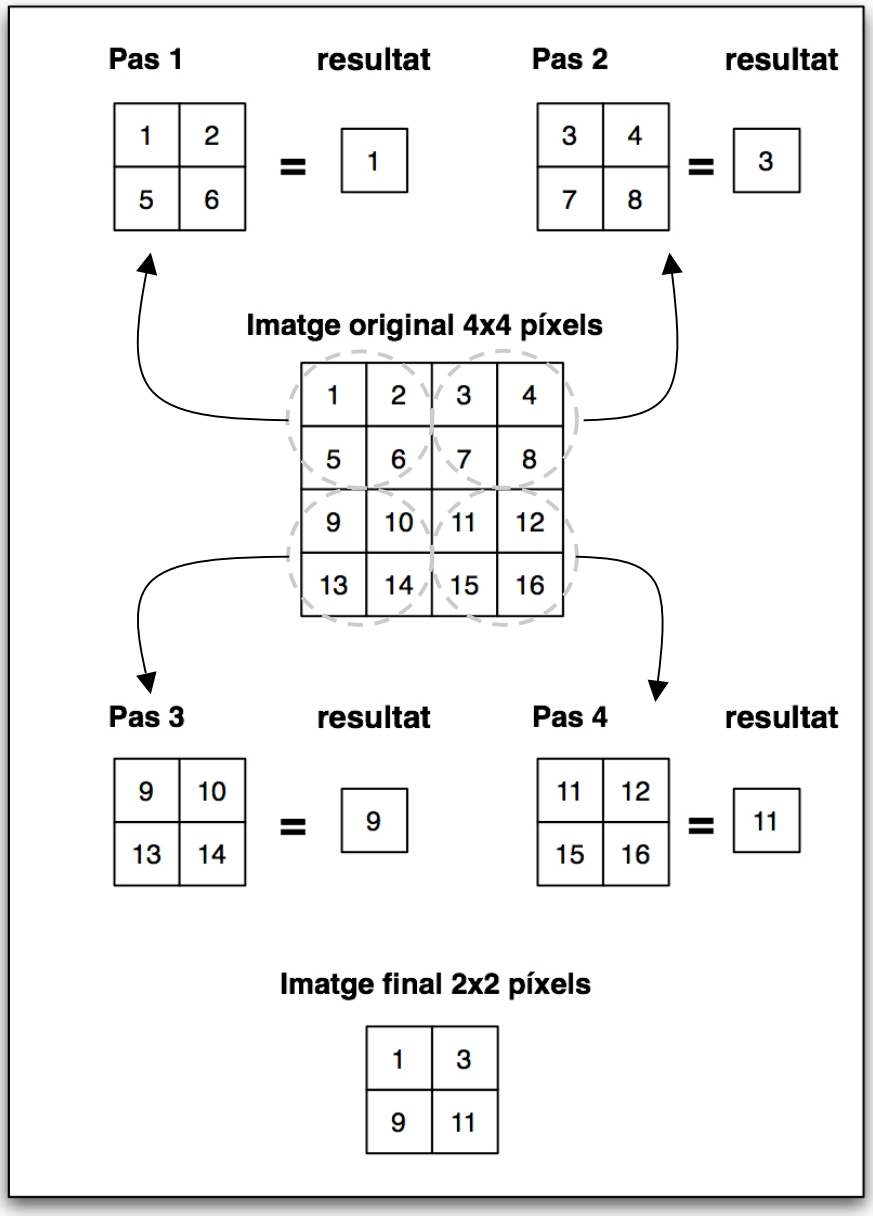


Fig 5.13 Exemple del procés de reducció sense filtrat per una imatge de 4x4 píxels.

En el procés detallat a la figura anterior la 5.13, podem veure que s'ha escollit el sistema de quedar-nos amb el píxel superior esquerra del grup de quatre. És evident que l'elecció del píxel que ens quedem farà que la imatge es vegi d'una manera o una altra. L'elecció del píxel en aquest sistema és completament arbitrari, tant podíem haver escollit aquest com un altre, que el resultat seria força semblant, encara que la imatge es veiés lleugerament diferent.

A la figura 5.14, podem veure el pseudocodi a alt nivell de tot aquest procés de reducció, tal i com l'hem implementat en el nostre projecte. El sistema és el que he esmentat abans, anirem recorrent la

imatge original per files i per columnes, agafant un píxel de cada quatre. La funció *EscullPixel*, es quedarà amb el píxel de la posició actual.

```
files = ALÇADA_DE_LA_IMATGE_EN_PÍXELS
columnes = AMPLADA_DE_LA_IMATGE_EN_PÍXELS
mentre (files > 0 ) fer
  mentre (columnes > 0) fer
     píxel_escollit = EscullPixel(imatge_original, fila, columna)
     generaImatgeReduida(imatge_reduida, píxel_escollit)
     columnes = columnes – 2
  fmentre
  files = files – 2
fmentre
```

Fig 5.14 Pseudocodi de la reducció d'imatge sense filtrat

5.4.2 Procés de reducció amb matriu de pesos

El procés de reducció, mitjançant una matriu de pesos no és molt complex, i ofereix uns resultats bastant bons. Primer de tot, hem d'aclarir un aspecte referent al color, que serà necessari per entendre tot el que vé a continuació. Com ja hem dit al principi de l'apartat, els píxels són d'un únic color, que està representat per un valor numèric. És per això que es poden realitzar els càlculs matemàtics dels que parlarem en aquest apartat.

El procés de reducció amb matriu de pesos té l'inconvenient que pot arribar a ser molt costós. La mida de la matriu ens indica el número de càlculs per píxel que s'hauran de fer. Per tant, com més gran sigui la matriu més càlculs s'hauran de fer. El sistema és el següent: és crea una matriu de pesos de N x M, que farà de màscara, on a cada casella hi trobarem un valor numèric (pes). Aquest pes indica la importància relativa del píxel que ocupa la posició de la matriu. Es tracta d'aplicar la matriu sobre la imatge original per calcular els píxels de la imatge reduïda, a partir de la mitjana ponderada obtinguda d'aplicar la màscara a la imatge original. Un cop hem aplicat la màscara hem de desplaçar la matriu X píxels (hem de tenir en compte que per reduir l'*aliasing* al màxim, X = N),

recorrent la fila i de la imatge. Un cop arribat al final de la fila i, hem d'avançar i+Y files (idealment Y = M). En el cas òptim, X = N i Y = M a cada pas de la màscara per sobre de la imatge estem reduint N x M píxels a un.

Els valors dels pesos dins de la matriu poden estar organitzats de vàries maneres (veure figura 5.15):

- Tots els pesos són iguals. Estem donant la mateixa importància a tots els píxels de la màscara. És el més utilitzat quan les matrius tenen unes dimensions petites.
- Els valors dels pesos són diferents en funció de la posició que ocupen dins la màscara. Ens pot interessar donar més pes als píxels centrals, o bé tot el contrari i donar més pes als píxels del voltant. Se sol utilitzar per matrius de dimensions grans, ja que al voler reduir molts píxels de cop, si donem el mateix pes a tots els píxels, ens pot distorsionar molt la imatge.

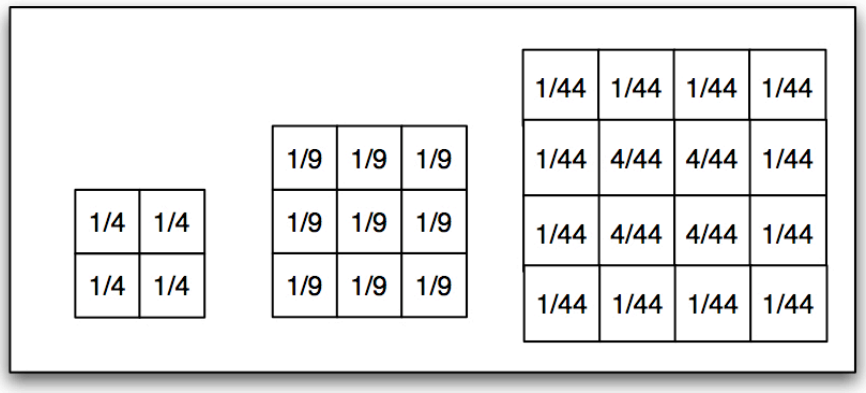


Fig 5.15 Exemple de vàries matrius de pesos diferents.

Per aconseguir bons resultats de reducció, s'ha de trobar el compromís entre la mida de la matriu i l'elecció dels pesos d'aquesta.

Com ja hem dit abans, com més gran és la matriu, més càlculs s'han de fer per obtenir el color de cada píxel de la nova imatge, i això pot arribar a ser molt costós. Quan es tracta de reduir imatges que no canvien, aquest fet pot no importar gaire, sempre i quan l'usuari estigui disposat a esperar.

En el nostre cas ens trobem que l'usuari no pot esperar, ja que les imatges que volem reduir formen part d'un joc, el que significa que *l'ScummVM,* per poder representar el moviment, ha de refrescar (dibuixa) les imatges a la pantalla constantment. Per tant ens trobem que la nostra aplicació no ha de fer una reducció i prou, n'ha de fer moltes i les ha de fer contínuament. La freqüència de refresc

de la imatge a la pantalla (interval de temps que passa des de què es dibuixa una imatge fins que es dibuixa la següent), la marca l'*ScummVM*. Aquesta freqüència, juntament amb la velocitat de càlcul del processador i la mida de la imatge, ens limitarà la mida de la màscara. Hem de vigilar amb la mida de la màscara. Si n'apliquem una de massa gran, el processador tardarà molt a fer els càlculs de la reducció i perdrem la sensació de moviment dels jocs.

Com ja hem dit anteriorment, nosaltres vàrem desenvolupar tota aquesta part, primer per PC i després la vàrem passar a la PDA, però l'elecció de la màscara es va fer en previsió de les limitacions de càlcul del processador de la PDA. És a causa d'aquest aspecte que hem escollit només, dos tipus de reducció per pesos (veure figura 5.15):

- Amb una matriu de 2x2 on tots els píxels tenen el mateix pes.
- Amb una matriu de 3x3 on els píxels centrals tenen més importància que la resta.

El sistema és el mateix que hem explicat al principi d'aquest apartat: les matrius de pesos es desplacen per damunt de la imatge original calculant el color de cada píxel de la imatge reduïda. Els algorismes per recórrer la imatge i aplicar la màscara són pràcticament idèntics, només variarà el nombre de posicions que ha d'avançar la màscara per tornar a ser aplicada, però de totes maneres hem posat l'algorisme a la figura 5.16, per poder entendre tot el procés. Només s'ha de canviar el valor de les variables *pas_fila* i *pas_columna* i substituir-lo per les dimensions N x M de la matriu que vulguem aplicar. En el nostre cas 2x2 i 3x3.

```
fila = 0
columna = 0
pas_fila = N
pas_columna = M
fila_final = 0
columna_final = 0
mentre (fila < ALÇADA_DE_LA_IMATGE_EN_PÍXELS ) fer
  mentre (columna < AMPLADA_DE_LA_IMATGE_EN_PÍXELS ) fer
     píxel_escollit = AplicarMascara (mascara, imatge_original,
```

```
                                        fila, columna, N, M)
    imatge_reduida  = generaImatgeReduida(imatge_reduida,
                                          píxel_escollit,
                                          fila_final,
                                          columna_columna_final)
    columna = columna + pas_columna
    columna_final = columna_final + 1
  fmentre
  fila = fila + pas_fila
  fila_final = fila_final + 1
fmentre
```

Fig 5.16 Pseudocodi de la reducció d'imatge amb mètode de filtrat, utilitzant una màscara de NxM.

Com podem veure l'algorisme per recórrer la imatge és força senzill. A les figures 5.17 i 5.18 podem veure més a fons com es faria el càlcul del píxel escollit. Hem posat el pseudocodi de la funció *AplicarMascara* genèrica, on li passem la màscara, la imatge original, la fila i la columna on tenim posicionada la màscara i la dimensió de la màscara (N files i M columnes). El càlcul és simple, es tracta de multiplicar el color del píxel de cada posició de la imatge pel valor del pes que trobem a la màscara en aquella mateixa posició. El resultat d'aquesta operació el sumarem amb el resultat dels càlculs per les posicions successives. Finalment realitzarem la mitjana aritmètica d'aquest valor dividint per la suma dels pesos de la màscara.

```
AplicarMascara (mascara, imatge_original, fila, columna, N, M)
F = fila
C = columna
per i = 0 fins a N fer
  per j = 0 fins a M fer
    píxel += (imatge_original[f][c] * mascara[i][j])
```

```
    total = total + mascara[i][j]
    c = c + 1
  fi per
f = f +1
fi per
píxel = píxel / total
retorna píxel
```

Fig 5.17 Pseudocodi del càlcul del color del píxel resultant mitjançant l'aplicació d'una màscara de dimensions N x M

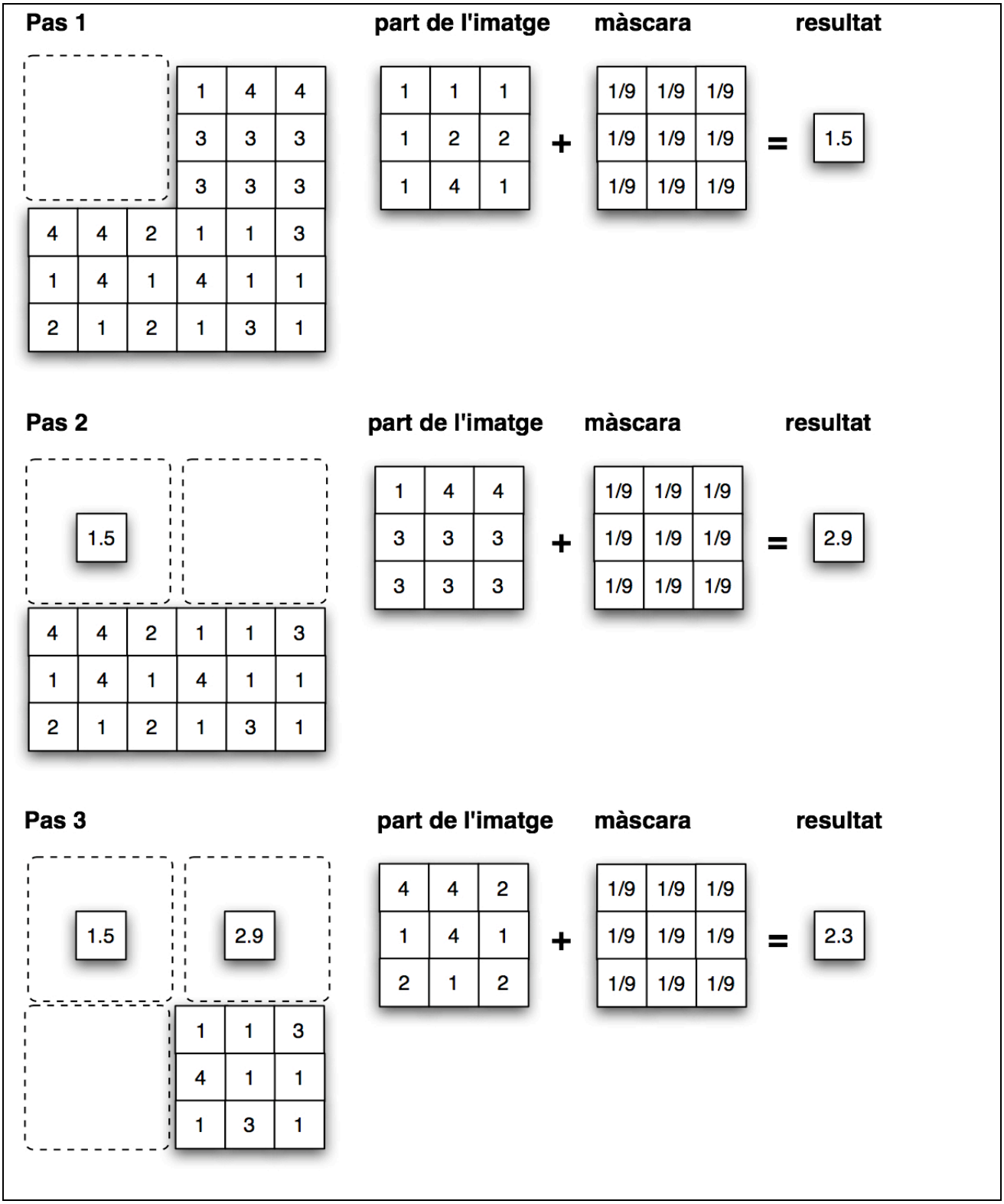
A la figura 5.18, hem intentat il·lustrar l'execució de l'algorisme de reducció. Hem representat una imatge com una matriu numèrica, on a cada posició de la matriu hi ha un número que ens indica el color del píxel que ocupa aquella posició. El mateix hem fet amb la màscara, però en aquest cas en comptes d'un color, a cada posició hi trobem el pes del píxel que ocupa la mateixa posició dins la màscara. A cada pas de la figura mostra el resultat dels càlculs d'aplicar la màscara sobre una zona de la imatge.

Imatge de 6x6 i 4 colors diferents

1	1	1	1	4	4
1	2	2	3	3	3
1	4	1	3	3	3
4	4	2	1	1	3
1	4	1	4	1	1
2	1	2	1	3	1

Màscara de 3x3

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9


Pas 1
part de l'imatge
màscara
resultat
1 4 4
3 3 3
3 3 3
4 4 2 1 1 3
1 4 1 4 1 1
2 1 2 1 3 1
1 1 1
1 2 2
1 4 1
+
1/9 1/9 1/9
1/9 1/9 1/9
1/9 1/9 1/9
=
1.5
Pas 2
part de l'imatge
màscara
resultat
1.5
4 4 2 1 1 3
1 4 1 4 1 1
2 1 2 1 3 1
1 4 4
3 3 3
3 3 3
+
1/9 1/9 1/9
1/9 1/9 1/9
1/9 1/9 1/9
=
2.9
Pas 3
part de l'imatge
màscara
resultat
1.5
2.9
1 1 3
4 1 1
1 3 1
4 4 2
1 4 1
2 1 2
+
1/9 1/9 1/9
1/9 1/9 1/9
1/9 1/9 1/9
=
2.3

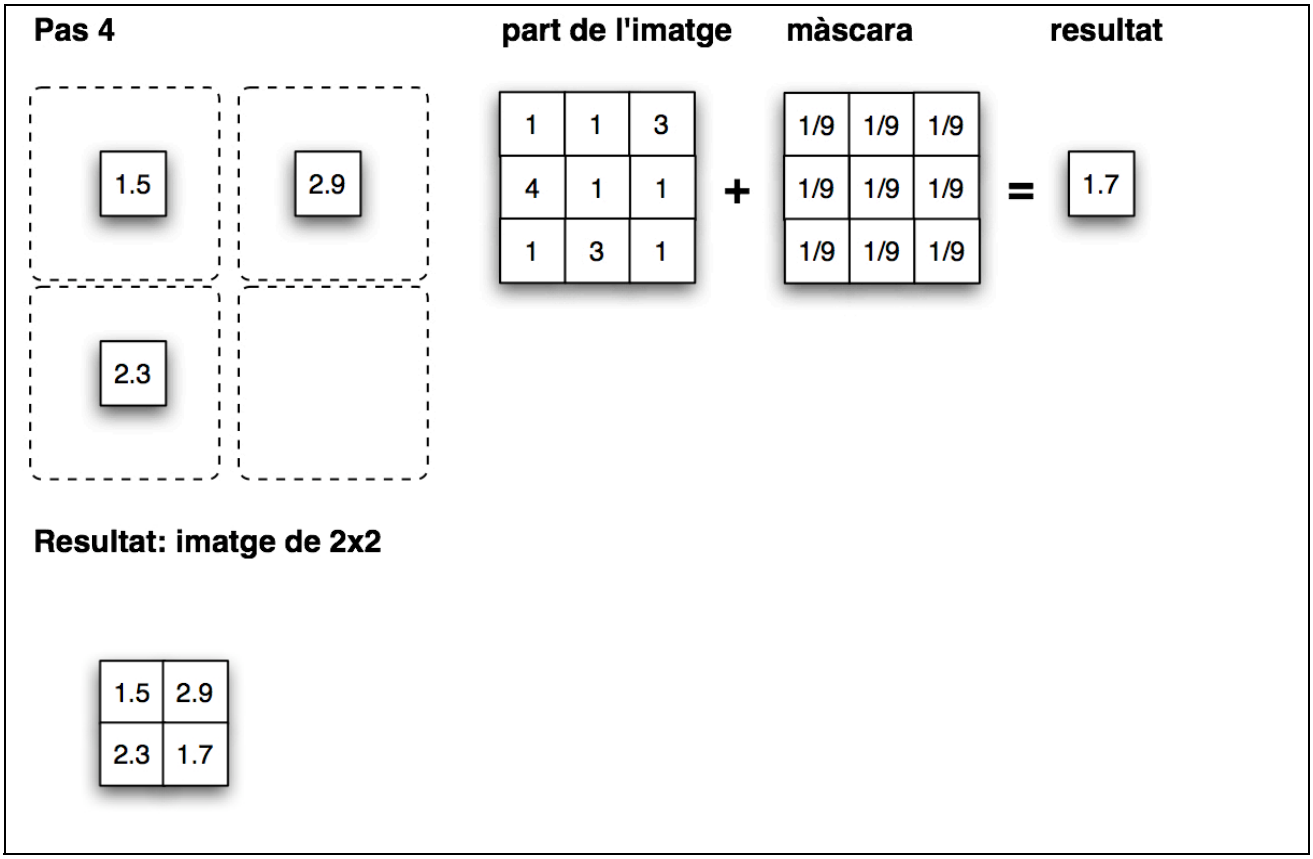


Fig 5.18 Reducció d'una imatge de 6x6 píxels amb 4 colors diferents mitjançant una matriu de pesos de 3x3 on el píxel central té el doble de pes.

Ara que ja hem explicat com funciona la reducció, només queda explicar com estan representats els colors a l'*ScummVM*, i com es realitzen els càlculs per aconseguir el nou color.

5.4.3 Representació del color d'un píxel a l'*ScummVM*

Existeixen moltes maneres de representar un color i moltes taules de colors diferents, les més utilitzades són les següents:

- **RGB (*Red Green Blue*):** representem el color mitjançant les seves components vermell, verd i blau.
- **CMYK (*Cyan Magenta Yellow Key*):** es representa un color mitjançant les seves components de cian, magenta, groc i color clau, que habitualment és el negre.
- **Color indexat:** Per representar el color, s'utilitza un número que fa referència a un color d'una taula de colors.
- Altres.

L'*ScummVM* utilitza el sistema RGB per la versió de PC, i de color indexat per la versió de *PalmOS* amb una taula de colors on els colors estan representats en RGB. Per representar aquest color en format RGB, l'*ScummVM*, utilitza un enter sense signe de 16 bits i per tant, cada color ocuparà 2 bytes a la memòria. Com podem veure a la figura 5.19, aquest número es descomposa en:

- 5 bits per representar el vermell.
- 6 bits per representar el verd.
- 5 bits per representar el blau.

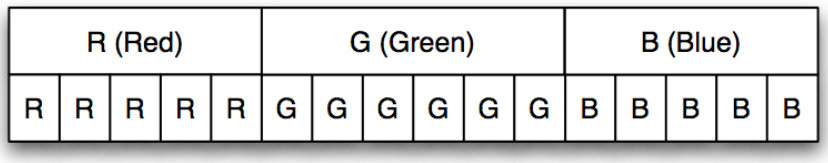


Fig 5.19 Representació del color a l'ScummVM. Enter sense signe de 16 bits

El fet de no tenir els colors separats per les seves components dificulta els càlculs que s'han de fer per poder reduir la imatge. El problema és el següent: per reduir la imatge, hem de sumar i dividir els colors. Aquest càlcul, el fem sumant i dividint el valor numèric que utilitzem per representar-los. Però com ens podem imaginar, si sumem i dividim els colors en el format de l'*ScummVM*, se'ns poden barrejar les components dels colors, obtenint resultats bastant estranys i indesitjables.

La manera correcta de fer aquest càlcul és sumant i dividint les components dels colors per separat, i un cop tenim el resultat les hem de tornar a ajuntar per formar l'enter de 16 bits.

Per fer-ho utilitzarem tres màscares, una per cada component. El sistema és basa en realitzar l'operació lògica AND amb el color RGB i cadascuna de les màscares, per obtenir tres enters de 16 bits on cada enter, representa cadascuna de les components del color original.

Les màscares són les següents (veure figura 5.20):

- Red: 0xf800.
- Green: 0x07e0
- Blue: 0x001f

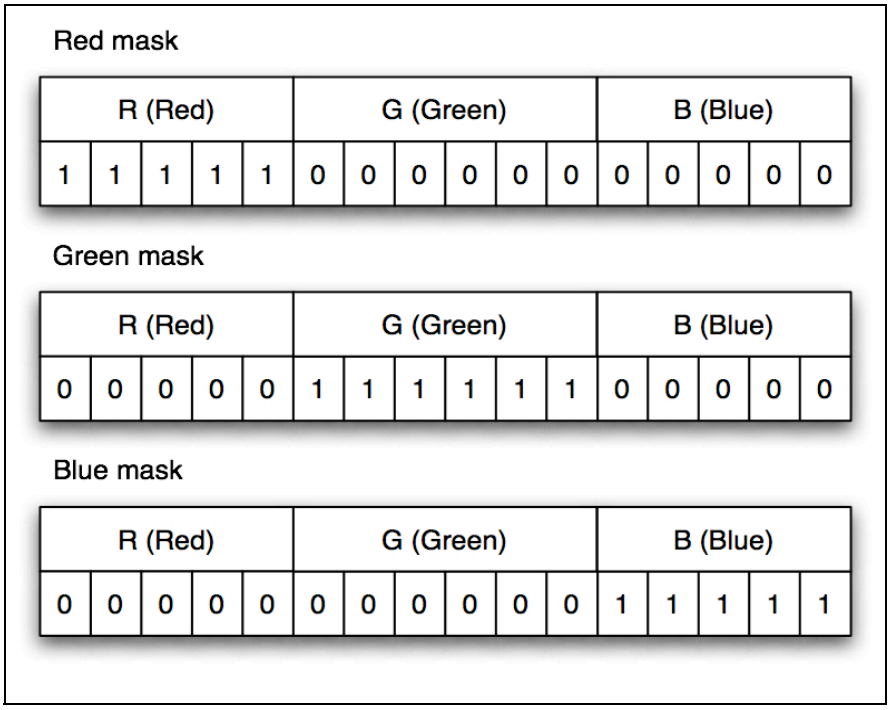


Fig 5.20 Màscares per aïllar les components del color.

Un cop separades les tres components hem de desplaçar els bits de cada component X posicions cap a la dreta, per tenir-los tots a la part baixa de l'enter de 16 bits, per així poder fer les operacions de suma i divisió amb les components separades. Aquests desplaçaments serien:

- Red >> 11.
- Green >> 5.
- Blue.

Notem que la component de blau no cal desplaçar-la, perquè ja es troba en la posició desitjada.

Ara ja podem operar amb les components per separat, sense haver de patir que se'ns barregin els colors per culpa dels *carry*'s, que apareixen en les operacions de suma.

Un cop haguem realitzat les operacions pertinents, haurem de desfer el procés per tornar a tenir totes les components en un sol número. Primer de tot farem els desplaçaments a la inversa, tornarem a aplicar les màscares i sumarem les tres components.

5.4.4 Com es realitzen els càlculs

Per entendre millor el procés explicat al subapartat anterior, a continuació posem un exemple pràctic. Realitzarem la suma dos colors per obtenir-ne un tercer. Color C = Color A + Color B.

A la taula següent podem observar els colors inicials i el resultat de la suma.

	Hexadecimal RGB	**Representació a l'*ScummVM***	**Color**
A	FF3333	0111100001100011	
B	3366FF	0001100011001111	
C = A+B	339933	0001100100100011	

Hem descompost el procés en 5 passos:

- **Pas 1 – Separar les components:** primer de tot cal recordar que l'*ScummVM* emmagatzema el color en d'un píxel en un número binari de 16 bits. El que s'ha de fer és aplicar les màscares que hem explicat anteriorment per separar les tres components de cada color (Red, Green, Blue). Al fer-ho obtenim tres números binaris de 16 bits per cada color amb les components aïllades.
- **Pas 2 – Desplaçar les components:** un cop aïllades les components, les desplacem totes a la part baixa del número de 16 bits per poder operar sense barrejar les components de cada color. Per fer-ho realitzem una operació de desplaçament de bits, explicat a l'apartat anterior.
- **Pas 3 – Suma de components:** tot i que l'*ScummVM* reserva 5 bits per la component Red, 6 per la component Green i 5 per la component Blue, en realitat només utilitza 4 bits per cadascuna. El que fem és convertir cada component a un número de 4 bits i llavors procedim a fer la suma de cada component. El resultat són 3 números de 16 bits, un per cada component. Si al sumar es produeix desbordament per *carry*, agafem aquest bit de *carry* i el sumem al número.
- **Pas 4 – Desplaçar les components:** un cop feta la suma, convertim cada component a un número de 16 bits una altra vegada i desplacem cada component a la zona del número que els hi pertoca.

- **Pas 5 – Sumar les components per obtenir el nou color:** l'últim pas és fer la suma dels 3 números de 16 bits que representen les components.

A continuació posem un exemple amb figures comentant cada un dels passos anteriors.

Pas 1: En el primer pas (veure figura 5.21), apliquem les màscares (R,G i B) a cadascun dels dos colors (A i B). Un cop aplicades les màscares obtenim les components (RGB) de cada color aïllades de la resta. Cada component és un color nou.

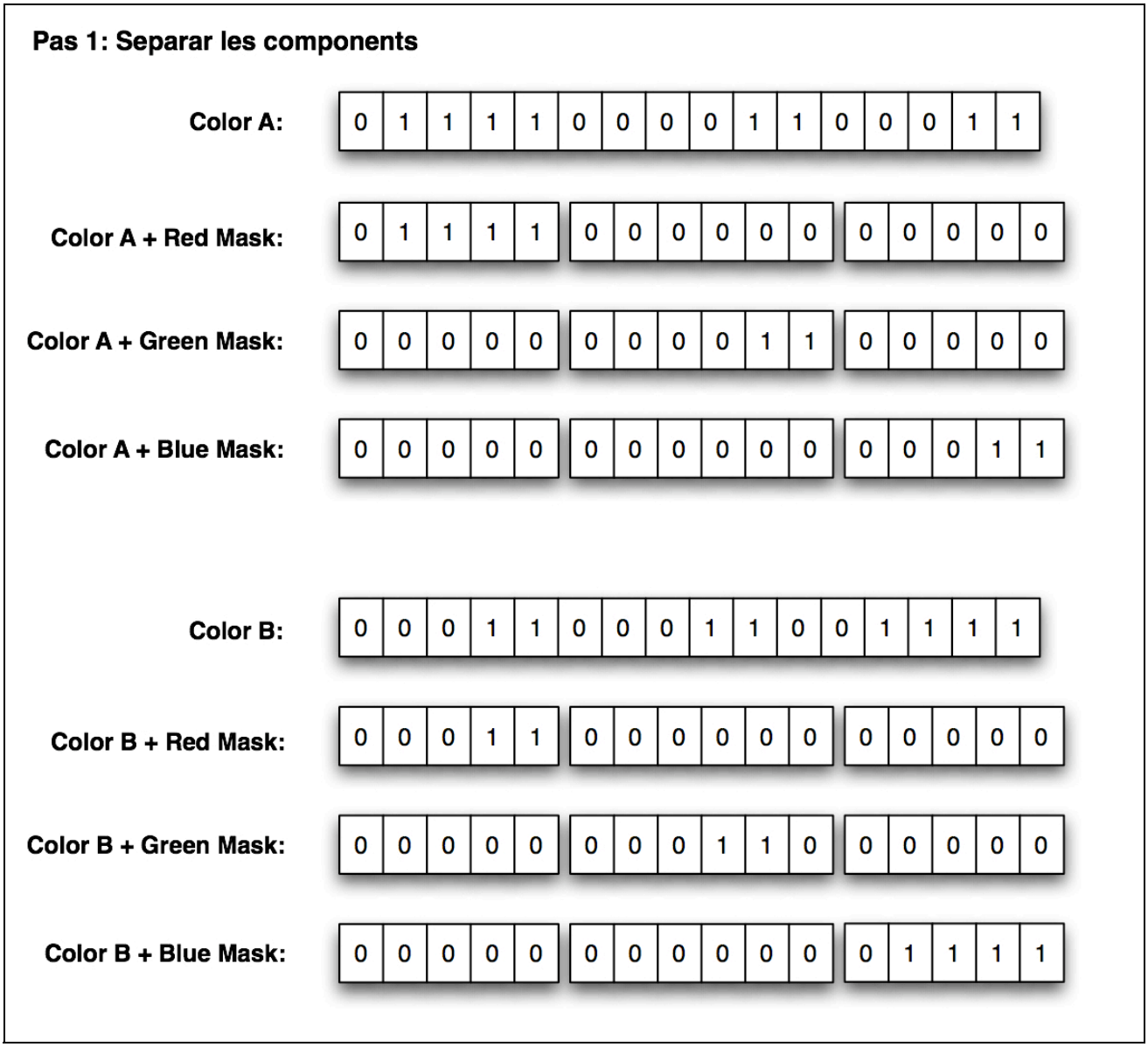


Fig 5.21 Pas 1: separar les components de cada color.

Pas 2: El pas dos (veure figura 5.22) consisteix en desplaçar el valor de cada component a la part baixa del número de cada número de16 bits. Apliquem desplaçaments diferents, en funció de la component com ja hem explicat anteriorment. Això ho fem per a què ens sigui més fàcil controlar el desbordament en l'operació de suma de components.

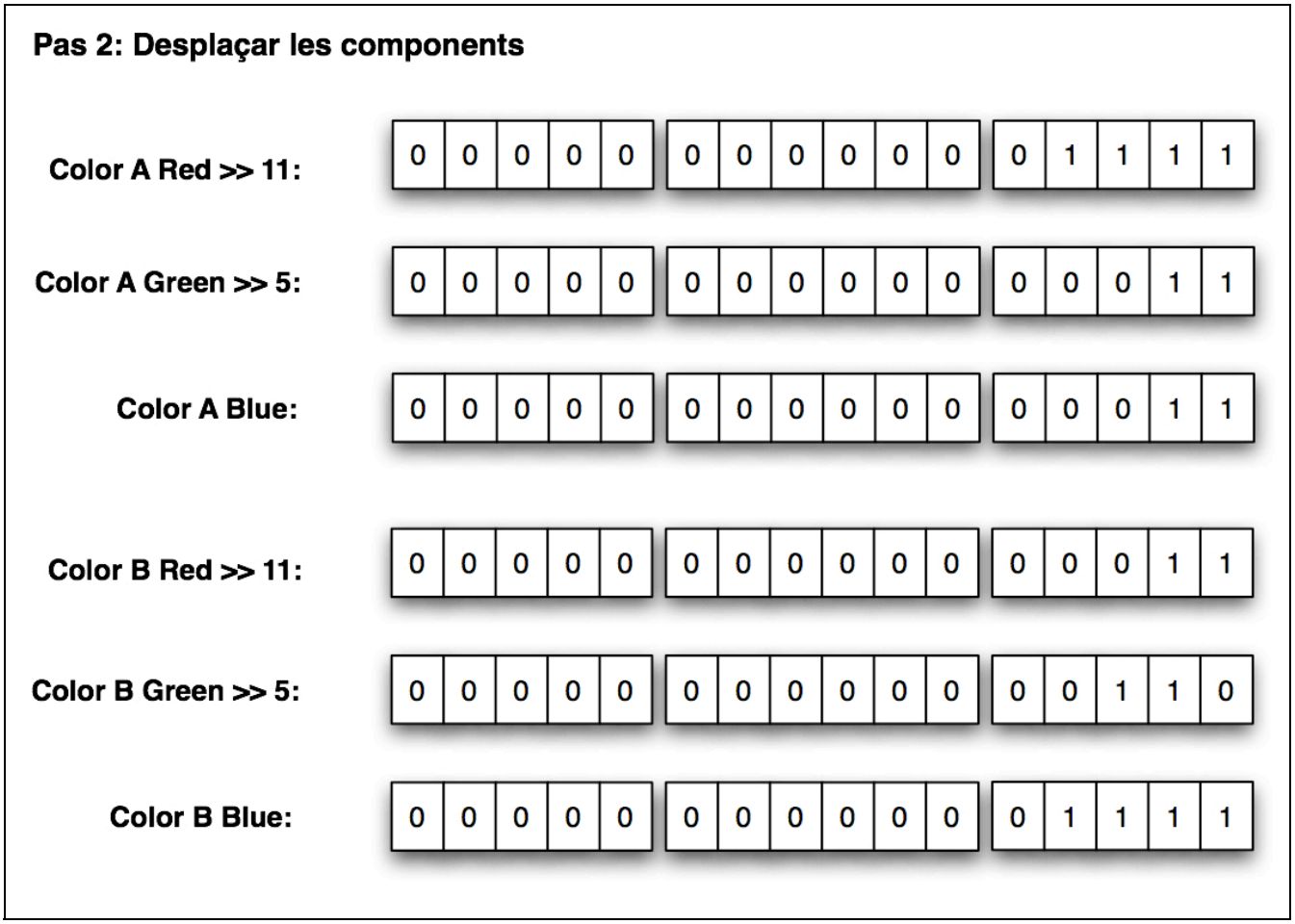


Fig 5.22 Pas 2: Desplaçar els bits de les components a la part baixa del número de 16 bits.

Pas 3: En el tercer pas del procediment (veure figura 5.23), realitzem la suma de cada component de cada color per obtenir el valor de la nova component. Si ens fixem en la figura veiem que es controla el *carry* per a què la mida de la nova component passi de 4 bits.

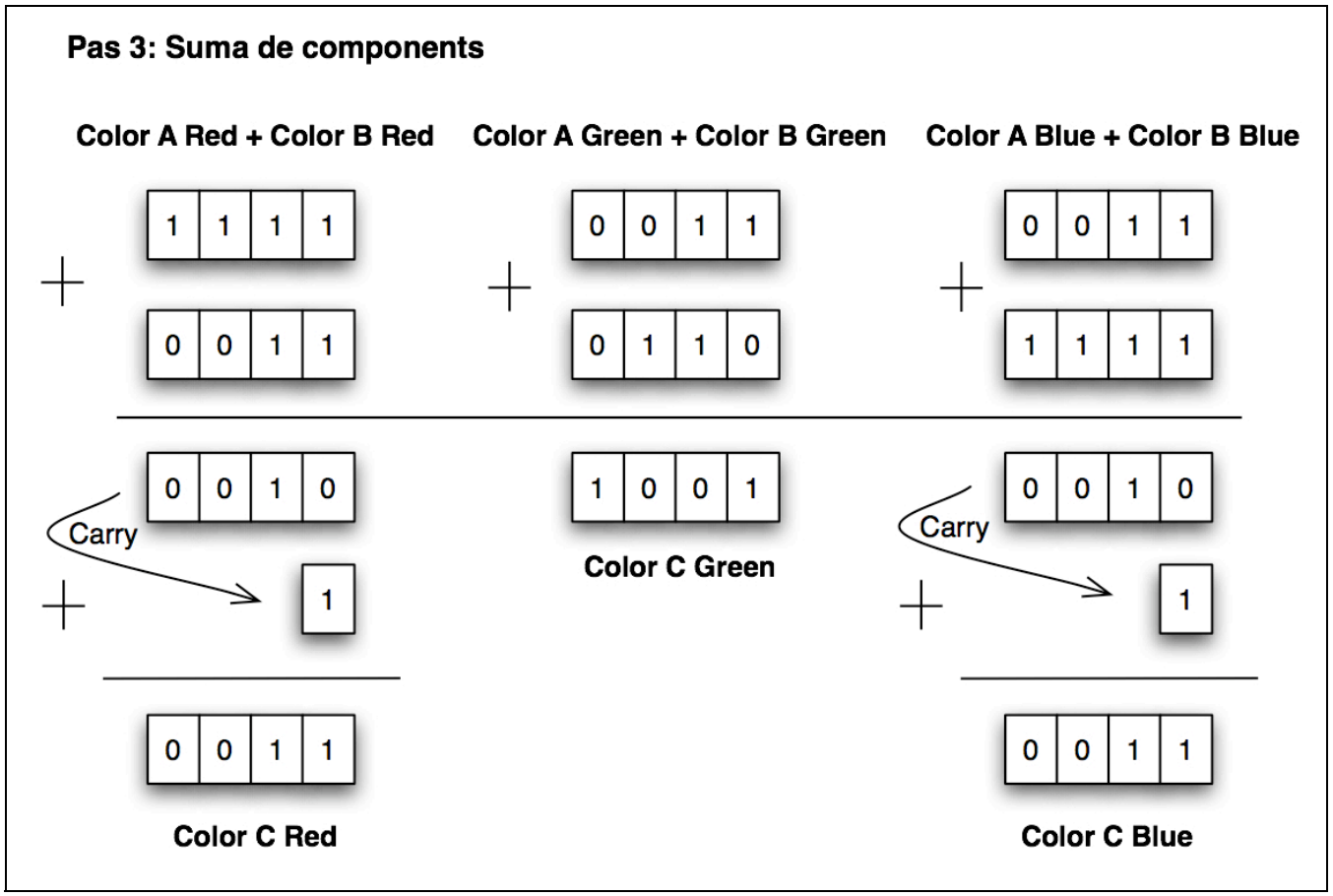


Fig 5.23 Pas 3: Suma de les components de cada color per separat.

Pas 4: En el pas quatre, desplacem la component de cada color a la part del nou número de 16 bits que li pertoca a cadascuna. Utilitzem desplaçaments diferents per cada tipus de component (veure figura 5.24).

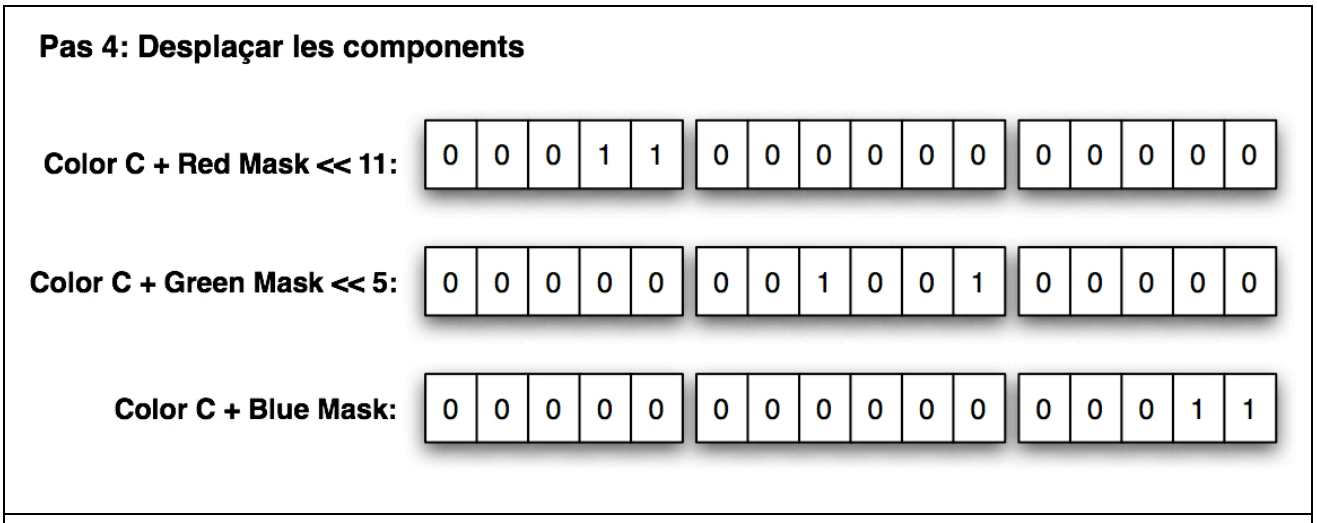


Fig 5.24 Desplaçament de les compoments cap a la seva zona del número que representa un color.

Pas 5: En el cinquè pas sumem els tres colors que representen cada component del color nou, per obtenir aquest color nou en el format de l'*ScummVM*. Però abans de sumar apliquem les màscares un altre cop sobre cada component per eliminar els possibles valors residuals resultat de la suma (veure figura 5.25).

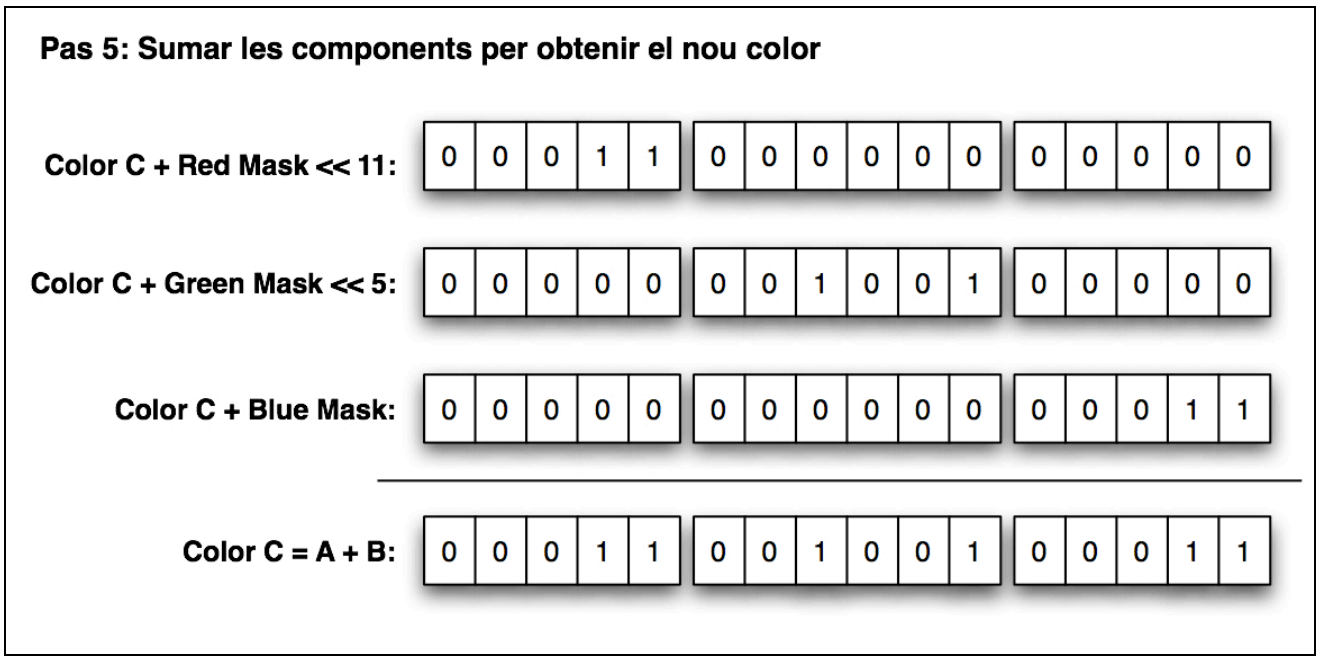


Fig 5.25 Suma de les components del color nou per obtenir el color en format ScummVM.

5.5 Controlar l'entrada/sortida en PC

Per poder treballar millor quan es passi a treballar amb el *PalmOS*, es va decidir que el millor procediment era fer el següent:

- Controlar el text dels jocs.
- Controlar els events de ratolí i teclat.

5.5.1 Controlar el text dels jocs

Pel que fa al text dels jocs, el que hem fet ha estat omplir unes estructures de dades pròpies per cada tipus de text. La manera d'omplir les nostres estructures de dades ha estat la menys invasiva possible, per fer-ho hem introduït crides als nostres mètodes dins del codi de l'*ScummVM*. Aquestes crides s'han posat dins dels mètodes descrits en l'apartat 5.3.2.2. D'aquesta manera, aconseguim tenir el text sota control i el podrem mostrar utilitzant les tipografies que cada sistema operatiu ens proporciona. Vam decidir que el millor era no eliminar el text real de les imatges, de manera que l'usuari veiés el text per la pantalla encara que aquest resultés il·legible per culpa de la reducció. Ho hem fet així perquè, d'aquesta manera no canviem el funcionament real de l'*ScummVM*, ni dels jocs, obtenint així una jugabilitat millor i més interactiva.

Mentre treballem sobre PC, el text el mostrem per la consola de sistema. No cal inventar una manera de mostrar-ho a la pantalla. El que ens interessa de moment, és tenir-lo emmagatzemat, per poder implementar una manera neta de mostrar-lo a la Palm. A causa de les similituds de funcionament de cada tipus de text, hem decidit implementar unes estructures de dades per emmagatzemar el text, i uns quants mètodes per tractar-lo. Ho farem per cada tipus de comportament i no per cada tipus de text. D'aquesta manera ens trobem que tenim la classificació següent:

- Text no interactiu.
    - Text de les converses entre personatges, o del narrador.
    - Text escollit del menú d'acció o de les converses interactives.
- Text interactiu.
    - Text dels menús d'acció que hi ha per cada joc.
    - Text interactiu de les converses.

5.5.1.1 Text no interactiu

Ens trobem que hi ha un tipus de text, que no cal emmagatzemar. Aquest serà tot el text no interactiu. Concretament, el text de les converses entre els personatges o del narrador, i el text que ens indica quina opció del menú d'acció o de les converses ha estat escollida.

El mètode de l'*ScummVM* encarregat de mostrar aquest text per pantalla és el *CHARSET_1(),* i el trobem implementat a dins de la classe *SCUMM* (veure apartat 5.3.2.2). Com ja hem comentat anteriorment, aquest és el mètode que hem utilitzat com a punt d'entrada de la nostra aplicació per recollir aquest tipus de text. Com que l'usuari no hi ha d'interactuar, amb recollir-lo juntament amb el seu color i la seva posició a la pantalla en fem més que suficient (veure figura 5.26). Així doncs, un cop el tenim recollit, el podem mostrar al mateix temps que l'*ScummVM* el mostra transformat en imatge per la pantalla.

```
recullTextNoInteractiu (text, propietats)
    escriu (text, propietats.color, propietats.posX,
                propietats.posY )
fi recullTextNoInteractiu
```

Fig 5.26 Pseudocodi del mètode de mostrar el text no interactiu

A la figura 5.26, podem veure un tall del pseudocodi necessari per fer el que acabem de comentar. El procediment seria el següent: El nostre mètode rep el text amb les seves propietats (color i posició de la pantalla) i de moment, el mostrarem a la consola de sistema, com ja hem comentat abans (veure figura 5.27).

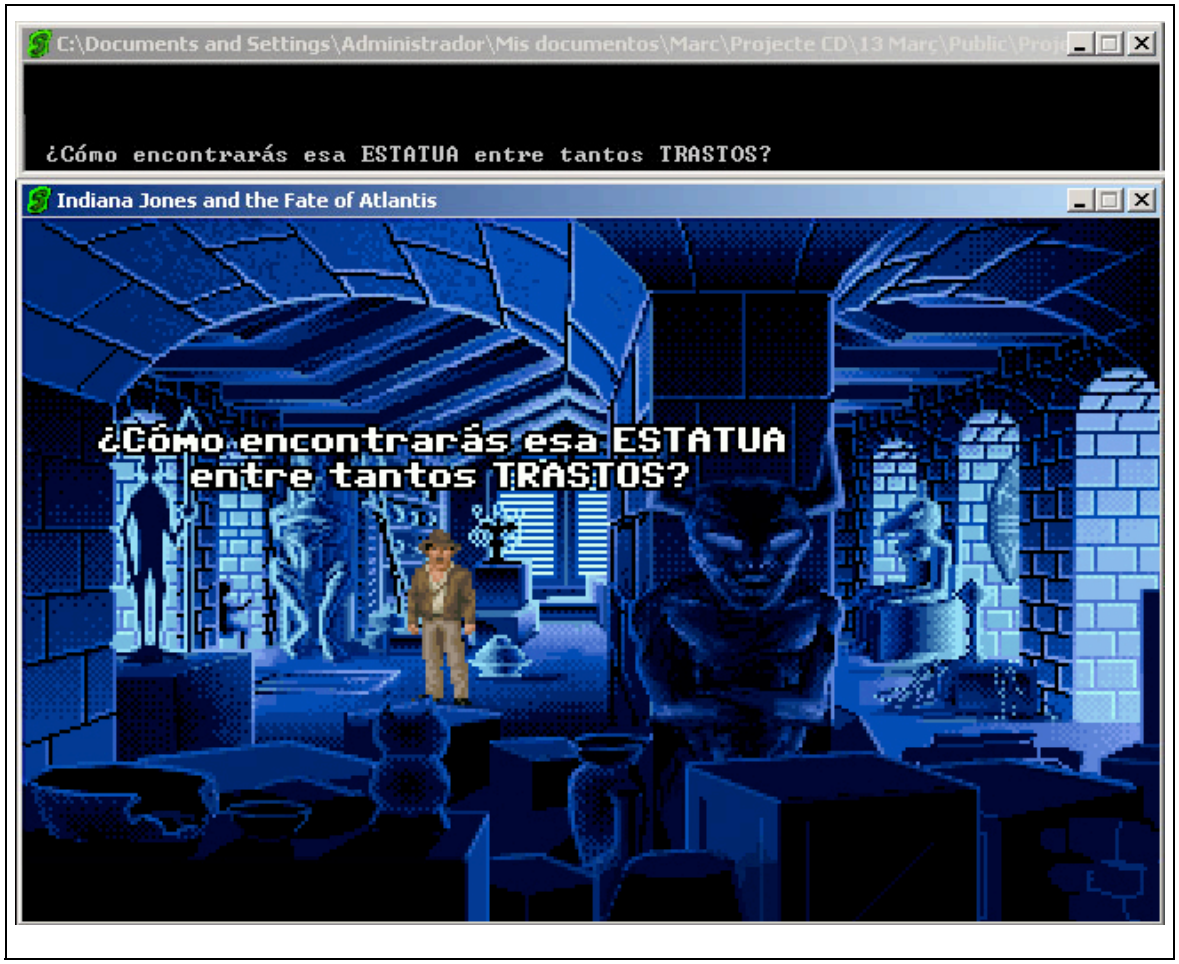


Fig 5.27 Captura de pantalla mostrant el text del jugador a la consola i a la pantalla.

5.5.1.2 Text interactiu

Pel que fa al text interactiu, ens adonem que l'hem d'emmagatzemar, ja que necessitem guardar les seves coordenades reals, per a què, quan l'usuari seleccioni una de les opcions del joc o de les converses interactives des dels menús que mostrem nosaltres, haurem d'enviar a l'*ScummVM* un event de clic amb les coordenades reals (posició x i posició y) de l'opció escollida abans de la reducció. També guardem el color del text per poder-lo mostrar tal i com el veu el jugador.

Per recollir el text de les converses interactives i el text del menú d'accions del joc ens cal un punt d'entrada de la nostra aplicació a dins del codi de l'*ScummVM* que ens proporcioni les dades que necessitem. El nostre punt d'entrada serà el mètode *DrawString()* que està implementat a dins de la classe *SCUMM*. Com ja hem comentat anteriorment (veure apartat 5.3.2.2), el mètode *DrawString()* és l'encarregat de tractar el text abans d'enviar-lo a renderitzar (convertir a imatge) i conté totes les propietats del text que necessitem.

Per emmagatzemar el text utilitzem dues estructures (veure figura 5.28) que es faran servir amb mètodes molt semblants. Una emmagatzemarà el text del menú d'accions del joc i l'altre el de les converses interactives. Encara que les estructures s'assemblin bastant, en necessitem dos perquè el text del menú es manté constant, mentre que les opcions de les converses varien en funció de la conversa. La implementació de les classes on guardem aquestes estructures l'explicarem a l'apartat 5.8 de *PalmOS*, que és on hi ha comentada l'arquitectura del codi que hem utilitzat.

```
classe ActionMenu
  tipus taccio
    enter num
    enter cx
    enter cy
    color colorText
    cadena accio[30]
  fi tipus
  taccio llistaAccions[30]
  ...
fi classe
```

```
classe TextTalk
  tipus tphrase
    enter num
    enter cx
    enter cy
    color colorText
    cadena phrase[100]
  fi tipus
  tphrase llistaPhrases[50]
  ...
fi classe
```

Fig 5.28 Pseudocodi de les estructures de dades dins les classes on guardarem el text de les accions del menú i el text de les converses interactives.

Com podem veure a la figura 5.28, cada estructura està formada per una llista d'elements, on cada element guarda les característiques del text que necessitem. Els atributs que guardem són els següents:

- **num:** és el número que utilitzarà l'usuari per seleccionar una acció o frase de la conversa i que ens l'identifica a dins de la nostra estructura.
- **cx i cy:** són les coordenades de la pantalla on l'*ScummVM* dibuixa l'acció o la frase de la conversa.
- **color:** color del text en el format de l'*ScummVM*, que ja hem explicat anteriorment.

- **cadena:** guarda una cadena de caràcters amb el text de l'acció o de la frase de la conversa interactiva.

Un cop explicades les estructures de dades, anem a explicar el funcionament de la nostra aplicació.

Quan el sistema detecta un clic amb el ratolí a sobre de l'àrea de la pantalla on apareixen les accions i les frases de les converses interactives del joc, la nostra aplicació atura l'execució i fa les comprovacions següents:

- **Si estem en el menú:** en aquest cas mostrarà les opcions del menú a la consola de sistema i s'esperarà fins que l'usuari esculli una opció. Un cop escollida, es reemprèn l'execució del joc i la nostra aplicació busca a dins de l'estructura de dades que emmagatzema les accions, les coordenades de pantalla originals de l'acció, i envia un clic de ratolí al sistema amb aquestes coordenades. D'aquesta manera el sistema creu que l'usuari ha clicat al damunt de l'acció i actua en conseqüència
- **Si estem en una conversa:** en el cas d'estar en una conversa, mostrarà les opcions de resposta de la conversa a la consola de sistema i s'esperarà fins que l'usuari esculli una opció. Un cop escollida, es reemprèn l'execució del joc i la nostra aplicació busca a dins de l'estructura de dades que emmagatzema les frases de les converses interactives les coordenades de pantalla originals de l'acció, i envia un clic de ratolí al sistema amb aquestes coordenades. D'aquesta manera el sistema creu que l'usuari ha clicat al damunt de la frase i actua en conseqüència
- **En qualsevol altre cas:** no farà res.

El motiu de controlar si estem en una conversa o escollint opcions del menú és que vam escollir que les dades estesin emmagatzemades en estructures diferents, encara que el funcionament fos similar. Es va escollir aquest sistema perquè era el més simple, ja que de moment només ens interessava tenir controlat el text, i poder provar que quan l'usuari escollís una opció, s'enviessin les coordenades correctes a l'*ScummVM* per poder tenir tot el funcionament sota control quan treballéssim sobre Palm. Per poder-ho afegir de la manera menys intrusiva possible al codi de *l'ScummVM,* la implementació d'aquest pseudocodi es troba dins del nostre mètode que controla els events (veure apartat 5.5.2).

A la figura 5.29 podem veure un exemple de com seria el pseudocodi per controlar, quan l'usuari fa clic sobre l'àrea de la pantalla, on es mostren les opcions del menú d'accions del joc i les frases del

text interactiu. Els mètodes *coordenades_accio* i *coordenades_text* utilitzats a la figura, són els encarregats de recórrer les estructures de dades, on guardem les accions i el text interactiu de les converses del joc, per buscar l'opció que correspon al número introduit per l'usuari. Un cop troba l'acció o la frase de la conversa, retorna les coordenades de la pantalla on aquesta està dibuixada. Aquests mètodes els trobem implementats a les classes encarregades d'emmagatzemar cadascun dels tipus de text (TextTalk i ActionMenu).

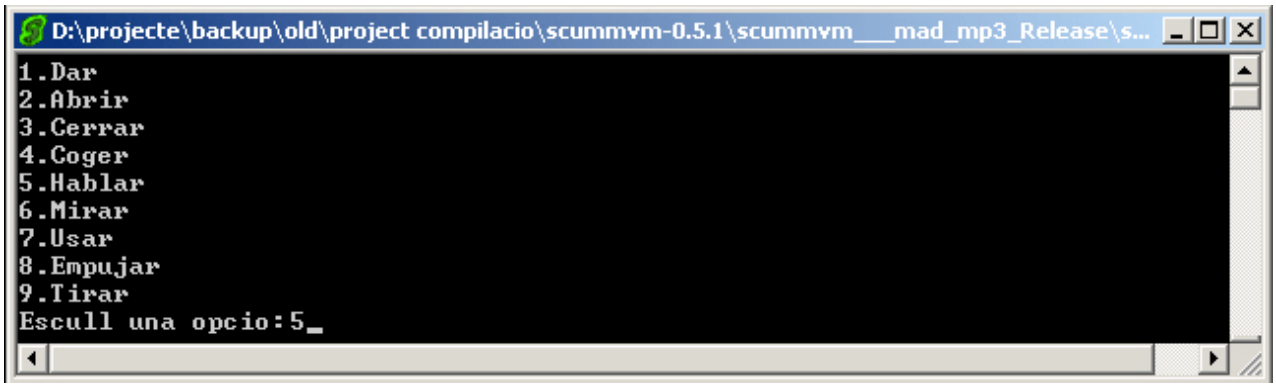
```
si usuari_fa_clic_sobre_area_interactiva fer
  si (estem_en_el_menu) fer
    dades_opcions = mostraOpcionsMenu(text_menu)
    mentre (no_esculli_una_opcio) fer
       opcio = demanaOpcio()
     fi mentre
     (coord_x, coord_y) = coordenades_accio(opcio, dades_opcions);
     enviaEventAlScummVM('event_clic', coord_x, coord_y)
  altre si (estem en una conversa) fer
    dades_ opcions = mostraOpcionsTextInteractiu(text_converses)
    mentre (no_esculli_una_opcio) fer
       opcio = demanaOpcio()
     fi mentre
     (coord_x, coord_y) = coordenades_text(opcio, dades_opcions);
     enviaEventAlScummVM('event_clic', coord_x, coord_y)
   altrament
   no_fer_res
  fi si
fi si
```

Fig 5.29 Pseudocodi del mètode que s'executa quan l'usuari fa clic sobre la pantalla d'accions

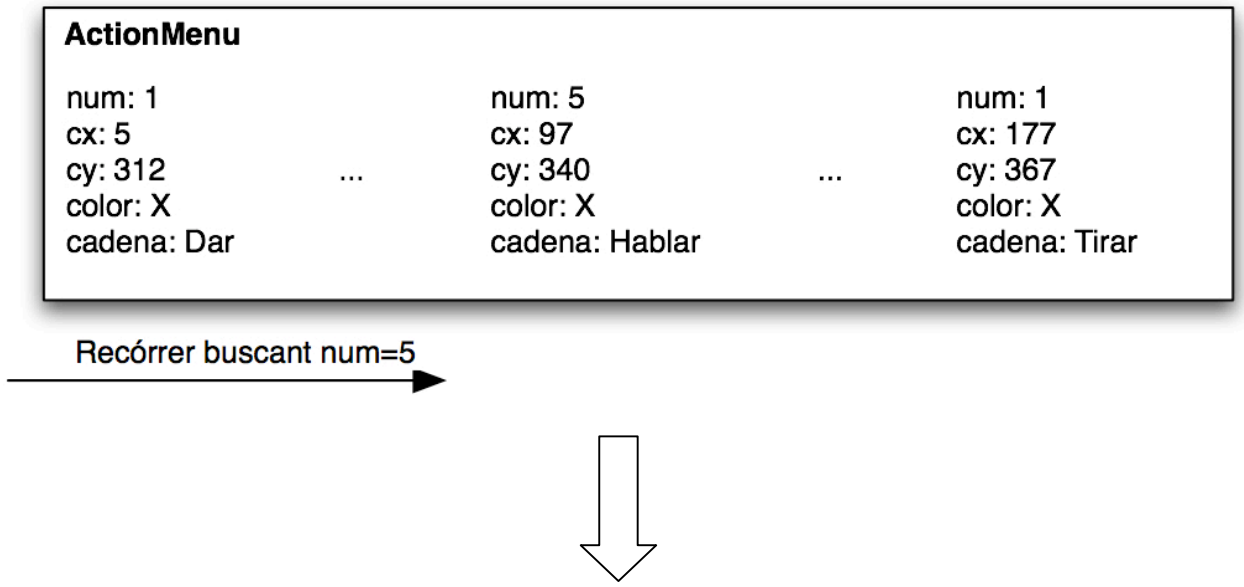
Per entendre millor el procés descrit a l'apartat, a la figura 5.30 podem veure un exemple gràfic del cas de selecció d'una acció del menú d'accions.

1 .- L'usuari fa clic a l'àrea d'accions.

2.- S'obre la consola amb les opcions del menú i s'atura l'execució a l'espera que l'usuari en triï una.



3.- L'usuari escull l'acció *5.Hablar*. Busquem a l'estructura de dades les coordenades de l'acció escollida.



4.- Generem un clic a les coordenades de l'acció *Hablar*.

Fig 5.30 Captura de pantalla mostrant el text del jugador a la consola i a la pantalla.

5.5.2 Controlar els events de ratolí i teclat

Pel que fa als events de teclat, en la versió per PC ens limitem a saber on es tracten i quines funcionalitats tenen assignades les tecles del teclat. Això ho fem així perquè, quan treballem a la Palm, el que voldrem serà convertir els events que generen els botons de la Palm en events de teclat que l'*ScummVM* entengui. No els convertirem tots, només els essencials per poder jugar sense el llapis de la Palm.

Els events de teclat que volem tenir controlats són dos:

- Tots els de les fletxes de direcció. Fletxa esquerra, dreta, amunt i avall. Ens serviran per moure el punter per la pantalla.
- Tecla enter o intro. Que ens servirà per fer el clic del mouse.

Per altra banda tenim els events de ratolí. Quan es genera un event de ratolí, l'*ScummVM*, rep dues coses, el tipus d'event (botó dret premut, moviment del mouse,…) i les coordenades de la pantalla, on s'ha produït aquest event. A causa de la reducció d'imatge que hem implementat (veure apartat 5.4), les coordenades de la pantalla que retornen els events de clic no són les correctes (són la meitat). Hem de solucionar aquesta problemàtica, i ja ho podem fer sobre PC, perquè quan treballem sobre Palm haurem de fer el mateix.

Un cop identificats els events que volem tractar hem de decidir com ho fem. A l'apartat 5.5.3 hem explicat quin és el mètode encarregat de tractar els events a dins de l'*ScummVM*, de manera que nosaltres capturarem l'event dins d'aquest mètode i l'enviarem a un mètode nostre per tractar-lo i retornar-lo, perquè l'*ScummVM* l'interpreti com nosaltres volem. D'aquesta manera aconseguim el nostre objectiu de fer el nostre codi el menys intrusiu possible.

El control dels events seria el següent (veure figura 5.31), primer de tot hem d'identificar l'event:

- **Si és un event de ratolí:** en aquest cas hem de comprovar si és un event de clic amb el botó esquerre o de desplaçament del ratolí. Si és un dels dos, agafem les coordenades del clic o del desplaçament que ens proporciona l'event mateix. A causa de la reducció d'imatge, aquestes coordenades no coincideixen amb les coordenades de la imatge original, és per això que hem de multiplicar per dos cada coordenada (X,Y).
- **Si és un event de teclat:** en aquest cas descartarem tots els tipus d'event de teclat excepte els de la tecla intro premuda i els de les tecles de les fletxes.

- **Tecla intro/enter:** enviarem a l'*ScummVM* un event de clic amb el boto esquerra del mouse en les coordenades on estigui col·locat el mouse en el moment de l'event. Un cop fet això multiplicarem les coordenades per dos, a causa de la reducció d'imatge.
- **Tecles de direcció/fletxes:** enviarem a l'*ScummVM* un event de desplaçament de ratolí, incrementant o decrementant una posició a les coordenades (X,Y) en la direcció que indiqui la tecla premuda a partir d'on estigui col·locat el mouse en el moment de l'event. Un cop fet això multiplicarem les coordenades per dos, a causa de la reducció d'imatge.

- **Qualsevol altre event:** si ens arriba qualsevol altre tipus d'event deixarem que l'*ScummVM* el tracti sense que nosaltres hi intervinguem.

```
ControlEvents(event)
  si ( tipusEvent(event) == ratoli ) fer
    si (tipusEventRatoli(event) == clic_boto_esquerra o bé
        tipusEventRatoli(event)  == desplaçar_ratoli ) fer
      (coord_x, coord_y) = coordenades(event);
      coord_x = 2 * coord_x
      coord_y = 2 * coord_y
    fi si
  altra si ( tipusEvent(event) == teclat)
    si (tipusEventTeclat(event) == tecla_intro ) fer
      event = canviaTipusEvent(event, 'clic_boto_esquerra')
    altra si ( tipusEventTeclat (event) == fletxes)
      event = canviaTipusEvent(event, 'desplaçament')
      event = canviaCoordenadesEvent(event);
    fi si
  altrament
    no fer res
```

```
  fi si

retorna ( coord_x, coord_y, event )
```

Fig 5.31 Pseudocodi del mètode que controla els events de teclat i de ratolí

5.6 Definició de l'aplicació

La nostra aplicació serà una capa de portabilitat sobre l'*ScummVM 0.51*, que serveix per poder utilitzar l'*ScummVM* en dispositius que funcionen amb *PalmOS* i que tenen prestacions reduïdes (veure apartat 3).

La nostra capa de portabilitat implementa:

- La reducció d'imatge a la meitat de la resolució original dels jocs, utilitzant 2 modes de filtrat diferents:
    - Reducció sense filtrat.
    - Reducció utilitzant un mètode de filtrat de matriu de pesos.

  L'elecció dels mètodes ens vé limitada per la velocitat del maquinari. Com ja hem explicat abans, la potència de càlcul de la PDA utilitzada és limitada. És per això, que hem escollit un parell de mètodes, a mode d'exemple de la reducció, que no alenteixin el funcionament del joc.
- Soluciona els problemes que genera aquesta reducció d'imatge:
    - Control i gestió del text que surt pantalla.
    - Control i gestió dels clics.

5.6.1 Funcionament de l'aplicació

Un cop hem executat l'aplicació, escollim el joc de la llista de jocs disponibles i l'executem. Pel que fa a la imatge, disposem de dos modes:

- **Mode reduït 2:1 sense filtrat:** és el mode de funcionament per defecte. Aquest mode mostra el joc a la meitat de la seva resolució original, és a dir de 320x200 a 160x100 píxels, sense utilitzar cap mètode de filtrat.

- **Mode reduït 2:1 amb filtrat 2x2:** aquest mode de funcionament mostra el joc utilitzant el mètode de reducció amb un filtre que utilitza una matriu de pesos de 2x2 on tots els elements de la matriu tenen el mateix pes. El rati de reducció és el mateix que en el mode reduït sense filtrat. Es passa de la resolució original de 320x200 a la meitat (160x100).

A la figura 5.32 podem veure un exemple dels dos modes d'imatge.

Fig 5.32 Captures de pantalla dels 2 modes d'imatge. La primera del mode reduït 2:1 sense filtrat i la segona del mode reduït 2:1 amb filtrat 2x2.

Aquests modes de funcionament és poden intercanviar en qualsevol moment del joc i són independents dels modes de text.

Pel que fa al text tenim els dos modes de funcionament següents:

- **Mode Text:** En el mode text, mostrem tot el text que surt per la pantalla utilitzant la tipografia del *PalmOS*. Aquest és el mode de funcionament per defecte.
- **Mode Sense Text:** en aquest mode, es mostra el joc tal i com és en realitat, però utilitzant la reducció d'imatge.

Igual que amb els modes d'imatge, els modes de text és poden intercanviar en qualsevol moment de l'execució del joc.

A la figura 5.33 podem veure un exemple dels varis modes de funcionament del text.

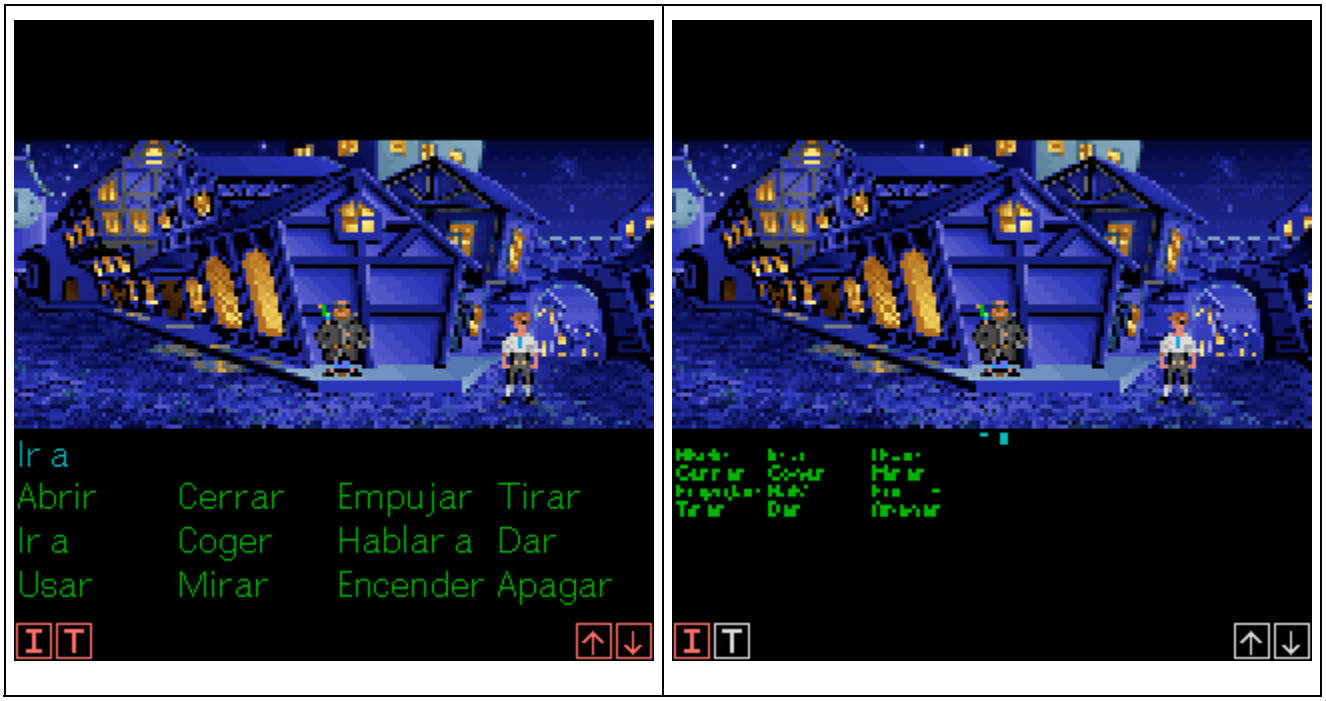


Fig 5.33 Captures de pantalla dels dos modes de funcionament del text. La primera amb el mode text activat i la segona amb el mode text desactivat.

5.8 Arquitectura de la capa de portabilitat

Per resoldre la problemàtica anunciada, hem decidit implementar les classes que explicarem a continuació. La reducció d'imatge no s'ha implementat com una classe, sinó que s'ha implementat com una funcionalitat de la nostra capa de portabilitat.

De manera que tenim les classes següents (veure figura 5.34):

- Classe ActionMenu.
- Classe TextTalk.
- Classe ClicksTreatment.
- Classe PlayerActions.
- Classe TextOutput.
- Classe Color.
- Funcionalitat de reducció d'imatge.

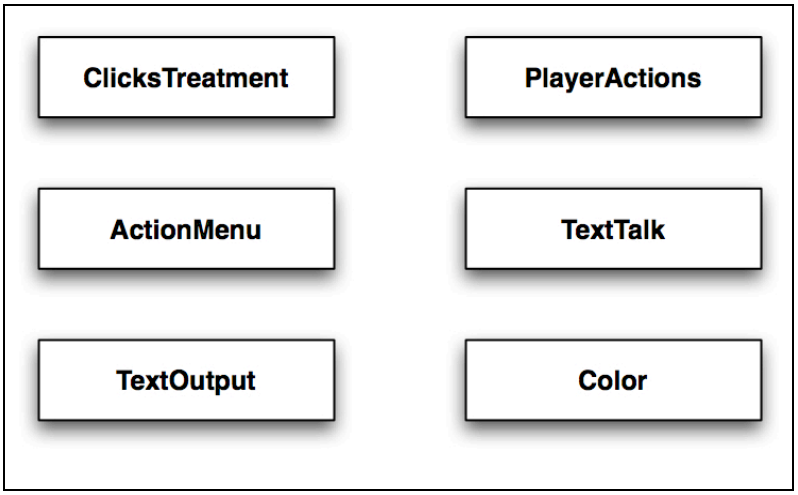


Fig 5.34 Diagrama de les classes més importants.

A la figura 5.35, podem veure la col·laboració entre les nostres classes per la classe *SCUMM*. En aquest diagrama s'aprecia com hem encapsulat les classes que implementen la funcionalitat de la nostra capa de portabilitat en una anomenada *ClicksTreatment*, d'aquesta manera aconseguim que només s'hagi d'incloure aquesta classe a dins de l'estructura de l'*ScummVM* fent que el nostre codi sigui menys invasiu. Així doncs, tots els punts d'entrada del nostre codi es faran mitjançant crides als mètodes d'aquesta classe aïllant d'aquesta manera tot el codi de la nostra capa del codi de l'*ScummVM*.

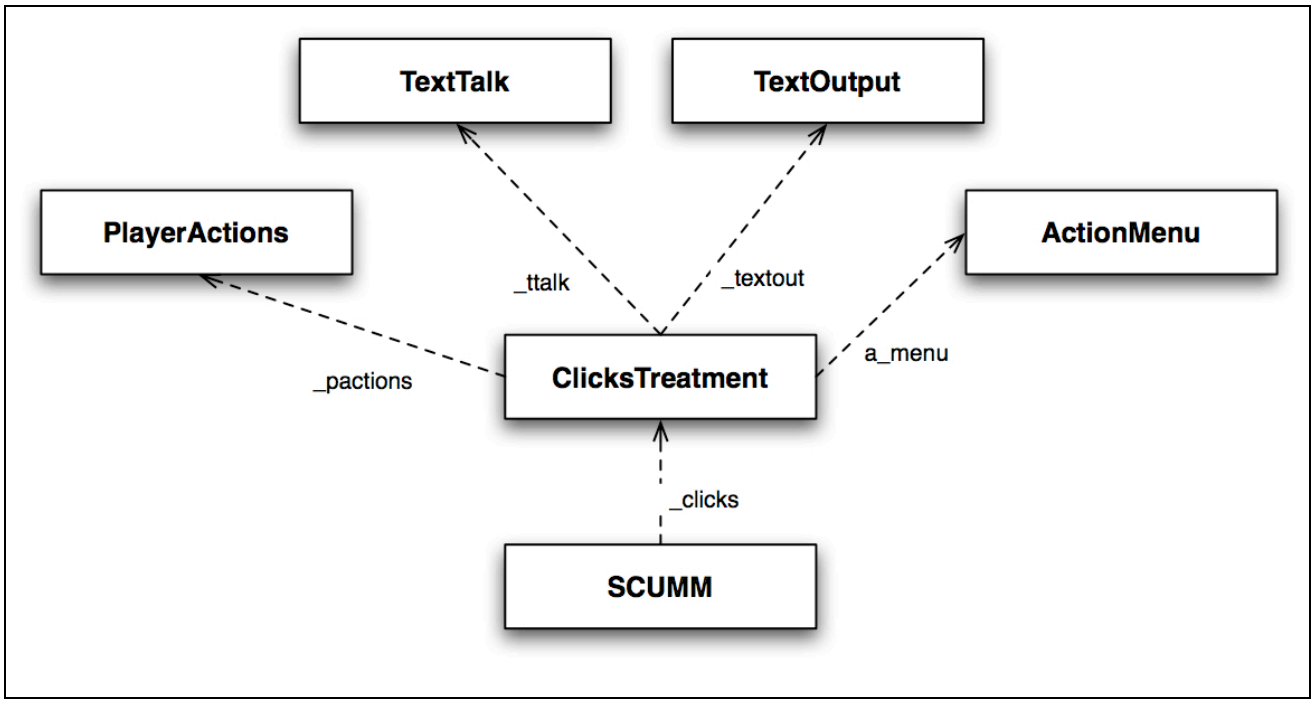


Fig 5.35 Diagrama de col·laboració.

Classe ClicksTreatment

Aquesta classe és l'encarregada de gestionar tot el sistema de clics. Cada cop que l'usuari fa un clic a la pantalla, aquesta classe identifica a quina zona de la pantalla s'ha clicat i actua en conseqüència (veure figura 5.36).

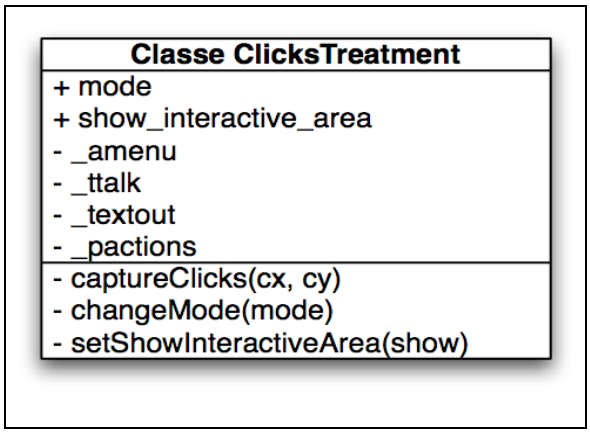


Fig 5.36 Classe ClicksTreatment.

Els mètodes més importants de la classe són els següents:

- ***captureClicks(cx, cy)*:** quan un usuari fa un clic sobre la pantalla, l'*ScummVM* detecta l'event i passa a tractar-lo. Com ja hem explicat anteriorment, el mètode de l'*ScummVM* encarregat de fer això és el *poll_event()*. El que nosaltres fem és utilitza aquest mètode com a punt d'entrada en el codi de l'*ScummVM*. Un cop el mètode *poll_event()* decideix que es tracta d'un event de clic utilitzem el mètode *captureClicks(cx, cy)* per poder tractar l'event. Utilitzem els paràmetres *cx* i *cy* per saber les coordenades del clic sobre la pantalla i poder actuar en funció de l'àrea on s'ha clicat i del mode de funcionament (Text/Sense Text o bé Mode d'imatge reduït sense filtrat/ Mode d'imatge reduït amb filtrat 2x2).
- ***setShowInteractiveArea(show)*:** serveix per mostrar o amagar el text del joc en el format de *PalmOS*. Utilitzem aquest mètode per alternar entre els modes Text/Sense Text, que hem descrit a l'apartat 5.7.

Els atributs *_amenu*, *_ttalk*, *_textout* i *_pactions*, són els objectes resultat d'instanciar les classes *ActionMenu*, *TextTalk*, *TextOutput* i *PlayerActions* respectivament. Hem encapsulat aquests objectes a dins de la classe *ClicksTreatment* perquè a l'hora d'introduir els nostres punts d'entrada al codi de l'*ScummVM* es fessin sempre passant pel mateix objecte. D'aquesta manera aconseguim

que el codi que hem afegit a dins de l'*ScummVM* sigui mínim. La instància de la classe *ClicksTreatment* s'anomena *_clicks*.

Classe ActionMenu

Aquesta classe és l'encarregada de tot el text interactiu dels menús d'accions (Ir a, coger, hablar a, …). El que fa és emmagatzemar el text del menú d'accions amb les seves coordenades. També conté els mètodes necessaris per mostrar el menú i amagar-lo (veure figura 5.37).

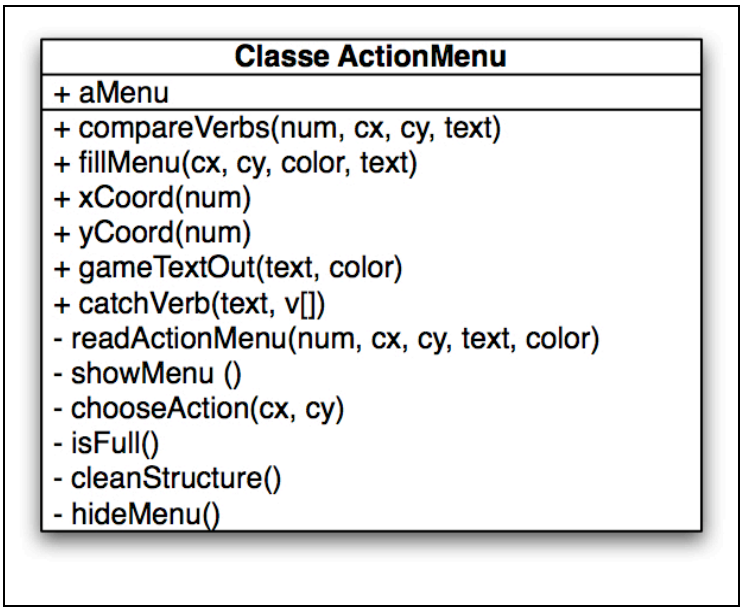


Fig 5.37 Classe ActionMenu.

Els mètodes més importants d'aquesta classe són els següents:

- ***readActionMenu(num, cx, cy, text, color)*:** aquest mètode és el que utilitzem per llegir i emmagatzemar les diferents opcions del menú d'accions del joc. A cada opció d'aquest menú se li assigna un número (paràmetre *num*), les coordenades reals (paràmetres *cx* i *cy*), el text de l'opció (paràmetre *text*) i finalment el color del text d'aquesta opció (paràmetre *color*). La crida d'aquest mètode es fa mitjançant la instància de la classe *ClicksTreatment* que, com ja hem explicat anteriorment, és l'únic que es troba a dins del codi de l'*ScummVM.* Concretament el punt d'entrada en el codi de l'*ScummVM* es fa dins del mètode *drawString()* que ja hem comentat a l'apartat 5.3.2.
- ***showMenu() i hideMenu()*:** aquests dos mètodes serveixen per amagar i mostrar el menú d'accions respectivament.

- ***chooseAction(cx, cy)*:** un cop l'usuari ha fet clic al damunt del menú d'accions del joc, aquest mètode és l'encarregat de trobar a la nostra estructura de dades l'opció del joc escollida per l'usuari, a partir de les coordenades *cx* i *cy* del clic. Posteriorment enviem a l'*ScummVM* les coordenades reals de l'opció escollida, ja que, a causa de la mida de lletra utilitzat, les coordenades de les accions que mostrem nosaltres no coincideixen amb les coordenades reals que té l'*ScummVM* emmagatzemades.

A la figura 5.38 que tenim a continuació, veiem un exemple del menú del joc. En el format de PC i el de la nostra aplicació de *PalmOS*.

Fig 5.38 Captura des de pantalla del menú en el format original de PC i del menú en el format de la nostra aplicació de PalmOS.

Classe TextTalk

La classe *TextTalk* (veure figura 5.39), emmagatzema el text interactiu de les converses entre el jugador i algun personatge del joc. Implementa els mètodes necessaris per mostrar i amagar el text quan sigui necessari i un sistema d'*scroll*, ja que el text de les converses pot arribar a ser molt llarg i no cabre a la pantalla. L'estructura és molt semblant a la de la classe *ActionMenu*, però s'ha fet per separat perquè aquest text no és el mateix durant tot el joc, va variant en funció de la conversa i el fet d'haver d'implementar la funcionalitat d'*scroll* fa que necessitem més dades a part de les coordenades i el text.

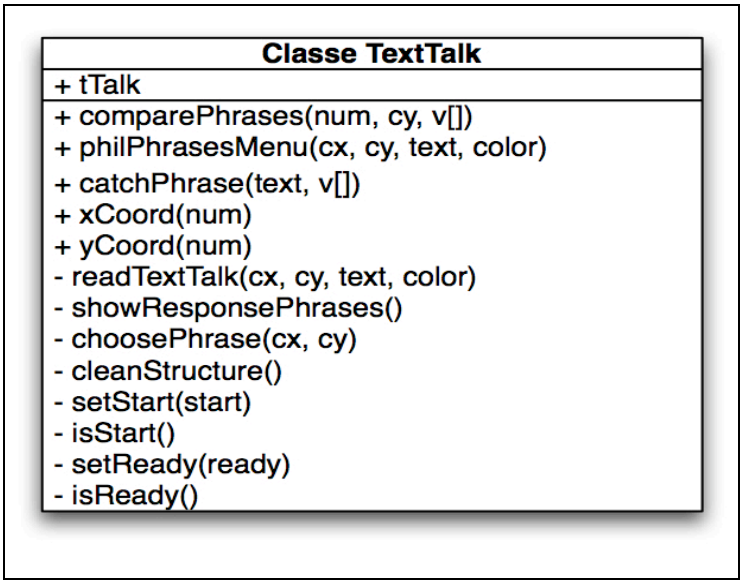


Fig 5.39 Classe TextTalk.

Els mètodes més importants d'aquesta classe són els següents:

- ***readTextTalk(cx, cy, text, color)*:** aquest mètode és el que utilitzem per llegir i emmagatzemar les diferents frases de resposta d'una conversa interactiva del joc. A cada frase d'aquest menú se li assignen les seves coordenades reals (paràmetres *cx* i *cy*), el text de la frase (paràmetre *text*) i finalment el color del text d'aquesta frase (paràmetre *color*). La crida d'aquest mètode es fa mitjançant la instància de la classe *ClicksTreatment*. El punt d'entrada en el codi de l'*ScummVM* es fa dins del mètode *drawString()* que ja hem comentat a l'apartat 5.3.2.
- ***showResponsePhrases()*:** aquest mètode és l'encarregat de mostrar les frases de la conversa per la pantalla. Implementa un sistema d'*scroll* per poder mostrar totes les possibles respostes de la conversa en el cas que aquestes no càpiguen a la pantalla. Això és degut a què la mida de lletra més petit que ens ofereix el *PalmOS* és molt més gran que el que utilitza l'*ScummVM* i per tant quan nosaltres mostrem les frases ens ocupa més espai.
- ***choosePhrase(cx, cy)*:** un cop l'usuari ha fet clic al damunt d'alguna resposta, aquest mètode és l'encarregat de trobar a la nostra estructura de dades la resposta escollida per l'usuari, a partir de les coordenades *cx* i *cy* del clic. Per poder enviar a l'*ScummVM* la resposta escollida. Ja que, a causa de la mida de lletra que ens proporciona l'*ScummVM,* les coordenades de la resposta que nosaltres mostrem difereixen amb les coordenades que té emmagatzemades l'*ScummVM*.

A la figura 5.40, podem veure un parell de captures de pantalla del text de les converses interactives.

Fig 5.40 Captura de pantalla del text de les converses interactives en el format original de PC i en el format de la nostra aplicació de PalmOS.

Classe TextOutput

Aquesta classe gestiona tot el text que surt per pantalla i no és interactiu. L'únic que fa és recollir-lo, formatejar-lo de manera que càpiga a la pantalla i mostrar-lo (veure figura 5.40).

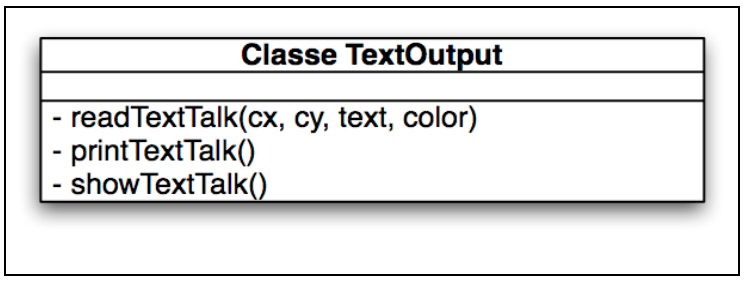


Fig 5.40 Classe TextOutput.

Els mètodes més importants d'aquesta classe són els següents:

- ***readTextTalk(cx, cy, text, color)*:** aquest mètode és el que utilitzem per llegir el text no interactiu que s'envia a la pantalla. A cada frase d'aquest menú se li assignen les seves coordenades reals (paràmetres *cx* i *cy*), el text de la frase (paràmetre *text*) i finalment el color del text d'aquesta frase (paràmetre *color*). La crida d'aquest mètode es fa mitjançant la instància de la classe *ClicksTreatment.* El punt d'entrada en el codi de l'*ScummVM* es fa

dins del mètode *CHARSET_1()* que ja hem comentat a l'apartat 5.3.2. Un cop llegit el text, aquest mètode crida al *printTextTalk()* per mostrar-lo per la pantalla.

- ***printTextTalk()*:** formateja el text amb la tipografia de l'*ScummVM* i el mostra per la pantalla.

En la figura 5.41 podem veure un parell de captures d'imatge del text que es mostra per pantalla quan els personatges del joc parlen entre ells. Les captures estan en el format de PC i en el format de la nostra aplicació de *PalmOS*.

Fig 5.41 Captures de pantalla del text de les converses dels personatges en el format de PC i de la nostra aplicació de PalmOS.

Classe PlayerActions

Aquesta classe tracta tot el text de les opcions escollides del menú d'acció. Aquest és el text situat entre les imatges del joc i el menú d'accions. També ens indica quan podem interactuar sobre els objectes dels jocs. El funcionament és molt semblant al de la classe *TextOutput*: emmagatzema el text momentàniament, el formateja i el mostra per pantalla (veure figura 5.42).

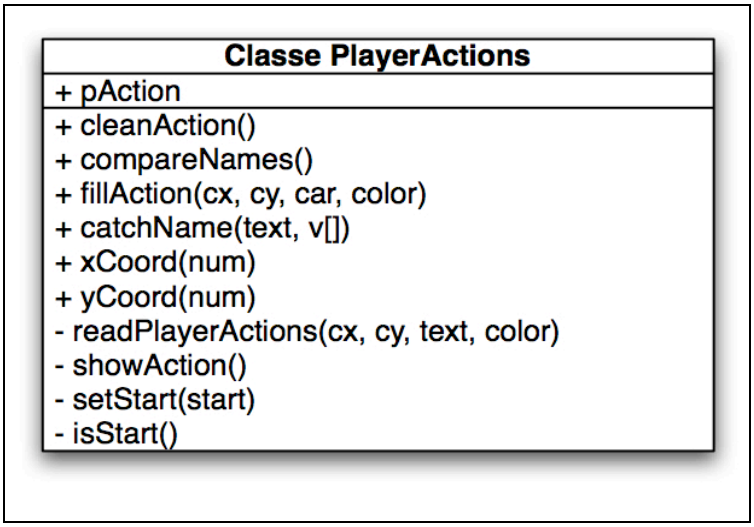


Fig 5.42 Classe PlayerActions.

Els mètodes més importants d'aquesta classe són els següents:

- ***readPlayerActions(cx, cy, text, color)*:** aquest mètode és el que utilitzem per llegir l'opció del menú d'accions escollida per l'usuari.. Un cop escollida una acció se li assignen les seves coordenades reals (paràmetres *cx* i *cy*), el text de l'acció (paràmetre *text*) i finalment el color del text d'aquesta acció (paràmetre *color*). La crida d'aquest mètode es fa mitjançant la instància de la classe *ClicksTreatment.* El punt d'entrada en el codi de l'*ScummVM* es fa dins del mètode *drawString()* que ja hem comentat a l'apartat 5.3.2.
- ***showAction()*:** formateja el text amb la tipografia de l'*ScummVM* i el mostra per la pantalla.

En la figura 5.43 podem veure un parell de captures d'imatge del text de les accions de l'usuari en els formats de PC i de la nostra aplicació de *PalmOS* respectivament.

Fig 5.43 Captures de pantalla mostrant el text de les accions de l'usuari en el format de PC i de la nostra aplicació de PalmOS.

Classe Color

Aquesta és la classe que utilitzem per realitzar operacions aritmètiques amb els colors dels píxels i així poder aplicar la reducció d'imatges amb filtres (veure figura 5.44). També hi trobem els mètodes necessaris per transformar un color de l'*ScummVM* a *RGB*. Aquests mètodes són especialment importants perquè en la versió *PalmOS* de l'*ScummVM* no treballem directament amb colors RGB com fèiem en la versió per PC. En aquesta versió treballem amb una paleta de colors de manera que primer hem d'obtenir el color RGB, fer les operacions necessàries i després obtenir l'índex de la paleta del nou color (veure apartat 5.4.4).

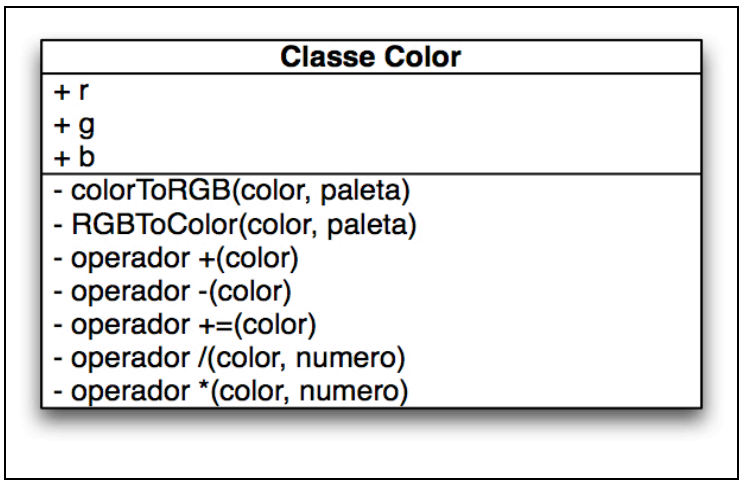


Fig 5.44 Classe Color.

Els mètodes més importants d'aquesta classe són els següents:

- ***colorToRGB(color, paleta)*:** donat un color (paràmetre color) que és un índex d'una paleta de colors i donada aquesta paleta (paràmetre paleta), ens retorna el color de la paleta en format *RGB*.
- ***RGBToColor(color, paleta)*:** donat un color (paràmetre color) en format *RGB* i una paleta de colors, retorna l'índex de la paleta (paràmetre paleta) que pertany al color en format *RGB*.
- ***Operadors +, -, +=*:** donats 2 colors en format *RGB*, realitza les operacions descrites.
- ***Operadors /, **:** donat un color i un número , realitza les operacions descrites.

Funcionalitat de reducció d'imatge:

La funcionalitat de reducció d'imatge implementa els mètodes necessaris per la reducció de les imatges dels jocs. Com ja hem explicat en apartats anteriors, hem implementat dos tipus de reducció d'imatge, una sense filtre i una amb filtre. Totes dues redueixen la imatge a la meitat de la seva mida original, és a dir, de 320x200 a 160x100 (veure figura 5.45).

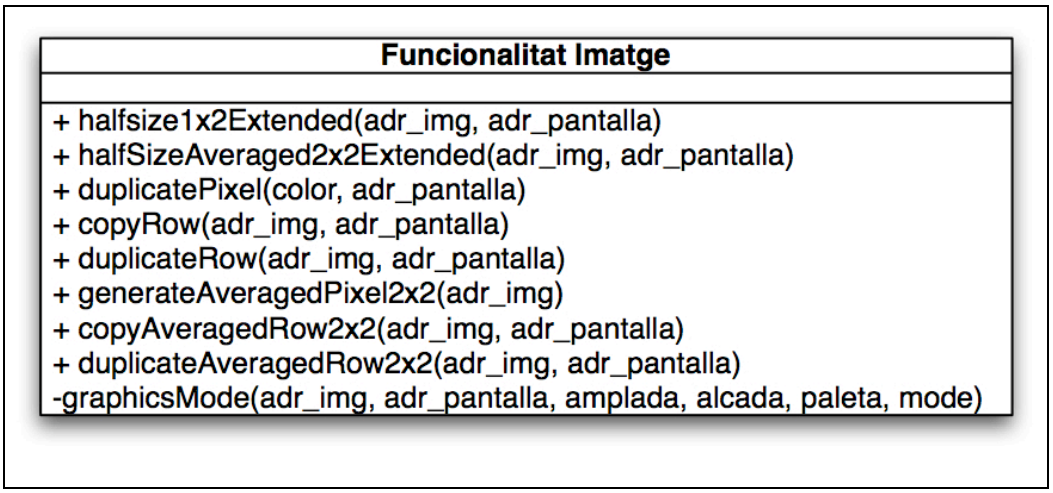


Fig 5.45 Funcionalitat de reducció d'imatge.

Els mètodes més importants d'aquesta classe són els següents:

- ***graphicsMode(adr_img, adr_pantalla, amplada, alcada, paleta, mode)*:** aquest mètode és l'encarregat de reduir les imatges dels jocs a la meitat utilitzant el mètode que vol l'usuari (paràmetre mode). Els modes de funcionament són els que ja hem descrit anteriorment,

reducció amb filtrat i reducció sense filtrat. Aquest mètode es crida a dins del mètode *update_screen()* que ja hem explicat a l'apartat 5.3.1. Els paràmetres que se li passen són l'adreça de la imatge original, l'adreça de la memòria de vídeo o adreça de destí de la imatge ja reduïda, l'amplada i alçada de la imatge original, la paleta de colors utilitzada, que pot varia en funció de cada joc, i finalment, el mode de reducció escollit.

- ***halfSize1x2Extended(adr_img, adr_pantalla)*:** redueix la imatge donada a la meitat sense utilitzar cap filtre de reducció.
- ***halfSizeAveraged2x2Extended(adr_img, adr_pantalla)*:** redueix la imatge donada a la meitat utilitzant un mètode de filtrat d'imatge que consta d'una matriu de pesos de 2x2 on cada element de la matriu té el mateix pes 0,5.

Com ja hem comentat anteriorment, la implementació d'aquestes classes, s'ha fet tenint en compte que el nostre codi ha de ser el menys intrusiu possible amb el codi original de l'*ScummVM*.

Per accedir a la nostra aplicació des de qualsevol punt del codi de l'*ScummVM*, necessitem que les nostres classes formin part de l'estructura de dades de la classe *SCUMM* (veure apartat 4.5). Com acabem de dir, la nostra intenció és fer-ho de la manera menys intrusiva possible, i és per això que hem implementat la classe *ClicksTreatment* de manera que pugui accedir a totes les altres classes de la nostra aplicació. De manera que només ens faci falta afegir aquesta estructura de dades a l'*ScummVM* i ja podrem introduir els punts d'entrada a la nostra aplicació en qualsevol punt del codi.

5.9 Portar a *PalmOS*

Per implementar la nostra capa de portabilitat, s'ha utilitzat gran part del codi que havíem implementat per PC.

El que s'ha fet ha estat adaptar part d'aquest codi a la Palm, i implementar tot el que faltava. El que expliquem en aquest apartat són els canvis que s'han hagut de fer al codi que ja teníem, a més a més de tota la part que hem implementat nova.

Per desenvolupar la nostra aplicació sobre *PalmOS* hem utitilitzat una versió prèvia de l'*ScummVM*, que ja treballava sobre aquesta plataforma, però amb la diferència que aquesta versió estava desenvolupada per una màquina que tenia unes característiques tècniques i un rendiment molt superior a la màquina per la qual nosaltres vam decidir desenvolupar el projecte. La principal

diferència era la resolució de la pantalla: nosaltres utilitzem un dispositiu que treballa a 160x160 píxels, mentre que la versió de l'*ScummVM* treballa al doble de la resolució. El que hem fet ha estat obviar la part del codi preparada per treballar a aquesta resolució, i desenvolupar la nostra aplicació utilitzant la base de *PalmOS* que ens proporcionava aquesta versió.

Per començar, el primer que s'ha de fer és instal·lar el compilador i totes les llibreries necessàries i aconseguir compilar la versió de l'*ScummVM* per *PalmOS*. Tot aquest procés està detallat a l'Apèndix 1.

Un cop tenim el compilador funcionant, es va desenvolupar la capa de portabilitat, que hem definit a l'apartat 5.6.

La nostra capa consta de les parts següents:

- Reducció d'imatge.
- Tractament del text.
- Tractament del events del llapis òptic i botons del PDA.

5.9.1 Reducció d'imatge

Per la reducció hem pogut utilitzar gran part del codi que vam implementar per PC (veure apartat 5.4). El sistema utilitzat per la reducció d'imatge ha estat el mateix, però tot i això, ens hem trobat amb certes diferències de funcionament entre la versió per PC i *PalmOS* que ens han obligat a modificar aquest codi.

Les principals diferències són les següents:

- En el tractament del color.
- Punt d'entrada del codi on recollim la imatge final per poder reduir-la.
- Els mètodes de filtrat utilitzats basant-nos en la velocitat de càlcul de la màquina.

5.9.1.1 Tractament del color de l'*ScummVM* per *PalmOS*

Pel que fa al tractament del color, ens trobem que a la versió per PC de l'*ScummVM* tenim el color representat per un número binari de 16 bits que segueix una codificació pròpia de *l'ScummVM*.

Aquesta codificació ens permet tenir les tres components *RGB* representades en un sol d'aquests números (veure apartat 5.4.3).

En la versió per *PalmOS* ens trobem que el *backend* de l'*ScummVM* no treballa d'aquesta manera, si no que ho fa de la manera següent:

- Per representar els colors, utilitza una paleta de colors pròpia per a cada joc, en funció de la profunditat de color de les imatges del joc (una paleta de 256 colors per als jocs de 8 bits, una de 65536 colors pels jocs de 16 bits, …). Dins de la paleta, els colors estan en el mateix format en el que els trobàvem a l'*ScummVM* per PC (veure apartat 5.4.3).
- La informació del color del píxel que trobem a la imatge del joc, és un índex a la paleta de colors, i no el color directament.

Tot això només afecta als mètodes de reducció d'imatge que utilitzen un mètode de filtratge. Pel que fa a la reducció sense filtres, podem utilitzar el codi que ja havíem implementat, ja que senzillament descartem la meitat dels píxels i no hem de fer cap operació aritmètica entre ells.

En canvi, pels mètodes de reducció d'imatge amb filtre, farà falta un mètode que, donat un índex del color que trobem a la imatge, retorni el color real del píxel, i un altre mètode que faci just el contrari. És a dir, que donat un color de la imatge, retorni un índex de la paleta de colors. Un cop tinguem aquests dos mètodes, podem utilitzar la implementació que teníem feta per PC, però afegint-t'hi un parell de passos.

Els passos a seguir són els següents:

- Llegir els píxels i obtenir els colors a través de l'índex a la paleta.
- Realitzar les operacions aritmètiques de la reducció per obtenir el color del nou píxel (veure apartat 5.4).
- Obtenir l'índex de la paleta que correspon al color resultant de la reducció.

Per tant, l'algorisme per generar la nova imatge serà el mateix que tenim a la figura 5.16 de l'apartat 5.4.2, el que canviarà serà l'algorisme del mètode *AplicarMascara* que tenim a la figura 5.46 del mateix apartat. Així doncs, a la figura següent podem observar el nou algorisme. Les parts que hem ressaltat en negreta són les que s'han afegit respecte l'algorisme anterior.

```
AplicarMascara (mascara, imatge_original, fila, columna, N, M,
                paleta)
F = fila
C = columna
per i = 0 fins a N fer
  per j = 0 fins a M fer
    pixel = buscaColorIndex(imatge_original[f][c], paleta);
    pixel += (imatge_original[f][c] * mascara[i][j])
    total = total + mascara[i][j]
    c = c + 1
  fi per
f = f +1
fi per
píxel = píxel / total
píxel = buscaIndexColor(pixel, paleta);
retorna píxel
```

Fig 5.46 Pseudocodi del càlcul del color del píxel resultant mitjançant l'aplicació d'una màscara de dimensions N x M

Si observem l'algorisme ens adonem que apareix un nou paràmetre (paleta) i dues funcions (*buscaColorIndex()*, *buscaIndexColor ()*). El paràmetre representa la paleta de colors que s'està utilitzant i les funcions serveixen per:

- ***buscaColorIndex(index, paleta):*** serveix per obtenir el color d'un píxel representat en el format de l'*ScummVM,* a partir d'un índex a una paleta de colors (veure figura 5.47).
- ***buscaIndexColor(color, paleta):*** serveix per obtenir, l'índex de la paleta que identifica a un color representat en el format de l'*ScummVM* (veure figura 5.48).

```
buscaColorIndex(index, paleta)
  color = paleta[index];
retorna color
```

Fig 5.47 Pseudocodi de l'obtenció d'un color de la paleta a partir d'un índex de la paleta.

En el cas d'obtenir un color de la paleta a partir de l'índex, és immediat, ja que la implementació de la paleta de l'*ScummVM* és un array de colors com el que podem veure a la figura 5.48.

```
buscaIndexColor(color, paleta)
mentre quedin_colors_a_la_paleta fer
  index = obte_index_color(color_paleta);
  si color_paleta = color fer
    retorna index
  altrament
    color_seguent
  fi si
fi mentre
```

Fig 5.48 Pseudocodi de l'obtenció d'un índex de la paleta a partir d'un color de la paleta.

El cas d'obtenir l'índex d'un color a la paleta a partir d'aquest color, és una mica més complicat, ja que hem de recórrer la paleta comparant els colors fins a trobar el color que busquem.

A la figura 5.49 podem veure un exemple d'una paleta de colors.

Fig 5.49 Exemple d'una paleta de colors de 4 bits (16 colors).

A la figura 5.50 podem veure un exemple de la correspondència d'un píxel d'una imatge amb el seu color en la paleta.

A l'esquerra de la figura hi trobem la imatge original, al centre hem ampliat una zona de la imatge per poder apreciar millor la correspondència amb el color. A la dreta de tot hi trobem la paleta de 256 colors que s'ha utilitzat per dibuixar la imatge.

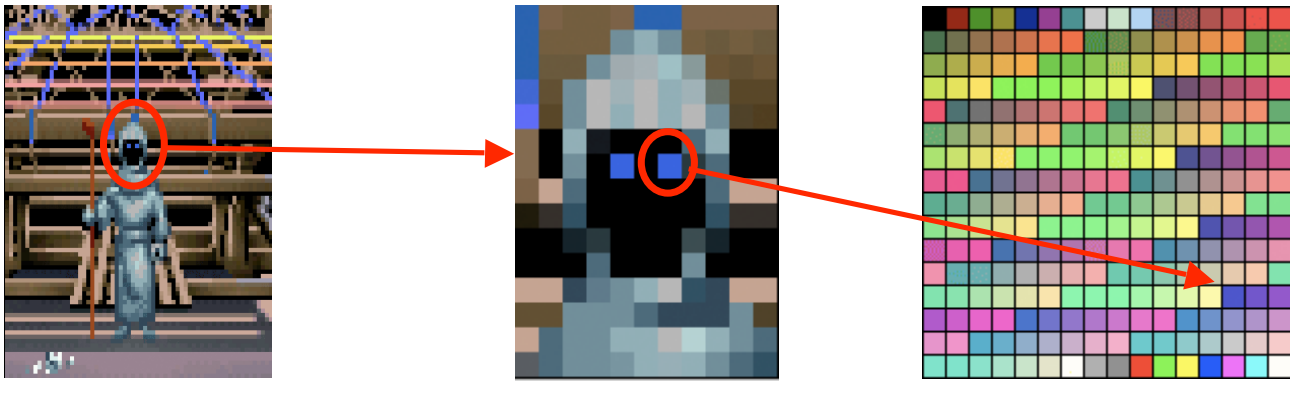


Fig 5.50 Exemple de la correspondència d'un color d'un píxel amb el color a la paleta.

5.9.1.2 Punt d'entrada del codi on recollim la imatge

En la versió de l'*ScummVM* de *PalmOS* trobem que la manera de recollir la informació per generar la imatge que es mostrarà finalment a la pantalla ha canviat, i que el mètode escollit per introduir el nostre punt d'entrada i poder reduir la imatge, no serveix.

Hem hagut de buscar cop entre els mètodes de l'*ScummVM* l'adient per poder utilitzar com a punt d'entrada i aplicar la nostra reducció d'imatge.

A la versió per PC utilitzàvem el mètode *update_screen()*, però aquest ja no ens serveix, bàsicament perquè a la implementació del *backend* (veure apartat 4.4.1) per *PalmOS* ha desaparegut. Enlloc d'aquest, ens trobem que n'hi ha tres que corresponen a tres maneres diferents de tractar les imatges abans de mostrar-les per pantalla i que en funció del que, ha escollit l'usuari s'utilitza un o altre. Els mètodes són els següents:

- ***update_screen_direct()*:** aquest mètode no tracta la imatge: tan bon punt rep la informació, la dibuixa a la pantalla.
- ***updade_screen_doublebuffer()*:** genera la imatge utilitzant un *buffer* (una memòria intermèdia) que va omplint fins a tenir la imatge sencera i un cop la té acabada, la mostra per pantalla. Quan es modifica una àrea de la pantalla respecte a la imatge anterior, regenera tota la imatge amb la informació que té emmagatzemada al buffer i la informació nova que li arriba.
- ***update_screen_flipping()*:** molt semblant al mètode del buffer, però sense emmagatzemar tota la imatge: utilitza un *buffer* (una memòria intermèdia) on hi guarda la informació de les àrees de la pantalla que s'han modificat respecte la imatge anterior. Un cop guardada tota la informació de l'àrea que s'ha modificat, l'envia a la pantalla perquè la dibuixi.

Per escollir el mètode que faríem servir com a punt de trencament, vam observar que en la versió que havíem implementat per PC. En aquesta versió disposàvem de tota la imatge i després la reduíem. Després de mirar el funcionament dels tres mètodes vam decidir utilitzar el mètode *update_screen_doublebuffer()*, ja que és en l'únic dels tres en el que tenim tota la imatge abans de dibuixar-la per pantalla i el seu funcionament és molt similar a l'*update_screen()* que teníem en la versió per PC. Vam descartar els altres dos perquè el fet de no tenir tota la imatge sencera complicava l'algorisme de reducció i havíem de modificar els mètodes de reducció que vam crear per la versió en PC. Un cop escollit el mètode per utilitzar com a punt d'entrada, vam utilitzar els mateixos mètodes de la versió per PC.

Per entendre millor el procés de reducció podem mirar la figura 5.51.

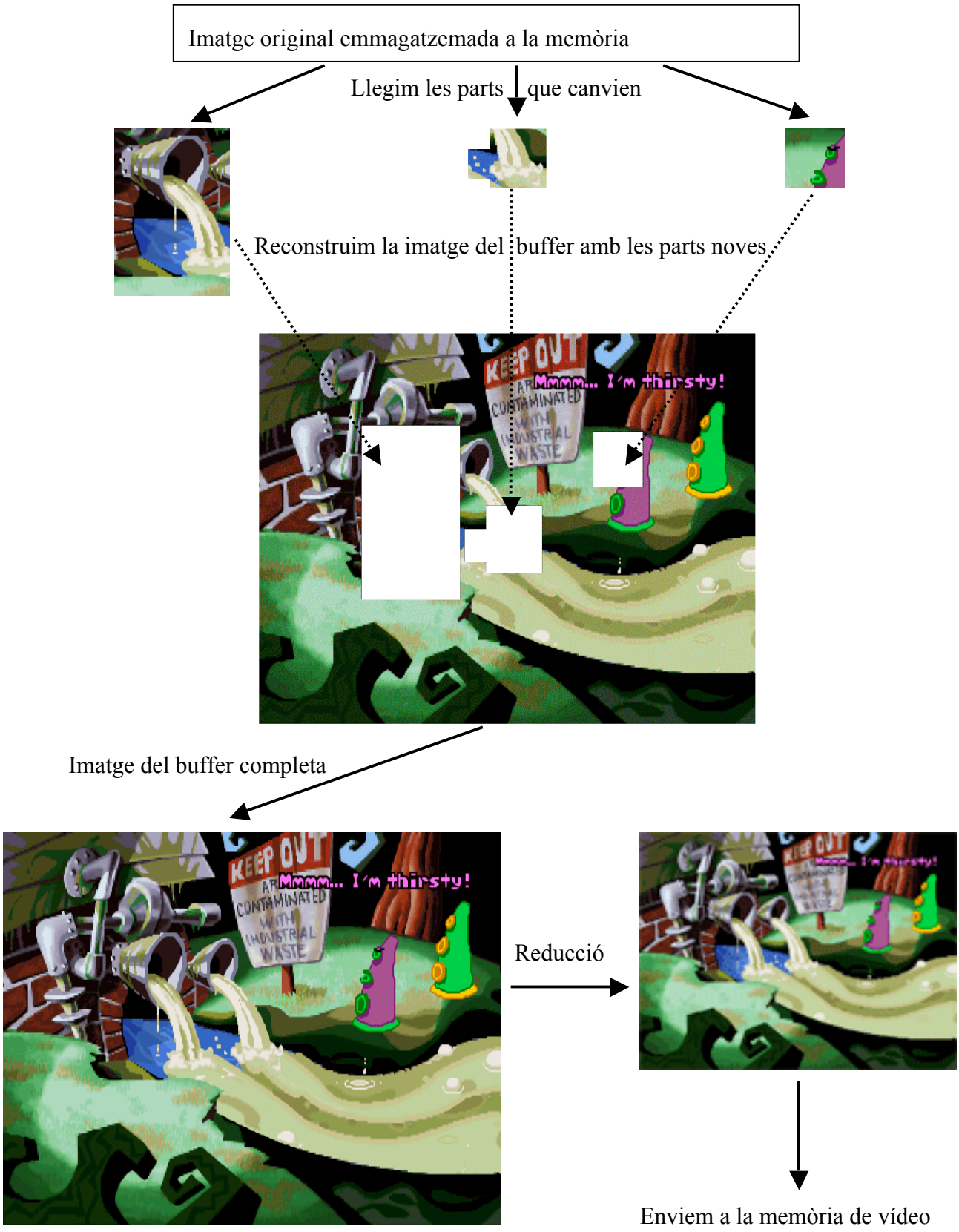


Fig 5.51 Exemple de la reducció d'imatge utilitzant el buffer de memòria.

El funcionament de la reducció mitjançant el buffer és el següent: tenim una estructura de dades que ens emmagatzema l'última imatge que s'ha dibuixat per pantalla. Per dibuixar la imatge següent, l'*ScummVM* ens proporciona només les parts de la imatge que s'han modificat respecte la imatge anterior, de manera que nosaltres només modifiquem les parts de la imatge que tenim en el buffer amb les parts noves que ens proporciona l'*ScummVM*.

Un cop hem copiat totes aquestes parts, en el *buffer* hi tenim la pròxima imatge que s'ha de mostrar per pantalla. És en aquest punt quan apliquem la reducció a la imatge que tenim en el buffer i copiem la imatge reduïda a la memòria de vídeo perquè el sistema la mostri per pantalla.

5.9.1.3 Mètodes de reducció utilitzats

A causa de la velocitat de càlcul de la PDA utilitzada, hem hagut d'escollir quins mètodes de reducció implementàvem.

Com ja hem explicat en l'apartat de reducció d'imatge per PC, vam decidir implementar mètodes que creiem que eren poc costosos per així poder-los posar a la nostra capa de portabilitat. Però un cop els vam haver posat, vam veure que no tots ens servien.

Fins al moment, teníem els mètodes implementats per PC que són els següents:

- Reducció 2:1 sense filtrat.
- Reducció 2:1 utilitzant un una matriu de pesos 2x2 on cada element té el mateix pes.
- Reducció 2:1 utilitzant un una matriu de pesos 3x3 on cada element té el mateix pes.

El fet del canvi en el tractament del color a partir d'una paleta de colors ha fet augmentar el número de càlculs necessaris per píxel, ja que cada cop que calculem el color del nou píxel s'ha de recórrer la matriu que representa la paleta per obtenir l'índex d'aquest color. Tot i que s'ha intentat optimitzar el funcionament de la reducció evitant realitzar els càlculs quan els píxels a reduir eren del mateix color, no ha estat suficient.

Després de fer vàries proves, ens vam veure obligats a eliminar el mètode de reducció utilitzant una matriu de pesos 3x3, ja que en aquest mètode la velocitat del joc s'alentia moltíssim, fent que fos impossible jugar-hi. Per tant els modes d'imatge que hem utilitzat finalment són els següents:

- Reducció 2:1 sense filtrat.
- Reducció 2:1 utilitzant un una matriu de pesos 2x2 on cada element té el mateix pes.

5.9.2 Tractament del text

Com hem explicat a l'apartat 5.5.1 quan treballàvem amb la versió per PC vam identificar els tipus de text i vam implementar mètodes per capturar la informació necessària per poder-los tractar com volguéssim i mostrar-los pel nostre compte. Els diferents tipus de text són (veure apartat 5.3.2):

- Text no interactiu.
    - Text de les converses entre personatges, o del narrador.
    - Text escollit del menú d'acció o de les converses interactives.
- Text interactiu.
    - Text dels menús d'acció que hi ha per cada joc.
    - Text interactiu de les converses.

Aquests tipus de text són els que la nostra aplicació ha de controlar i mostrar utilitzant la tipografia per defecte del *PalmOS*, ja que a causa de la reducció de les imatges del joc, el text original del joc queda il·legible. La nostra solució no elimina el text original dels jocs, si no que el dibuixa igualment, ja que la nostra intenció no és la de variar el funcionament del joc.

A diferència de la reducció d'imatge, per poder capturar el text, hem pogut utilitzar els mateixos punts d'entrada que vam fer servir quan treballàvem per PC. Això és perquè els punts d'entrada per capturar el text es troben tots dins del codi de l'*SCUMM*, i aquesta part és independent de la plataforma amb la que treballem.

El fet que la part que controla el text no depengui de la plataforma ens ha servit per poder aprofitar gaire bé tot el codi que servia per tractar el text que vam implementar en la versió per PC. Tot i així vam haver de fer canvis i ampliar les funcionalitats del que vam implementar.

Alguns dels canvis, són a causa de les limitacions de la nova plataforma (*PalmOS*) i uns altres a les limitacions de la pantalla del dispositiu on s'executa la nostra aplicació. Com ja hem comentat a l'apartat 5.5.1, vam deixar tota una part de la implementació del control del text per quan treballéssim sobre *PalmOS*, perquè intuíem que si el desenvolupavem per PC l'hauríem de canviar. La majoria dels canvis la versió per PC han estat en la manera de mostrar el text per pantalla, en PC ens limitàvem a mostrar-lo a la consola (*printf*), per poder comprovar que el text i les dades que recollíem a través del nostre punt d'entrada a l'*ScummVM* era correcte. En *PalmOS*, el que fem és mostrar el text per la pantalla juntament amb les imatges del joc. Aquest ha estat un dels canvis comú a tots els tipus de text. Per donar format al text i mostrar-lo, hem canviat la crida a *printf()* de

la versió per PC i hem utilitzat les llibreries d'entrada i sortida que ens proporciona l'*SDK* de *PalmOS*.

5.9.2.1 Text de les converses entre personatges, o del narrador

Pel que fa a aquest tipus de text, hem aprofitat tota l'estructura de dades i la funcionalitat que vam implementar per PC, però adaptant-la als canvis que ens demanava la plataforma. Recordem que aquest text no l'emmagatzemàvem durant tota l'execució del joc, si no que, com que era un text que anava canviant, senzillament el recollíem juntament amb el seu color i el seu format (canvis de línia, tabuladors, ...) i un cop recollit el mostràvem per pantalla.

Per fer el que acabem de descriure, hem implementat la classe *TextOutput* que trobem descrita a l'apartat 5.8. En aquesta classe hi trobem els atributs i els mètodes que vam implementar en el seu moment per PC, però hi hem afegit el mètode *printTextTalk(),* encarregat de mostrar el text .

Recordem que quan treballàvem per PC el nostre objectiu només era tenir controlat el text i prou, sense definir com el mostràvem per la pantalla. Per tant, hem hagut de decidir a on i com mostrem aquest text. La pantalla del nostre dispositiu és de 160x160 píxels i un cop reduïdes les imatges del joc ocupen 160x100 píxels. Això ens deixa una àrea de 60x160 píxels on podem mostrar aquest tipus de text (veure figura 5.52) i hem destinat una àrea de 30x160 píxels a la part superior per mostrar-hi aquest tipus de text, ja que així podem visualitzar el text al mateix temps que la pantalla del joc.

Com que l'àrea on hem decidit mostrar el text és un espai limitat, hem decidit trencar el format del text (canvis de línia, tabuladors, ...) que ens arriba, per poder mostrar el màxim número de caràcters alhora. A la figura 5.52 podem veure com es mostra finalment aquest text comparat amb la versió original: a la part superior hi veiem el text que hem capturat mentre que a dins de la imatge hi veiem el text original, il·legible a causa de la reducció.

Fig 5.52 Captures de pantalla on podem veure l'àrea de la pantalla on mostrarem el text i el resultat de mostrar-lo. Observar l'error en el text original (Re-eliga) i en el text que mostra l'aplicació.

5.9.2.2 Text escollit del menú d'acció o de les converses interactives

Pel que fa a aquest tipus de text, també hem pogut aprofitar tota l'estructura de dades i la funcionalitat que vam implementar per PC. Però, com en el cas anterior, també hem hagut de fer certs canvis per poder-lo adaptar a la plataforma.

Aquest tipus de text no el guardem durant tota l'execució del joc, si no que, com que és un text que va canviant, senzillament el recollim juntament amb el seu color un cop recollit i el mostrem per pantalla.

Hem implementat una classe anomenada *PlayerActions* (veure apartat 5.8) que és l'encarregada de tractar aquest tipus de text i de mostrar-lo per la pantalla. La classe inclou tots els mètodes implementats quan treballàvem sobre PC, però hem afegit el mètode *showAction()* que és l'encarregat de mostrar el text per pantalla (veure apartat 5.8).

Quan treballàvem sobre PC mostràvem tot el text a la consola, però ara que treballem amb *PalmOS*, hem de decidir a quina part de la pantalla el mostrem. Hem de tenir en compte que l'espai és limitat i que volem que no interfereixi amb el joc.

Com ja hem mencionat, la pantalla del nostre dispositiu és de 160x160 píxels i un cop reduïdes les imatges del joc ocupen 160x100 píxels. Quan reduïm la imatge el text original queda il·legible, ens adonem que podem utilitzar la mateixa zona on es mostra originalment aquest text, per mostrar-hi al

damunt el text en el format de *PalmOS*. A la figura 5.53 podem veure la zona escollida per mostrar el text i el resultat de mostrar-lo mitjançant la tipologia de *PalmOS*.

Fig 5.53 Àrea on es mostra el text del menú d'acció escollit i el resultat de mostrar-lo.

5.9.2.3 Text dels menús d'acció que hi ha per cada joc

Pel que fa a aquest tipus de text, hem pogut aprofitar l'estructura de dades i la funcionalitat que vam implementar per PC. Però, com en el cas anterior, també hem hagut de fer certs canvis per poder-lo adaptar a la plataforma.

Aquest tipus de text és llegeix a l'inici del joc i es guarda en la nostra estructura de dades durant tota l'execució del joc, perquè es manté invariant. A causa d'aquest fet, el primer canvi respecte el que havíem implementat per PC ha estat afinar la mida de l'estructura de dades que guarda aquest text. La mida de la memòria que disposem en el dispositiu amb el que treballem és bastant limitat, per tant, és molt important optimitzar la quantitat de memòria que utilitzarem. Per tot això, hem reduït la mida de l'estructura de dades a 16 accions i a una mida màxima de 14 caràcters per acció del menú (veure figura 5.54).

Per poder mostrar el text per pantalla, igual que en els casos anteriors, hem hagut de buscar una àrea de la pantalla on el pogués simvisualitzar i no interferís amb el funcionament del joc. Com que el text de les accions del menú del joc queden il·legibles, hem decidit aprofitar el mateix espai per mostrar les accions en el format de text del *PalmOS*.

El format de *PalmOS* és més gran que la mida de text original. És per això que, per aprofitar el màxim l'espai del que disposem, no mostrarem les accions del joc en el mateix ordre en el que es dibuixen en el joc si no que les anirem posant una al costat de l'altre per aprofitar tota la pantalla (veure figura 5.54). El mètode encarregat de mostrar les accions és el *showMenu()* (veure apartat 5.8). Per solucionar el cas en el que no caben totes les accions possibles, hem implementat un sistema d'*scroll* vertical que ens permet veure totes accions disponibles.

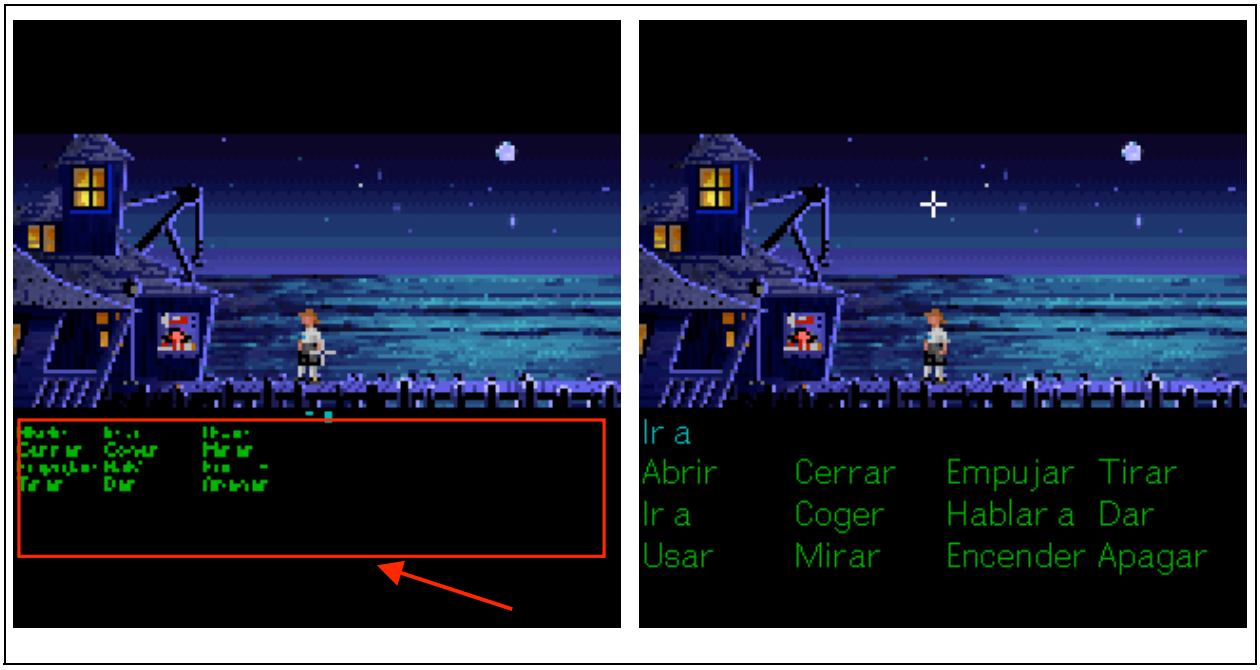


Fig 5.54 Àrea on es mostra el text del menú d'acció i el resultat de mostrar-lo utilitzant la tipografia del PalmOS.

Per tractar i emmagatzemar el text del menú d'accions hem implementat la classe *ActionMenu* (veure apartat 5.8). La classe *ActionMenu* conté l'estructura de dades modificada que vam implementar per PC i una sèrie de mètodes per poder tractar el text. Aquestes modificacions són a causa del canvi de funcionament a l'hora d'identificar l'acció seleccionada, que explicarem més endavant. La nova estructura és que podem veure a la figura 5.55. Com podem observar ha desaparegut l'atribut *num* que guardava el número de l'acció que ens servia com a identificador quan l'usuari escollia una acció mitjançant la consola. En comptes d'aquest atribut, en trobem quatre de nous (*cx_inici, cy_inici*) i (*cx_fi, cy_fi*), que representen les coordenades de la pantalla entre les quals escrivim l'acció.

```
classe ActionMenu
  tipus taccio
    enter cx
    enter cy
    enter cx_inici
    enter cy_inici
    enter cx_fi
    enter cy_fi
    color colorText
    cadena accio[14]
  fi tipus
  taccio llistaAccions[16]
  ...
fi classe
```

Fig 5.55 Pseudocodi de l'estructura de dades dins les classes on guardem el text de les accions del menú.

El nou procés utilitzat per omplir l'estructura i identificar l'acció és el següent: per omplir l'estructura de dades, utilitzem el mètode *readPlayerActions()* de la classe *ActionMenu* com a punt d'entrada dins del mètode *drawString()* de la classe *SCUMM*. Un cop plena, assignem a cada acció les coordenades de la pantalla on la mostrarem, tenint en compte la mida de lletra i la mida en píxels del que ocupa cada acció, escrita utilitzant la tipografia de *PalmOS*. Guardem aquestes coordenades per poder identificar l'acció sobre la qual ha fet clic l'usuari. És a dir, quan l'usuari fa clic al damunt d'una acció, la nostra aplicació captura les coordenades d'aquest clic, i busca a dins de l'estructura de dades l'acció que correspon a aquestes coordenades. Quan la troba envia un clic al sistema amb les coordenades reals de l'acció.

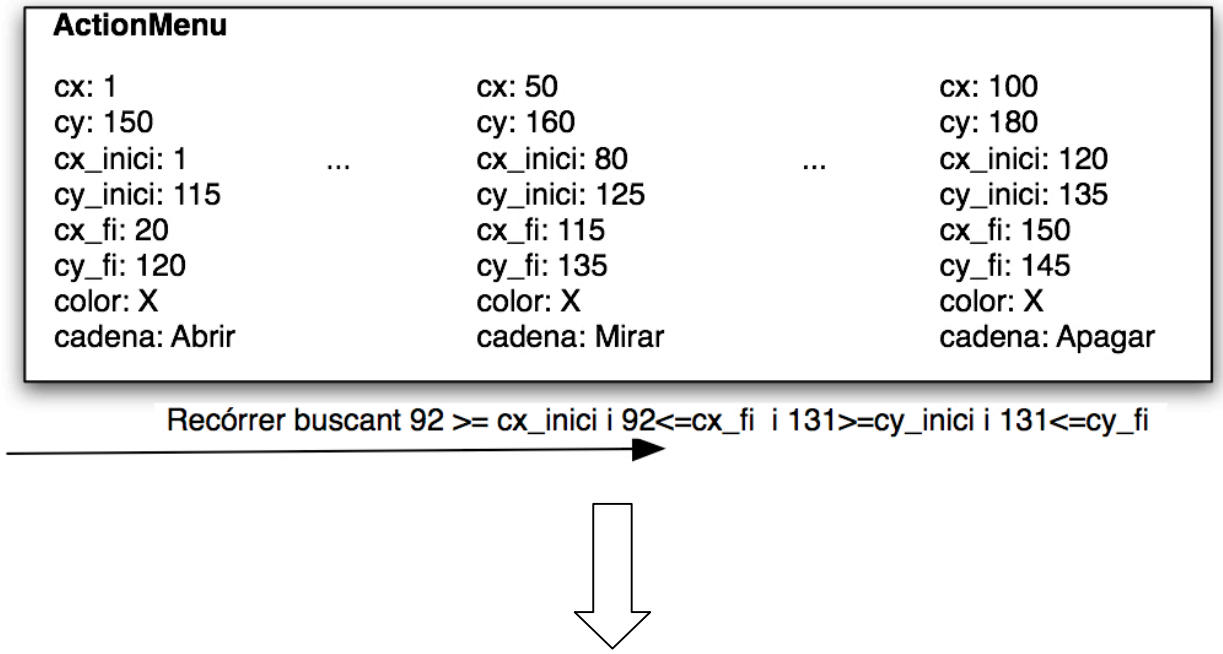
A la figura 5.56 podem veure el procés de selecció de l'acció de forma gràfica.

1 .- L'usuari fa clic les coordenades (92,131) al damunt de l'acció *Mirar*. L'acció *mirar* la tenim escrita entre les coordenades (80,125) i (115,135). Aquestes coordenades les tenim emmagatzemades a l'estructura de dades.

2.- A la imatge real l'acció *mirar* està escrita a entre les coordenades (45,220) i (95,240). Aquestes coordenades les tenim emmagatzemades a l'estructura de dades.

3.- Busquem a l'estructura de dades les coordenades reals de l'acció escollida, utilitzant les

coordenades del clic (92,131) i comparant-les amb les coordenades del text de cada acció.



4.- Un cop trobada la coincidència amb l'acció *Hablar a*, generem un clic a les coordenades reals d'aquesta (50,220) i l'enviem a l'*ScummVM*.

Fig 5.56 Procés de selecció de l'acció del joc que ha clicat l'usuari.

5.9.2.4 Text interactiu de les converses

Pel que fa a aquest tipus de text, també hem pogut aprofitar tota l'estructura de dades i la funcionalitat que vam implementar per PC. Però com en els casos anteriors, també hem hagut de fer certs canvis per poder-lo adaptar a la plataforma.

Aquest tipus de text és llegeix cada vegada que l'usuari inicia una conversa amb algun personatge del joc. Les frases d'aquesta conversa s'emmagatzemen durant tota l'estona que duri la conversa però no al llarg del joc.

Aquí també hem de reduir al màxim la mida d'aquesta estructura de dades com hem hagut de fer en el cas anterior, ja que el dispositiu té una mida de memòria força limitat. Hem reduït la mida a 30 frases i a 80 caràcters per frase.

Per poder mostrar el text per pantalla, també hem hagut de buscar una àrea de la pantalla on el poguéssim visualitzar i no interferís amb el funcionament del joc. L'àrea escollida en aquest cas ha estat la mateixa on es mostren les frases de les converses en el joc original, ja que com passava en els altres casos les frases es tornen il·legibles. El mètode encarregat de mostrar les frases és el *showResponsePhrases()* (veure apartat 5.8). El que fa és modificar el format (canvis de línia, tabuladors, ...) original d'aquestes frases per poder aprofitar el poc espai del que disposem.

A causa d'aquesta manca d'espai, també hem hagut d'implementar un sistema d'*scroll* vertical per poder mostrar totes les frases de les converses ja que en pantalla només podem visualitzar quatre línies.

En la figura 5.57 podem veure un exemple de com mostrem les frases d'una conversa.

Fig 5.57 Àrea on es mostra el text del menú d'acció i el resultat de mostrar-lo utilitzant la tipografia del PalmOS.

Per tractar i emmagatzemar el text de les converses hem implementat la classe *TextTalk* (veure apartat 5.8). Aquesta classe agrupa l'estructura de dades modificada que vam implementar per PC i una sèrie de mètodes per poder tractar el text. Aquestes modificacions són a causa del canvi de funcionament a l'hora d'identificar la frase seleccionada, que explicarem més endavant.

La nova estructura és la que podem veure a la figura 5.58. En PC s'assignava un número a cada frase i l'usuari escollia la que volia mitjançant el teclat numèric. Aquest atribut ha desaparegut en el cas de *PalmOS*, i apareixen quatre nous atributs (*cx_inici, cy_inici, cx_f, cy_fi*) que representen les coordenades de la pantalla entre les quals està escrita la frase.

```
classe TextTalk
  tipus tphrase
    enter cx
    enter cy
    enter cx_inici
    enter cy_inici
    enter cx_fi
    enter cy_fi
    color colorText
    cadena phrase[80]
  fi tipus
  tphrase llistaPhrases[30]
  ...
fi classe
```

Fig 5.58 Pseudocodi de l'estructura de dades del text de les converses interactives.

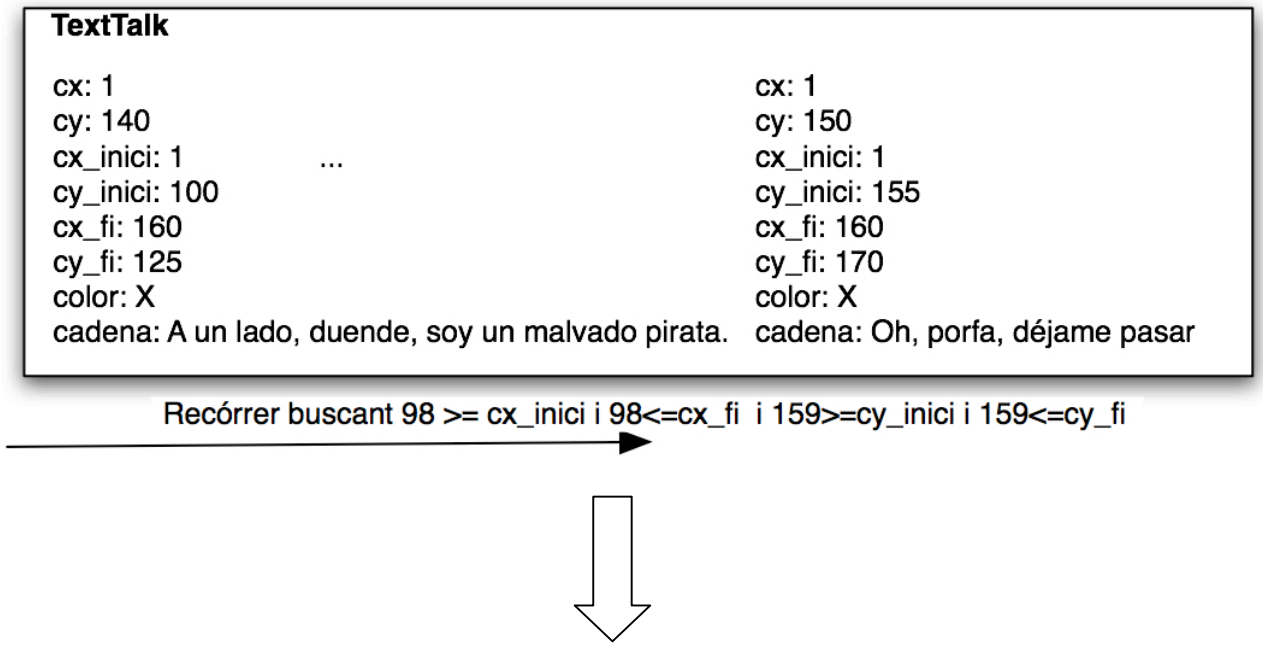
El procés utilitzat per omplir l'estructura i tractar identificar la frase és el següent: per omplir l'estructura de dades, utilitzem el mètode *readTextTalk()* de la classe *TextTalk* com a punt d'entrada dins del mètode *drawString()* de la classe *SCUMM*. Un cop plena, assignem a cada frase les coordenades de la pantalla on la mostrarem, tenint en compte la mida de lletra, la mida en píxels i el número de línies, del que ocupa cada frase escrita utilitzant la tipografia de *PalmOS*. Guardem aquestes coordenades per poder identificar la frase sobre la qual ha fet clic l'usuari. És a dir, quan l'usuari fa clic al damunt d'una frase, la nostra aplicació captura les coordenades d'aquest clic i busca a dins de l'estructura de dades la frase que correspon a aquestes coordenades. Quan la troba envia un clic al sistema amb les seves coordenades reals.

A la figura 5.59 podem veure el procés de selecció de la frase de forma gràfica.

1 .- L'usuari fa clic les coordenades (98,159) al damunt de la frase *Oh, porfa, déjame pasar*. La frase la tenim escrita entre les coordenades (1,155) i (160,170). Aquestes coordenades les tenim emmagatzemades a l'estructura de dades.

2.- A la imatge real l'acció mirar està escrita entre les coordenades (1,150) i (320,160). Aquestes coordenades les tenim emmagatzemades a l'estructura de dades.

3.- Busquem a l'estructura de dades les coordenades reals de la frase escollida, utilitzant les coordenades del clic (98,159) i comparant-les amb les coordenades del text de cada frase.



4.- Un cop trobada la coincidència amb l'acció *Hablar a*, generem un clic a les coordenades reals d'aquesta (50,220) i l'enviem a l'*ScummVM*.

Fig 5.59 Procés de selecció de la frase interactiva d'una conversa del joc que ha clicat l'usuari.

5.9.3 Control dels events

En la part del nostre codi que vam desenvolupar per PC (veure apartat 5.5.2), vam decidir detectar i tractar els events generats pel teclat i els generats pel ratolí. Com que, en un dispositiu amb *PalmOS*, principalment no disposem de cap d'aquests dos elements, hem utilitzat els que serien els equivalents.

En el nostre dispositiu disposem d'un llapis òptic que fa les mateixes funcions que un ratolí i també disposem dels botons del dispositiu, els quals podem utilitzar com si fossin les tecles d'un teclat. És ben cert que el dispositiu només consta de 6 botons que són molts menys que les tecles en un teclat, però amb aquests 6 botons ja en fem més que suficient per poder jugar als jocs que interpreta l'*ScummVM*. Per tant, pel que fa als controls dels events en *PalmOS* hem tractat amb els dels dos tipus d'events següents:

- Event de clic de llapis òptic.
- Events de clic dels botons del PDA.

Per poder-ho fer, hem encapsulat els mètodes que utilitzàvem per tractar els events que vam implementar per treballar en PC en la classe *ClicksTreatment* (veure apartat 5.8).

El fet de canviar de plataforma ha significat haver de buscar de nou el punt d'entrada al codi de l'*ScummVM*, ja que el control d'events és dependent de la plataforma utilitzada. La dependència és deguda a que el sistema de control dels events utilitza crides al sistema operatiu, i aquestes són diferents en funció del sistema en el que estem treballant.

Necessitem capturar aquests events al mateix temps que l'*ScummVM,* i tractar-los abans que ell ho faci. Per tant el sistema utilitzat per buscar a quina part del codi de l'*ScummVM* es recullen i es tracten els events ha estat el mateix que quan treballàvem per PC.

El mètode en qüestió s'anomena *poll_event(),* igual que el mètode que els tractava en PC i es troba ubicat a dins de la classe *OSystem::PalmOS*, que és l'homòloga a la classe de sistema de l'*ScummVM* implementada per PC.

El funcionament del mètode és pràcticament el mateix, recull els events que es generen al sistema, tant si són generats pel propi usuari o pel mateix sistema, els identifica i seguidament els tracta.

És, doncs, en aquest mètode en el que nosaltres posarem la crida a la nostra classe C*licksTreatment* perquè tracti els events que ens interessa. Aquesta crida anirà just després que el mètode *poll_event()* identifiqui l'event.

5.9.3.1 Tractament de l'event de clic del llapis òptic

Quan el mètode *poll_event()* recull un event de clic del llapis òptic, identifica les coordenades de la pantalla d'on s'ha fet aquest clic mitjançant les dades de l'event que ens proporciona el sistema. Un cop recollides, envia al motor de l'*ScummVM* un event de clic sobre la pantalla amb les coordenades perquè ell actuï en conseqüència.

A causa de la reducció d'imatge, quan un usuari fa clic a la pantalla, el sistema envia unes coordenades errònies a l'*ScummVM*: concretament, envia el valor de les coordenades dividit per la meitat perquè hem reduït la resolució del joc. Una de les nostres tasques ha estat capturar aquestes coordenades i corregir-les, abans que el mètode *poll_event* , les enviï al motor de l'*ScummVM*. Per fer això, quan detectem un event de clic sobre la pantalla, fem una crida al mètode *captureClicks()* de la classe *ClicksTreatment* (veure apartat 5.8) i li enviem les coordenades per a què les corregeixi.

Aquest mètode no només corregeix les coordenades sinó que depenent del mode actual de funcionament de la nostra aplicació (veure apartat 5.6.1) i de la zona on l'usuari ha fet clic, realitza una operació o una altra.

A la figura 5.60 podem veure les diferents àrees o zones en les quals hem dividit la pantalla del joc.

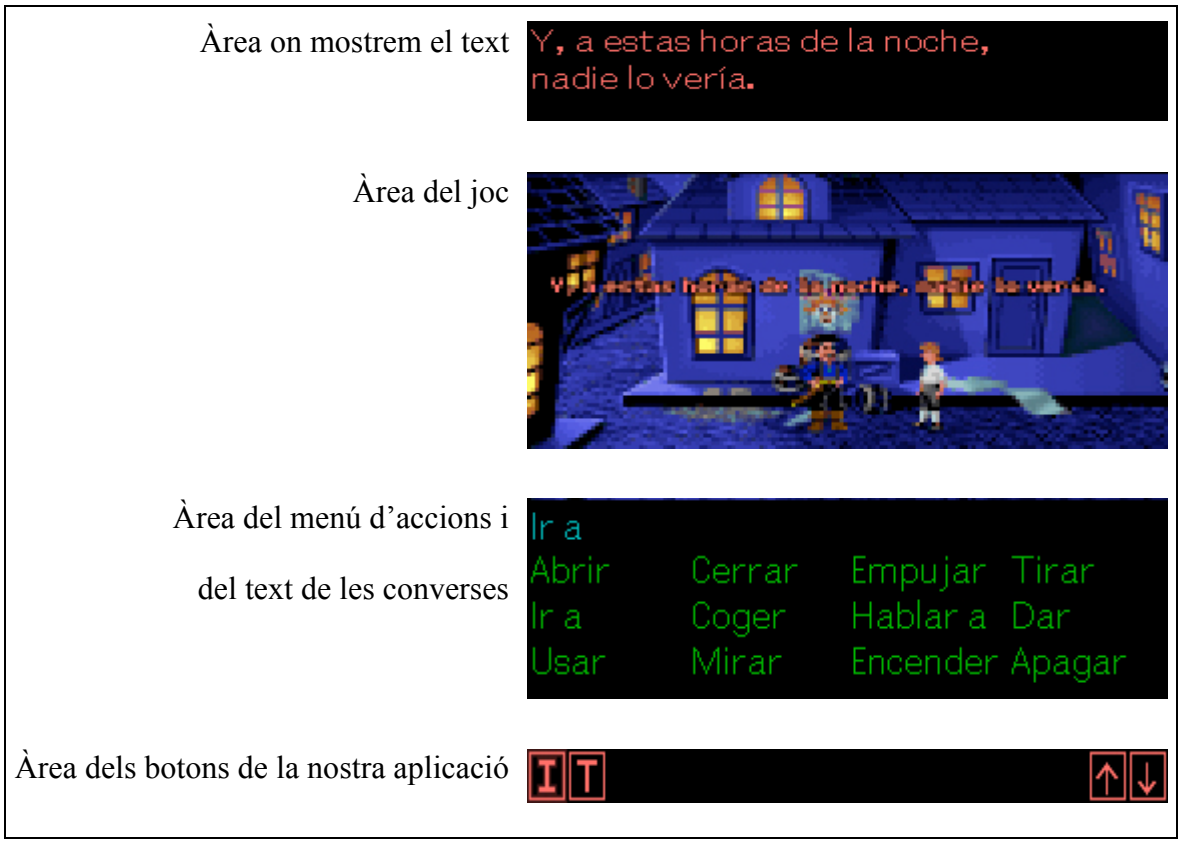


Fig 5.60 Diferents àrees de pantalla.

Com ja hem dit abans, en funció de la zona de la pantalla on ha fet clic l'usuari i del mode de funcionament de la nostra aplicació, realitzarem les operacions següents:

- **Mode Sense Text:**
    - Independentment de l'àrea on hagi fet clic l'usuari, el mètode *captureClicks()* rep les coordenades del clic, les multiplica per dos (a causa de la reducció d'imatge a la meitat) i envia les coordenades corregides a l'*ScummVM*.
- **Mode Text:**
    - **Clic a l'àrea de joc:** el mètode *captureClicks()* rep les coordenades del clic, les multiplica per dos i envia les coordenades corregides a l'*ScummVM*.
    - **Clic a l'àrea d'accions:** el mètode *captureClicks()* rep les coordenades del clic, crida al mètode *chooseAction()* passant-li aquestes coordenades per poder identificar l'acció sobre la que s'ha fet el clic i recollir les coordenades reals de l'acció. Un cop identificada, retorna les seves coordenades reals a l'*ScummVM* (veure figura 5.61).
    - **Clic a l'àrea de frases interactives:** el mètode *captureClicks()* rep les coordenades del clic, crida al mètode *chooseresponsePhrase()* passant-li aquestes coordenades per poder identificar la frase sobre la que s'ha fet el clic i recollir les coordenades reals d'aquesta frase. Un cop identificada, retorna les seves coordenades reals a l'*ScummVM* (veure figura 5.62).
- **Mode Sense Text / Mode Text:**
    - **Clic a l'àrea on mostrem el text:** No es fa res, no es capturen els clics perquè és una àrea de la pantalla que està fora del joc.
    - **Clic a l'àrea d'icones de la nostra aplicació:** el mètode *captureClicks()* rep les coordenades del clic, i identifica a sobre de quina icona ha fet clic l'usuari.
        - **Canvi de mode d'imatge:** Canvia el mode d'imatge. Si estem en mode sense filtre passa al mode amb filtrat, i viceversa.
        - **Canvi de mode text:** Activa i desactiva el mode text.
        - ***Scrolls*:** només funciona en el mode text, en aquest cas farà *scroll* de les accions o de les frases de les converses en la direcció indicada per la fletxa.

La zona d'accions i la zona de frases interactives és la mateixa, per identificar-les, la nostra aplicació detecta quan estem en una conversa i quan no hi estem. D'aquesta manera l'aplicació

captureClicks() pot saber si ha de cridar el mètode d'identificar una acció o bé si ha de cridar el mètode d'identificar una frase.

Al diagrama de seqüència de la figura 5.61 podem veure les classes que intervenen en la gestió dels clics quan estem en el mode de funcionament text i l'usuari fa un clic al damunt d'una acció del joc. Podem veure que quan l'usuari fa clic al damunt d'una acció, el sistema captura el clic mitjançant el mètode *poll_event(),* i que la nostra aplicació l'intercepta mitjançant el mètode *captureClick()* de la classe *ClicksTreatment*. Un cop interceptat, la nostra aplicació detecta que ha estat un clic a la zona d'accions i busca l'acció corresponent a les coordenades del clic, mitjançant el mètode *chooseAction()*, un cop l'ha trobat envia les coordenades reals de l'acció al mètode *captureClick()*. *Aquest* mètode s'encarrega de generar un clic amb aquestes coordenades, i enviar-lo a *poll_event()* perquè el tracti.

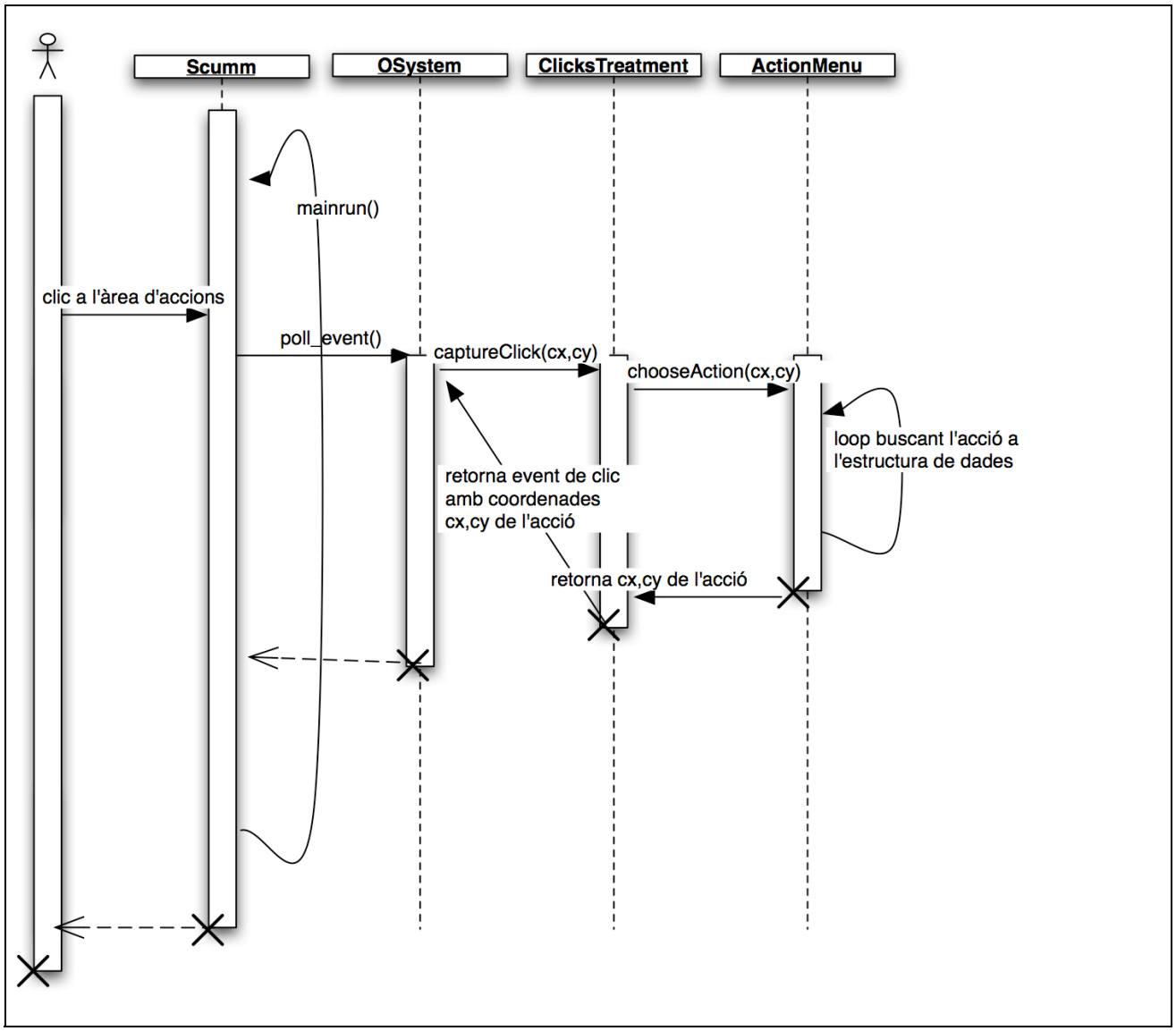


Fig 5.61 Diagrama de seqüència del procés de cerca d'una acció quan l'usuari fa clic al menú d'accions.

En el cas del text de les frases interactives el funcionament és similar, però hi intervé la classe *TextTalk* i els seus mètodes, en comptes de la classe *ActionMenu*. A la figura 5.62 podem veure aquest procés de cerca de les coordenades reals, de la frase escollida per l'usuari, quan fa clic al damunt de les frases de la conversa.

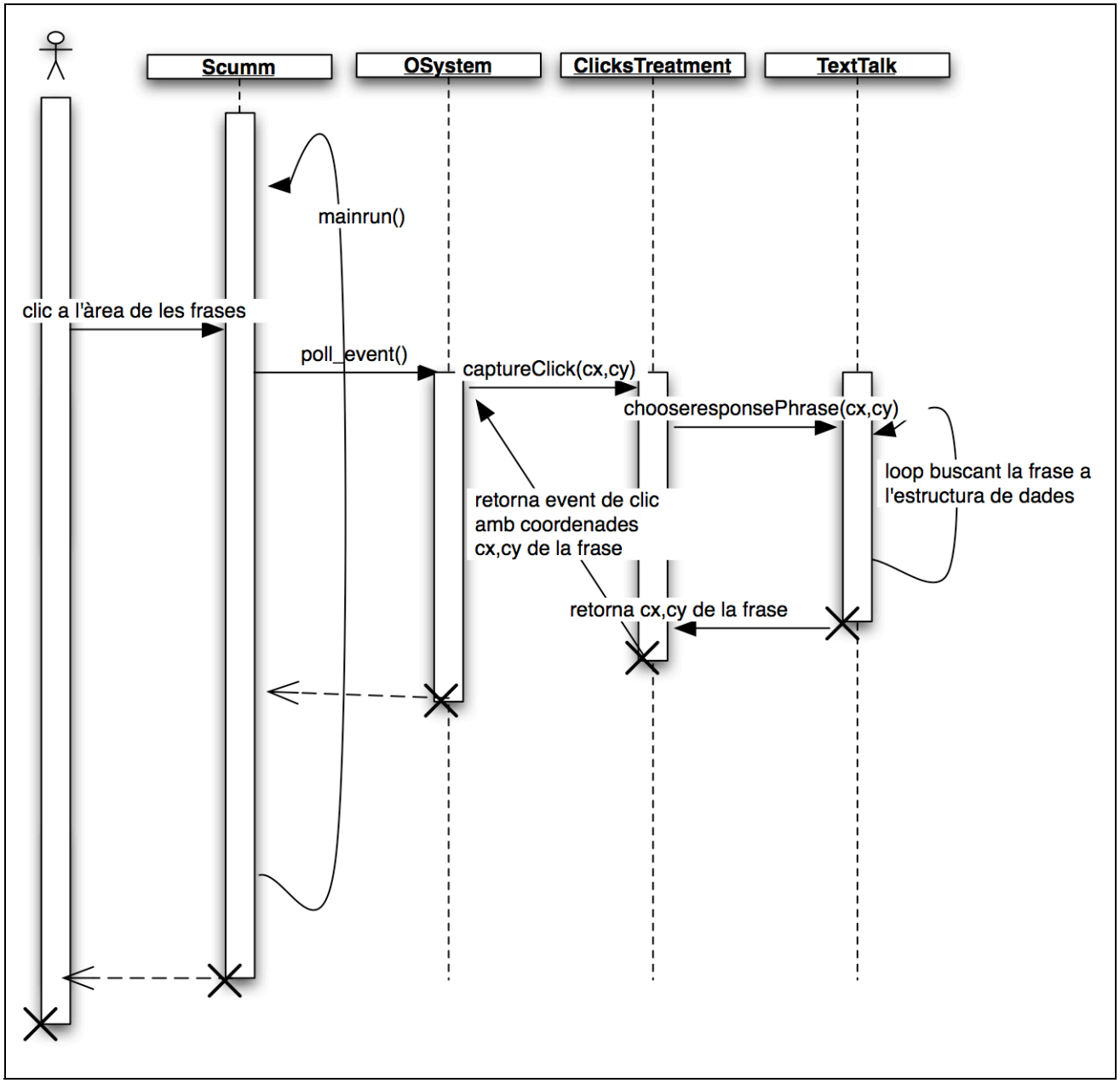


Fig 5.62 Diagrama de seqüència del procés de cerca d'una acció quan l'usuari fa clic al menú d'accions.

5.9.3.2 Tractament dels events dels botons del PDA

Com que es podria donar el cas que no disposéssim del llapis òptic i no poguéssim jugar, vam decidir implementar els moviments del cursor amb els botons del PDA. En la versió per PC de l'*ScummVM*, el moviment del cursor està implementat mitjançant les fletxes de direcció del teclat. Però en la versió per *PalmOS* no disposem d'aquestes tecles ni tampoc la detecció d'aquests events, per tant, hem escollit els quatre botons que hi ha just a sota l'àrea del *graffiti* (veure figura 5.63) per a què facin les funcions de les tecles de les fletxes del teclat del PC.

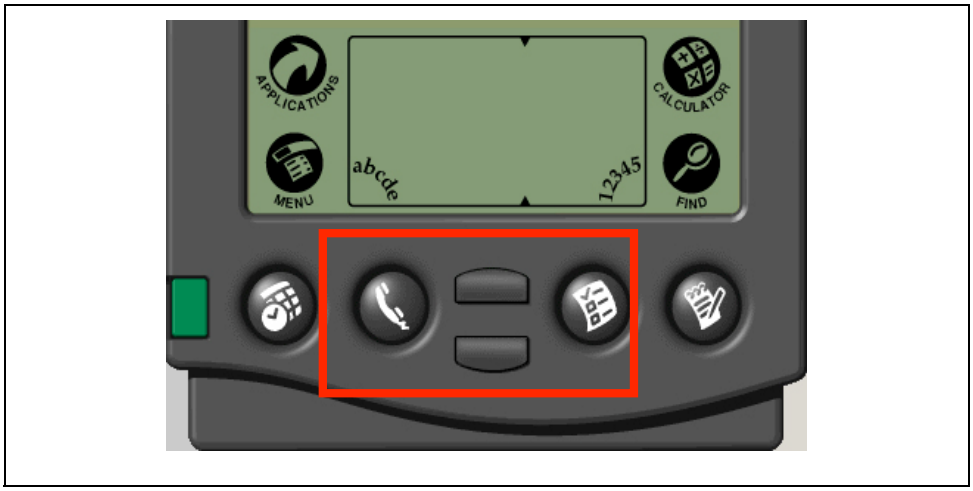


Fig 5.63 Imatge dels botons utilitzats per fer la funció de les fletxes del teclat d'un PC.

Per fer-ho hem utilitzat el mètode *poll_event()* com a punt d'entrada, quan aquest mètode identifica l'event de clic a algun d'aquests quatre botons, recollim les coordenades inicials del cursor i enviem a l'*ScummVM*, la instrucció de desplaçar el cursor una posició cap a la direcció que marca el botó. A causa de la reducció d'imatge a la meitat, les coordenades que ens envia el sistema no són les coordenades reals sobre la imatge que l'*ScummVM* espera rebre, per tant hem d'aplicar la correcció multiplicant aquestes coordenades per dos abans d'enviar-li a l'*ScummVM* les coordenades de destí del cursor.

6 Resultats i conclusions

6.1 Resultats obtinguts

Des del principi del projecte, els objectius eren clars: aconseguir portar una aplicació de PC a un dispositiu mòbil amb el sistema operatiu *PalmOS*, donant solucions a les limitacions que el dispositiu mòbil presenta respecte al PC sense haver de reimplementar l'aplicació. Per fer-ho vam utilitzar com a cas d'estudi l'aplicació *ScummVM*, com a aplicació genèrica que ens permetia exemplificar els problemes més comuns que es presenten quan els vol portar una aplicació dissenyada inicialment per una plataforma a una altra.

S'ha triat aquesta aplicació particular, *ScummVM*, perquè és un motor general per a un conjunt de videojocs, els que ens presenten tot tipus de problemes de compatibilitat a l'hora de portar-los a una altra plataforma, donat que tenen elevades exigències gràfiques així com texts i tot tipus d'elements de menú.

Partint doncs, d'un PDA amb sistema operatiu *PalmOS* i de l'aplicació *ScummVM* com a cas d'estudi, hem realitzat les següents tasques:

- hem elaborat una guia del que s'ha de fer en el cas de voler resoldre la problemàtica de la portabilitat.
- hem implementat una capa de portabilitat sobre l'aplicació *ScummVM*, que ens il·lustra el resultat d'aplicar la guia i ens mostra les solucions que hem desenvolupat als diferents problemes que ens representa la portabilitat.

Així doncs, la guia general seria la següent:

1. Documentar-se sobre la plataforma i el sistema operatiu al qual es vol portar l'aplicació.
2. Analitzar el codi de l'aplicació que es vol portar a la nova plataforma i assegurar-nos que no presenta limitacions de disseny o arquitectura que no permeti portar l'aplicació.
3. Identificar quins seran els problemes que haurem de resoldre per culpa de les limitacions de la nova plataforma respecte l'aplicació per PC (resolució i mida de la pantalla, velocitat del processador, mida de memòria, etc.).
4. Mirar d'entendre a grans trets el funcionament d'aquesta aplicació.

5. Localitzar i analitzar el funcionament dels mètodes i/o funcions encarregats de l'entrada/sortida de:
    - Imatges
    - Text
    - Events de teclat, ratolí i altres dispositius que interactuïn amb l'aplicació.
6. Localitzar les crides a aquests mètodes i/o funcions per poder establir els punts d'entrada al codi de l'aplicació per part de la capa de portabilitat.
7. Treballar sobre PC tant temps com sigui possible per:
    - Establir aquests punts d'entrada i recollir la informació que necessitarem per la nostra capa de portabilitat.
    - Resoldre els problemes que afecten a les imatges, el text i control d'events de l'aplicació que volem portar respecte a la nova plataforma.
8. Definir la capa de portabilitat, el seu funcionament, les classes, mètodes i funcions que s'implementaran.
9. Finalment portar l'aplicació a la nova plataforma, canviant totes les crides al sistema per crides al sistema de la nova plataforma i adaptant el codi de la capa de portabilitat al nou sistema.

S'ha de tenir en compte que cada programa és un món i que no tots els programes seran portables, ja sigui perquè les limitacions del PDA siguin impossibles de solucionar, o bé sigui perquè el programa no hagi estat dissenyat correctament i no sigui factible portar-lo a una altra plataforma. És per això que cal recordar que aquesta guia és de caràcter general, i que encara que el programa i la plataforma permetin la portabilitat, és molt possible que s'hagin de fer variacions a causa que les diferents problemàtiques que es puguin presentar són molt diverses. Nosaltres hem solucionat les més comunes.

Aplicant la guia al nostre cas d'estudi tenim que els passos a seguits han estat els següents:

1. Estudiar el sistema operatiu *PalmOS* (veure apartat 3).
2. Analitzar en detall el codi de l'*ScummVM* per PC (veure apartat 4).
3. Mirar d'entendre a grans trets el funcionament de l'*ScummVM* (veure apartat 5.3).

4. Localitzar i analitzar el funcionament de les funcions i/o mètodes encarregades de l'entrada/sortida de:
    i. Imatges (veure apartat 5.3.1).
    ii. Text (veure apartat 5.3.2).
    iii. Control d'events de teclat i ratolí (veure apartat 5.3.3).
5. Localitzar les crides a aquestes funcions per poder establir els punts d'entrada de la nostra aplicació (veure apartat 5.3).
6. Reducció d'imatge sobre la versió de PC:
    i. Sense filtrat (veure apartat 5.4.1).
    ii. Amb filtres (veure apartat 5.4.2).
7. Separar i controlar l'entrada/sortida en PC.
    i. Separar el text de les imatges (veure apartat 5.5.1).
    ii. Controlar els events de ratolí i teclat (veure apartat 5.5.2).
8. Definir l'aplicació, els objectes, els mètodes i les funcions que s'implementaran (veure apartat 5.6).
9. Portar a *PalmOS*.
    i. Instal·lar el compilador i les llibreries necessàries per desenvolupar per *PalmOS* (veure apèndix 1).
    ii. Compilar l'*ScummVM* per *PalmOS* (veure apèndix 1).
    iii. Definir funcionament i interfície de la nostra aplicació (veure apartat 5.6).

b. Adaptar el codi generat per PC i implementar la part de Palm, per a què l'aplicació funcioni d'acord amb la definició del punt 4 (veure apartat 5.9).

El resultat ha estat el desenvolupament d'una capa de portabilitat sobre l'*ScummVM* que resol els problemes següents descrits durant l'informe:

- **Resolució de les imatges:** Imatges originals dels jocs al doble de resolució (320x200 píxels) de la que disposem al PDA (160x100 píxels), per resoldre-ho hem implementat 2 mètodes de reducció d'imatge.
    - Sense filtrat (veure figura 6.1).

  - Amb filtrat utilitzant una matriu de pesos 2x2, on tots els elements de la matriu tenen el mateix pes (veure figura 6.2).

- **Text dels jocs:** al reduir les imatges el text s'ha tornat il·legible, ja que aquest formava part de la imatge. Hem capturat tot el text abans de què es convertís a imatge i el mostrem com a text pur, utilitzant les llibreries de *PalmOS*. Els tipus de text són els següents:
  - Text no interactiu.
    - Text de les converses dels personatges o del narrador del joc (veure figura 6.3).
    - Text de l'opció escollida del menú d'acció (veure figura 6.4).
  - Text interactiu.
    - Text del menú d'accions del joc (veure figura 6.5).
    - Text de les converses interactives entre els personatges dels jocs (veure figura 6.6).

Per poder observar els resultats obtinguts, hem realitzat una sèrie de captures de pantalla del funcionament de la nostra capa de portabilitat.

El joc utilitzat ha estat el joc *The Secret of Monkey Island*. A cada captura mostrem un exemple del funcionament de *l'ScummVM* utilitzant la nostra aplicació comparat amb el joc original. Executant-se en un PC.

Cadascuna de les figures presenta la solució a la problemàtica que acabem de descriure.

A la figura 6.1 podem veure el resultat de la reducció d'imatge 2:1 sense aplicar cap mètode de filtrat.

Fig 6.1 Captura de la reducció sense filtres comparada amb la imatge original.

La figura que tenim a continuació (figura 6.2), mostra el resultat de la reducció d'imatge 2:1 amb un filtre de matriu de pesos 2x2, on cada element d'aquesta matriu té el mateix pes. Com podem observar, el resultat és força millor que el mostrat a la figura 6.1.

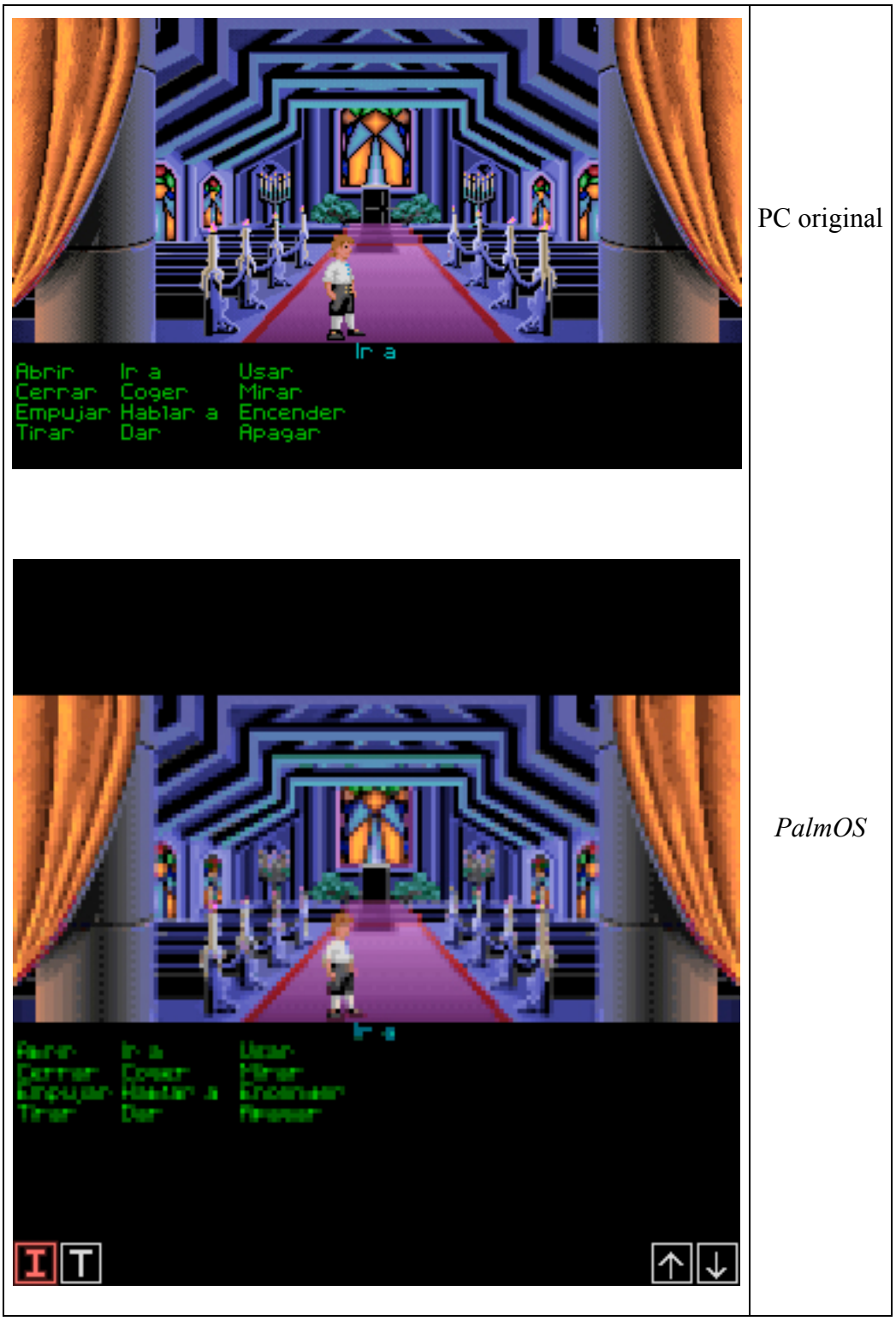


Fig 6.2 Captura de la reducció amb filtre utilitzant una matriu 2x2 comparada amb l'original

La figura 6.3 ens mostra com es visualitza el tipus de text no interactiu, utilitzant la tipografia de *PalmOS*, per solucionar la problemàtica que genera la reducció d'imatge al convertir el text original en il·legible.

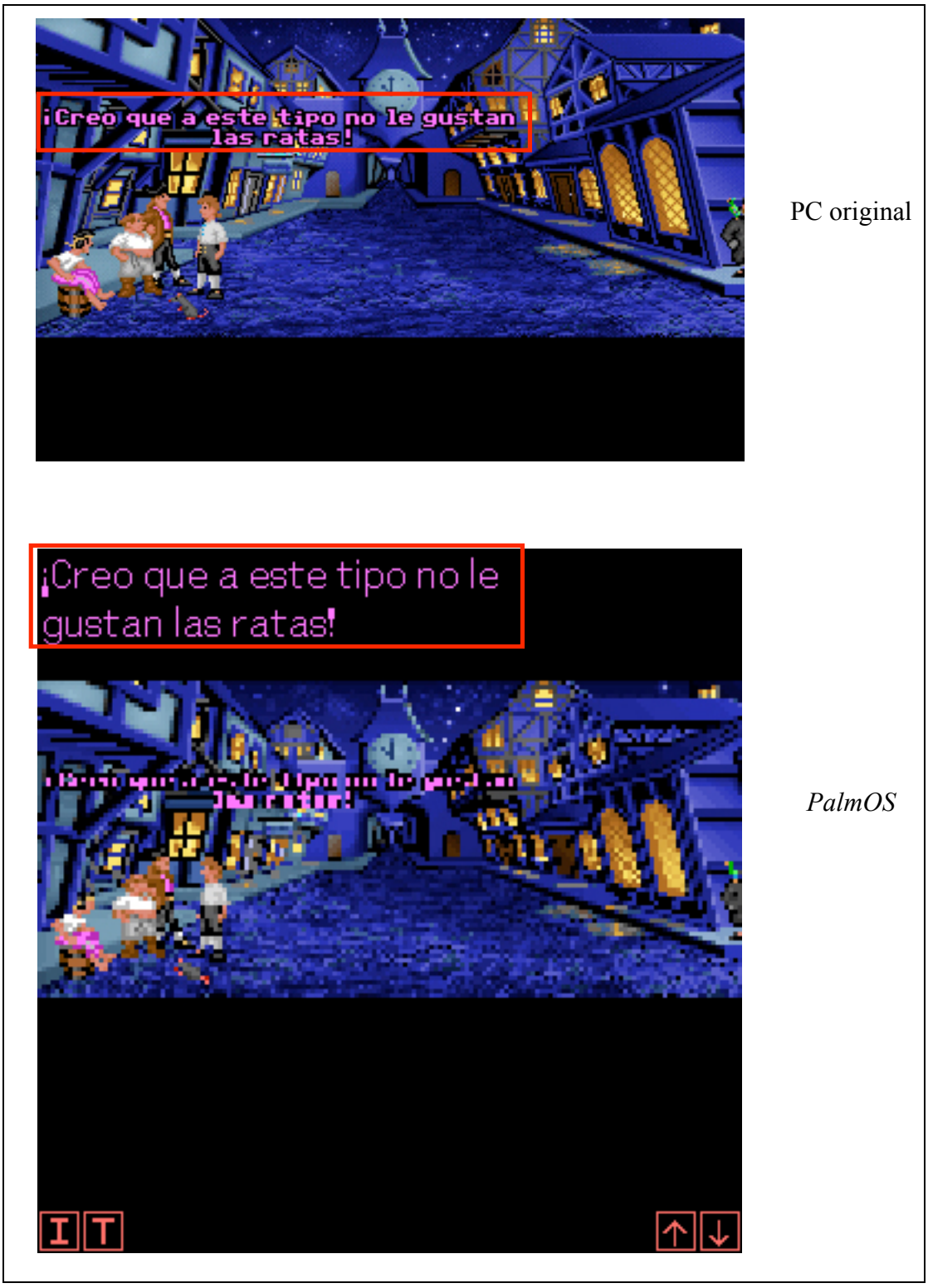


Fig 6.3 Captura del text no interactiu de les converses entre els personatges comparat amb l'original.

A la figura 6.4 veiem el resultat de mostrar el text de l'acció del menú d'accions escollida per part de l'usuari.

PC original

PalmOS

Fig 6.4 Captura del text de l'opció escollida del menú d'accions comparat amb l'original.

A la figura següent (figura 6.5) es pot apreciar el resultat de mostrar el menú d'accions del joc utilitzant la tipografia del *PalmOS*.

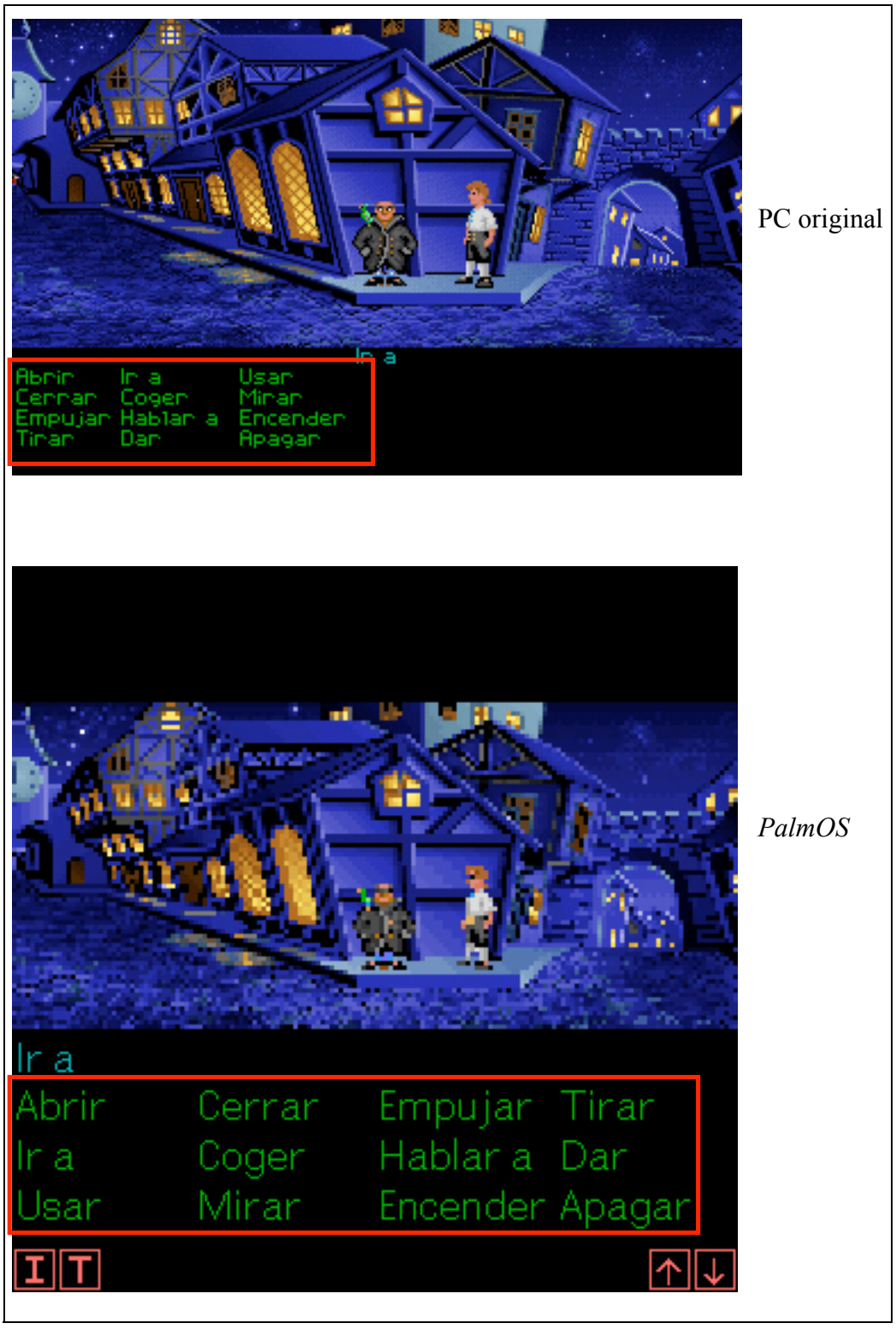


Fig 6.5 Captura del text del menú d'accions comparat amb l'original.

A la figura 6.6 podem veure el text interactiu de les converses que mostrem utilitzant la tipografia de *PalmOS*, i es pot apreciar la utilitat del sistema d'*scroll*, ja que si comparem les dues imatges veiem que a causa de la mida de la lletra del *PalmOS* no podem veure totes les opcions possibles.

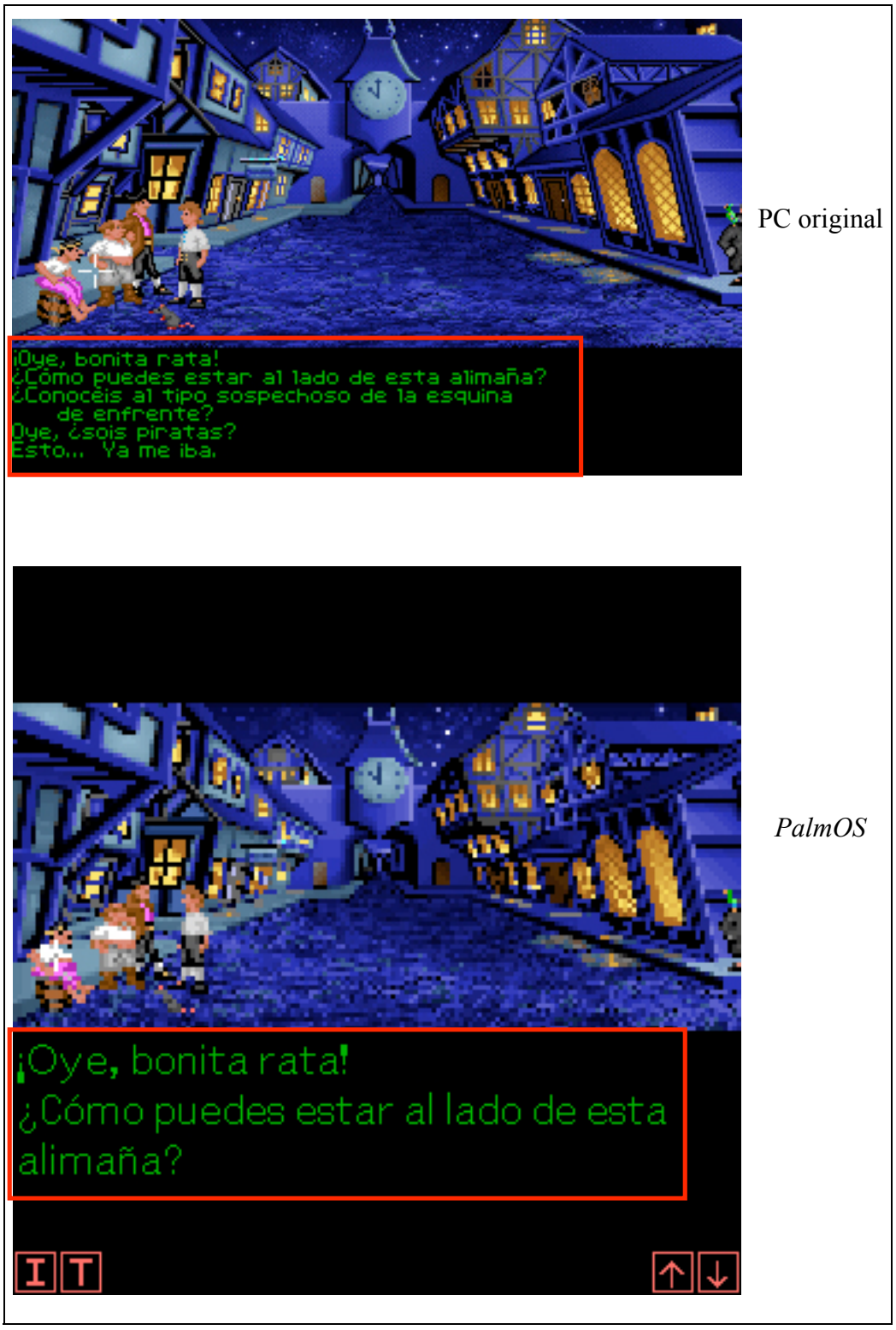


Fig 6.6 Captura del text de les converses interactives comparat amb l'original.

6.2 Conclusions del projecte

A la vista dels resultats obtinguts podem afirmar que hem assolit els objectius proposats en el projecte.

Hem estudiat els problemes que es presenten a l'hora de portar una aplicació de PC a un dispositiu portable de prestacions reduïdes i hem creat una guia genèrica que descriu els passos més importants a seguir a l'hora de portar una aplicació de PC a un dispositiu mòbil.

Més concretament, hem estudiat els problemes específics que presenta portar l'aplicació de PC anomenada *ScummVM* a una PDA amb sistema operatiu *PalmOS.*

Hem implementat una solució a la problemàtica generada a l'hora de mostrar imatges i textos renderitzats per una resolució superior en un PDA amb resolució de 160x160 píxels, també hem pogut solucionar els problemes derivats de la reducció d'imatge necessària per poder mostrar aquestes imatges. Van aparèixer problemes de coordenades de pantalla que s'han hagut de solucionar, ja que, al no modificar el sistema original aquest segueix creient que està treballant amb el sistema de coordenades de la resolució real.

També hem pogut solucionar els problemes que apareixen al no disposar dels dispositius d'entrada/sortida típics del PC, com el teclat o el ratolí. Hem trobat dispositius d'entrada i sortida que fessin les funcions d'aquests i així poder substituir-los. D'aquesta manera aconseguim controlar l'aplicació del cas d'estudi (*ScummVM*) de la mateixa manera que ho faríem amb el teclat o el ratolí.

Per acabar podem dir que també hem assolit l'objectiu d'implementar tot això de la manera menys invasiva possible, deixant el codi font de l'aplicació original pràcticament intacte i seguint sempre les guies de programació (*code formatting conventions*) del projecte *ScummVM*.

7 Treballs futurs i possibles millores

Aquests són els treballs futurs i millores que trobem més interessants:

- Optimitzar els sistemes de reducció d’imatge: Una de les millores més interessants seria implementar algorismes de reducció d’imatge més òptims, o intentar optimitzar encara més els que hem utilitzat nosaltres. Ja que, tot i que els dispositius mòbils ofereixen cada cop més prestacions i més rendiment, sempre hi haurà aplicacions que demanaran més del que aquests dispositius tenen.
- Implementar sistema de zoom, per poder mostrar la imatge a la resolució original dels jocs: Un dels possibles treballs futurs seria implementar un sistema de zoom. D’aquesta manera, l’usuari podrà clicar al damunt d’una àrea de la pantalla i veure-la a la mida real. D’aquesta manera l’usuari podria veure elements que per culpa de la reducció, es fan difícils de veure o d’interpretar.
- Implementar una solució pel tema del so: En el nostre projecte hem tractat els aspectes gràfics de les aplicacions i del dispositiu, però no hem donat cap solució pel so de les aplicacions. Hi ha moltes aplicacions de caire musical, per les quals és necessari que l’usuari pugui escoltar el so.

8 Bibliografia

Inc PalmSource. Palm OS Programmer's Companion.

O'reilly. Palm Programming: The Developer's Guide. 1999.

Inc PalmSource. Documentació de l'SDK de PalmOS. http://www.palmos.com/dev/support/docs/

Van heesch, Dimitri. Doxygen manual. 1997. http://doxygen.org

Ceballos, Fco Javier. Ra-ma. Curso de programación C/C++.

CTISA. Diccionario Tècnico Informático. http://www.ctisa.com/diccionario.htm

Booch, Grady. El lenguaje unificado de modelado. 1999.

Apèndix

Apèndix 1. Compilació *ScummVM* versió 0.51

PC - Windows mitjançant Microsoft Visual C++

Per compilar la versió de PC de l'*ScummVM*, utilitzarem l'entorn de desenvolupament Microsoft Visual C++ que funciona sota el sistema operatiu windows. L'*ScummVM* ens proporciona un projecte ja creat específicament per aquest compilador, de manera que ens facilitarà les coses.

Els següents punts, indiquen tots els passos i eines complementàries per compilar l'*ScummVM* en aquest entorn:

1. Instal·lar les llibreries "Simple DirectMedia Layer (SDL 1.2.x)" o versions més recents, que es poden descarregar de manera gratuïta a la pàgina web: http://www.libsdl.org/. Concretament es necessiten les "Development Libraries" per la plataforma "Win32", aconsello descarregar el fitxer ".rar".

   Un cop descarregades descomprimir el fitxer en un directori qualsevol, per exemple: "c:\SDL\". El fitxer ".rar." inclou les "Runtime Libraries", els fitxers que necessita el Microsoft Visual C++ per poder compilar el projecte de l'*ScummVM* i la documentació de les llibreries.

2. Instal·lar les llibreries "MPEG Audio Decoder (MAD)", que es poden descarregar de manera gratuïta de la següent pàgina web: http://www.underbit.com/products/mad/ . S'ha de descarregar el fitxer de la versió més actual en el que posi "libmad".

   Un cop descarregat, descomprimir el fitxer en un directori qualsevol, per exemple: "c:\libmad\" i tot seguit anar al subdirectori "/msvc++/" que es troba a dins del directori on hem descomprimit les llibreries, per exemple: "c:\libmad\msvc++\".

   Obrir el fitxer "libmad.dsp" amb el Microsoft Visual C++ i anar a:

   - "Build → Build libmad.lib".

   Un cop fet això comprovarem que se'ns ha creat un subdirectori "/debug/" allà on teníem descomprimides les llibreries, per exemple: "c:\libmad\debug\" i comprovarem que s'ha creat el fitxer:

   - "libmad.lib".

3. Obrir el "workspace", "scummwm.dsw" amb el Microsoft Visual C++. Afegir el "path" dels "includes" de les llibreries "SDL" al compilador, per fer-ho s'ha d'anar a:

   - "Tools → Options → Directories".

   Ara s'ha d'anar allà on posa "Show directories for:" seleccionar "Include files" i clicar a la icona puntejada que hi ha a sota de la llista per poder afegir el "path" dels fitxers ".h", que estan en el subdirectori "/include/" d'on hem descomprimit les llibreries SDL, per exemple "c:\SDL\Include", és necessari incloure aquest directori, perquè el projecte *ScummVM* utilitza les llibreries SDL i el compilador necessita saber a on ha d'anar a buscar els fitxers ".h" cada cop que l'*ScummVM* els inclou.

   També hem d'afegir el "path" de les llibreries SDL o fitxers ".lib", perquè el compilador pugui "linkar" el projecte de l'*ScummVM*. Per fer-ho s'ha d'anar allà on posa: "Show directories for", seleccionar "Library files" i clicar la icona puntejada que hi ha a sota de la llista per afegir el "path" de les llibreries, que estan en el subdirectori "/lib/" d'on hem descomprimit les llibreries SDL, per exemple "c:\SDL\Lib".

   Un cop fet això, ens ocuparem de les llibreries MAD, començarem amb els fitxers ".h", per fer-ho s'ha d'anar allà on posa "Show directories for:", seleccionar "Include files" i clicar la icona puntejada que hi ha a sota de la llista per afegir el "path" dels fitxers que estan en el directori arrel d'on hem descomprimit les llibreries MAD, per exemple "c:\MAD\", això s'ha de fer per poder compilar les classes de l'Scumm, ja que els fitxers d'aquest projecte utilitzen les llibreries MAD i el compilador necessita saber la ruta dels fitxers ".h" per poder-les incloure. Ara afegirem el "path" de les llibreries MAD anant a "Show directories for:", seleccionar "Library files" i clicar la icona puntejada que hi ha a sota de la llista per afegir el "path" de les llibreries que estan en el subdirectori "/debug/" d'on hem descomprimit les llibreries MAD, per exemple "c:\libmad\debug\", és necessari incloure el directori de la llibreria (o fitxer ".lib"), perquè el compilador pugui "linkar" correctament.

4. Comprovar que els fitxers:

   - "about.h"
   - "about.cpp"

   Es troben inclosos en el projecte, per comprovar-ho, hem de mirar l'explorador de fitxers del projecte que ens proporciona el Microsoft Visual C++ i seleccionar la pestanya que porta per

nom "File View", que hi ha a la part esquerra de la pantalla, els dos fitxers s'haurien de trobar dins de:

- "*ScummVM* Files / gui / ".

En el cas que no hi hagin aquests dos fitxers s'han d'incloure per fer-ho clicarem amb el botó dret del ratolí a sobre de la carpeta "gui" i seleccionarem l'opció "Add Files to Folder". Ara s'ha de buscar els dos fitxers en la carpeta a on tenim el codi font de l'*ScummVM*. Haurien d'estar dins del directori: "gui". Aquests dos fitxers s'han d'afegir al projecte, perquè sinó el compilador no els troba i ens dóna un error de linkatge.

5. Finalment per generar l'executable, anar a:

- "Project → Set Active Project"

I seleccionar *ScummVM*, perquè el compilador ens generi l'executable de l'*ScummVM*.

Per generar aquest executable anar al menú:

- "Build → Build *ScummVM*.exe"

Si ho fem d'aquesta manera el compilador s'encarrega de compilar automàticament els altres projectes l'ordre correcte i ens genera el binari de l'*ScummVM* per PC. Si ho voleu fer manualment heu de repetir els passos anteriors per cada un dels projectes i l'ordre és el següent:

- "Scumm"
- "Simon"
- "sky"
- "*ScummVM*"

Per poder l'*ScummVM* haurem de tenir el fitxer "SDL.dll" dins del directori on tinguem l'executable del programa, si no és així l'executable ens donarà error. Aquest fitxer es troba dins del subdirectori "/lib/" d'on hem descomprimit les llibreries SDL, per exemple "c:\SDL\Lib".

L'únic que hem de fer és copiar-lo al directori a on se'ns ha creat l'executable de l'*ScummVM* i ja estarà, aquest directori s'anomena "/*ScummVM*___mad_mp3_Release/" i s'haurà creat en el moment de compilar el programa a dins del directori on teníem el codi font de l'*ScummVM*.

Si tot ha sortit correctament en el moment d'executar l'*ScummVM* hauríem d'obtenir la finestra de la figura ap1.1.

Fig ap1.1 Captura de pantalla de la consola de l'ScummVM

A la figura ap1.2 podem veure captures de pantalla de l'execució d'un parell de jocs:

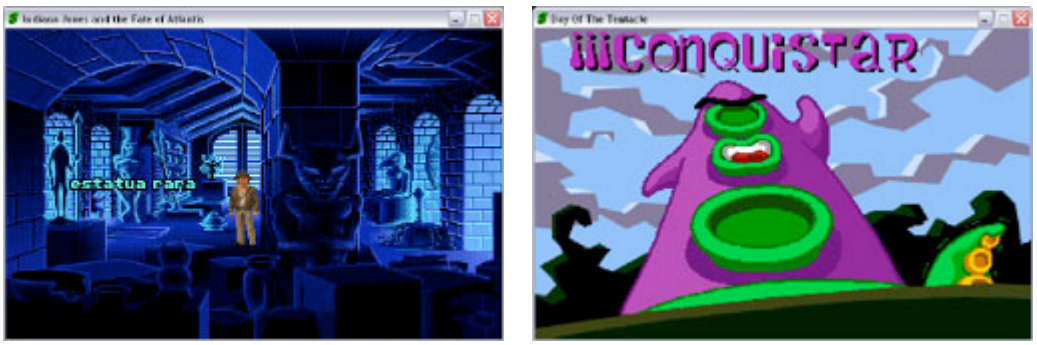


Fig ap1.2 Captures de l' Indiana Jones and The Fate Of Atlantis i del *Day of the Tentacle*

Sony Clié – *Palm OS* mitjançant *Codewarrior 8.0* for PalmOs

Per compilar la versió de Sony CLIÉ de l'*ScummVM*, utilitzarem l'entorn de desenvolupament *Codewarrior 8.0* per PalmOs, que funciona sota el sistema operatiu Windows. Utilitzem la versió de PalmOs del compilador, perquè la Sony CLIÉ treballa amb aquest sistema operatiu. L'*ScummVM* ens proporciona un projecte ja creat específicament per aquest compilador, de manera que això ens facilitarà les coses.

Els següents punts indiquen tots els passos i eines complementàries per compilar l'*ScummVM* en aquest entorn:

1. Instal·lar el "*Palm OS* 5 SDK (68K) R3", que es pot descarregar gratuïtament de la pàgina web: http://www.palmos.com/cgi-bin/sdk50.cgi. Aquest és el kit de desenvolupament estàndard que subministra l'empresa Palm. El kit conté totes les llibreries necessàries per crear aplicacions per als dispositius de Palm, així com la corresponent documentació i una petita recopilació d'exemples. Aquest SDK ens ve donat amb un fitxer instal·lable que detecta automàticament si tenim instal·lada alguna versió del *Codewarrior* compatible.

2. Instal·lar el "Sony SDK 5.0 (versió 1.2)", que es pot descarregar gratuïtament de la pàgina web: http://www.cliedeveloper.com/develop_tool/sdk_50.html. Aquest és el kit de desenvolupament que subministra l'empresa Sony que inclou les llibreries bàsiques per poder crear aplicacions per als dispositius de Sony, que funcionen amb PalmOs. També inclou la documentació d'aquestes i una petita recopilació d'exemples. Aquest kit de desenvolupament, o SDK, no es subministra amb un fitxer instal·lable com el de Palm, sinó que ho fa amb un fitxer comprimit, per tant, per instal·lar l'SDK 5, s'ha de descomprimir en el directori:

   - "Other SDK's/Sony/Sony SDK Support/"

   Que hi ha dins del directori on hi ha el *Codewarrior* instal·lat.

   Un cop fet això, obrir el Projecte de l'*ScummVM* amb el *Codewarrior* i anar a:

   - "Edit → *ScummVM* Settings → Target → AccesPaths → System Paths"

   I modificar el "path":

   - "{Compiler} Other SDK's / Sony / Sony SDK Support / Rel 3.0 / Incs"

Pel següent "path":

- "{Compiler} Other SDK's / Sony / Sony SDK Support / R5.0 / Incs".

3. Instal·lar el "Sony Sound Developer Kit", que es pot descarregar gratuïtament de la pàgina web: http://www.cliedeveloper.com/program/develop_tool/SoundLibrary.html. (Nota: cal ser usuari registrat per poder-lo descarregar). Aquest kit de desenvolupament de so conté tot el necessari per poder desenvolupar aplicacions que vulguin utilitzar el "Yamaha Sound Processor" dels "PDA's" Sony Clié, llibreries, documentació i recopilació d'exemples. Aquest SDK també ens ve donat amb un fitxer comprimit no instal.lable, com l'anterior, i per tant, per instal·lar l'SDK s'ha de descomprimir en el directori:

   - "Other SDK's / Sony / Sony SDK Support /"

   Que hi ha dins del directori on hi ha el *Codewarrior* instal·lat.

   Un cop fet això, anar al *Codewarrior* on es té obert el projecte de l' *ScummVM* i anar a:

   - "Edit → *ScummVM* Settings → Target → AccesPaths → System Paths"

   I afegir el "path" relatiu al projecte:

   - "{Project} Other SDK's / Sony / Sony SDK Support / Pa1Lib / Incs".

4. Instal·lar el Sony SDK 3.0 (versió 1.2), que es pot descarregar de la plana web: http://www.cliedeveloper.com/develop_tool/sdk_30.html. Per instal·lar l'SDK 3, s'ha de descomprimir en el directori "Other SDK's/Sony/Sony SDK Support/" que hi ha dins del *Codewarrior*, aquesta versió del compilador ja ens porta incorporada la versió 3.0 d'aquest SDK, però necessitarem alguns fitxers que utilitza l'*ScummVM* que no vénen amb el compilador i que han desaparegut en les posteriors versions de l'SDK.

   S'ha de copiar els fitxers següents:

   - SonySndLib.h.
   - SonyMsaLib.h.
   - SonyCapLib.h.

Que trobarem a dins del directori:

- "Other SDK's / Sony / Sony SDK Support / Rel. 3.0 / Incs / Libraries /"

Del *Codewarrior* al directori:

- "Other SDK's / Sony / Sony SDK Support / R5.0 / Incs / Libraries /".

Aquests fitxers els copiem a on tenim instal·lada la versió 5.0 de l'SDK de Sony perquè no volem actualitzar els "path's" del *Codewarrior* afegint l'SDK 3.0 de Sony, ja que es vol utilitzar la versió 5.0. Només s'instal·la aquesta versió anterior perquè ens interessen aquest 3 fitxers i no es troben en cap altra versió de l'esmentat SDK.

5. Anar al *Codewarrior* on tenim obert el Projecte de l'*ScummVM* i anar a:

   - "Edit → *ScummVM* Settings → Target → AccesPaths → System Paths"

   I afegir el "path" relatiu al compilador

   - "{Compiler} (*Palm OS* Support Old) / Incs / Core / System /".

   Aquest "path" s'ha d'afegir, perquè l'*ScummVM* utilitza el fitxer <CharAttr.h>, que es troba dins d'aquest directori. Si tot i així dóna aquest error, canviar el "path" que acabem d'afegir pel següent relatiu al compilador també:

   - "{Compiler} Palm Tools / *Palm OS* Emulator / Emulator_Src_33 / SrcShared / Palm / Platform / Incs / Core / System/".

6. Obrir el Fitxer "palm.h" des del *Codewarrior*. Si s'utilitza el navegador que incorpora el *Codewarrior*, el fitxer es troba a dins del directori:

   - "Source / *ScummVM* Files / backends".

   Un cop obert el fitxer, buscar les línies a on es fan els "includes", aproximadament a la línia 27, i afegir la línia:

- # include "SonyMsaLib.h".

S'ha d'incloure aquesta línia, perquè el compilador reconegui el tipus "AlbumInfoType", el qual es troba en les Llibreries Sony MSA.

7. En el mateix fitxer "palm.h", afegir les línies següents:
   - /* Msa-Lib */
   - #define sonySysFileCMsaLib 'SlMa' /* MS Audio */
   - #define sonySysLibNameMsa "Sony Msa Library"
   - #define sonySysFileTMsaLib sysFileTLibrary /* 'libr' */
   - #define sonyMsaErrorClass (sonyErrorClass | 0x300)

   Les tres primeres definicions es troben a dins del fitxer "SonySystemResources.h" i l'ultima definició es troba en el fitxer "SonyErrorBase.h". Teòricament incloent aquest dos fitxers el compilador hauria d'utilitzar aquestes definicions, però per alguna raó no les utilitza, així que s'han de fer les definicions de manera manual.

8. Substituir del projecte el fitxer "adlib.cpp", utilitzant el navegador del *Codewarrior*, el fitxer es troba a dins de:
   - "*ScummVM* Files / backends / midi"

   Pel fitxer "adlib.cpp" que es troba a dins del següent directori d'on tenim el codi de l'*ScummVM* 0.51:
   - "/ backends /PalmOS / src / midi /"

   Per fer-ho seleccionarem el fitxer "adlib.cpp" que veiem en el navegador del *Codewarrior* i apretarem la tecla "suprimir" un cop eliminat aquest fitxer del projecte clicarem amb el botó dret del ratolí a la carpeta del navegador on posa "midi", escollirem l'opció "Add Files" i afegirem el fitxer que es troba en la ruta especificada abans.

Realitzem aquest canvi de fitxer, perquè l'anterior "adlib.cpp" no era l'específic pel sistema operatiu PalmOs, si no es fa aquest canvi el compilador ens donarà errors de linkatge.

9. Per compilar tenim vàries opcions:

   - Prémer el botó "Make"
   - Prémer la tecla "F7"
   - Anar al menú "Project → Make"

   Però abans hem d'escollir si es vol compilar la versió "debug", que és la seleccionada per defecte o bé la versió "release". Per canviar d'una a l'altre ho podem fer anant a:

   - "Project → Set Default Target"

   Un cop fet això obtindrem un binari de PalmOs per Sony Clié de l'*ScummVM* 0.51. Aquest binari per defecte s'anomena "*ScummVM*.prc" Depenent de la versió de l'*ScummVM* 0.51 que haguem escollit, al moment de compilar el binari se'ns crearà en un subdirectori, o bé en un altre.

   - "Backends / PalmOS / Obj / Debug /"
   - "Backends / PalmOS / Obj / Release /"

Un cop s'obté el binari de PalmOs, necessitem compilar un petit programa per obtenir un binari. Això ens crearà un dels fitxers ".PDB" que l'*ScummVM* necessita per funcionar sobre PalmOs. En concret s'anomena "Scumm-Globals.pdb" . Aquest fitxer conté:

- Dades globals constants que utilitza l'*ScummVM*.
- Prevenen l'error de "Data Segment Full".
- Estalvien espai en la pila dinàmica.

Els següents passos expliquen com compilar aquest programa:

1. Descomprimir el fitxer "*ScummVM*_builder.rar" que es troba en el següent subdirectori d'on tenim el codi font de l'*ScummVM*:

   - "/ backends / PalmOS / "

   Un cop descomprimit se'ns crearà un directori anomenat "/*ScummVM*_builder/".

2. Obrir el fitxer "builder.mcp" amb el *Codewarrior* i generar el binari d'aquest programa. Per fer-ho ja hem vist abans que teníem aquestes tres opcions:

   - Prémer el botó "Make"
   - Prémer la tecla "F7"
   - Anar al menú "Project → Make"

   Igual que abans hem d'escollir primer la versió que volem compilar. En el nostre cas la versió "debug", que és la seleccionada per defecte o bé la versió "release". Per canviar d'una a l'altre ho podem fer anant a:

   - "Project → Set Default Target"

   Un cop fet això obtindrem un binari de PalmOs per Sony Clié de l'*ScummVM*-builder. Aquest binari per defecte s'anomena "Starter.prc" Depenent de si hem escollit compilar la versió "debug" o "release", el binari se'ns crearà en un subdirectori o bé en un altre.

   - "*ScummVM*_builder / Obj / Debug /"
   - "*ScummVM*_builder / Obj / Release /"

Per poder executar l'*ScummVM* que nosaltres hem compilat a la Sony CLIÉ, necessitarem un parell de fitxers més. La millor manera d'obtenir aquests fitxers és anant a la pàgina web oficial del projecte *ScummVM*, http://www.*ScummVM*.com, i descarregant el fitxer "PalmOS Binary" a la secció de "downloads" o des d'aquest "link":

- http://prdownloads.sourceforge.net/*ScummVM*/*ScummVM*-0.5.1-palmos.zip?download.

Un cop descomprimit aquest fitxer hi trobarem el següent:

- "ScummDefaultSkin.pdb", el necessitem i no ens proporcionen codi font per crear-lo.
- "MathLib.prc", el necessitem. Són unes llibreries matemàtiques.
- "Scumm-Globals.pdb", no ens interessa ja tenim l'*ScummVM* builder que ens el crea.
- "*ScummVM*.prc", no ens interessa, perquè ja hem creat el nostre.
- ...

Les llibreries matemàtiques "Mathlib" també les podem aconseguir a la web, però recomano utilitzar les que proporcionen els desenvolupadors de l'*ScummVM*.

Així doncs els fitxers necessaris per executar l'*ScummVM* a la Sony CLIÉ són els següents:

- "MathLib.prc"
- "Starter.prc"
- "ScummDefaultSkin.pdb"
- "*ScummVM*.prc"

Recomano instal·lar-los al "PDA" en aquest ordre.

Un cop instal·lats, s'haurà d'executar l'aplicació "Starter.prc" i prémer el botó "Build" que ens ofereix l'aplicació, per crear el fitxer "Scumm-Globals.pdb" (podem veure una captura del *builder* a la figura ap1.3).

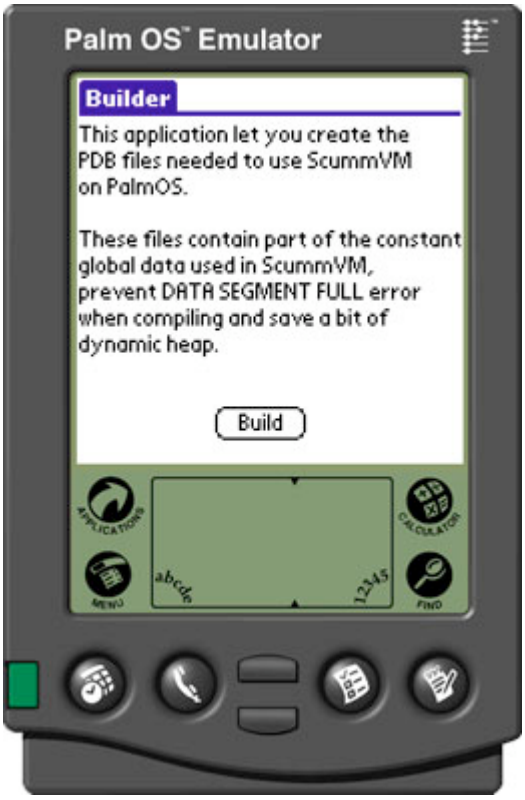


Fig ap1.3 Captura de l'execució del builder.

Un cop fet això ja podrem executar l'*ScummVM* i si tot ha sortit correctament al moment d'executar l'*ScummVM* hauríem d'obtenir la finestra de la figura.

Fig ap1.4 Interfície d'usuari de l'ScummVM per PalmOS

Captures de pantalla de l'execució d'un parell de jocs:

Fig ap1.5 Captures del "*Day of the Tentacle*" i del *Monkey Island 2: LeChuck's revenge*

Apèndix 2. Segmentació de codi en l'entorn *Palm OS*

Una aplicació de PalmOs és una col·lecció de recursos (*resources*) empaquetats a dins del que s'anomena un "resource database". A causa de les limitacions de l'arquitectura actual 'HotSync', un recurs no pot excedir els 64Kb. En el cas que excedís aquesta mida l'aplicació fallaria quan s'intentés carregar tant si s'intenta en una PDA Palm com si ho intentem a l'emulador de *Palm OS*.

Cada aplicació inclou un recurs del tipus 'CODE'. Aquest recurs emmagatzema el codi compilat de l'aplicació. Aquest codi és el que sol sobrepassar els 64 Kb. Quan això passa, el que s'ha de fer és segmentar l'aplicació. Aquest procés es basa en trencar el recurs 'CODE' en múltiples recursos 'CODE' de menor mida. Una aplicació que ha estat trencada en múltiples recursos 'CODE ' s'anomena una aplicació multi-segment.

Hi ha dues maneres de crear una aplicació multi-segment des del compilador *Codewarrior.*

- Si creem l'aplicació des de zero hem d'escollir l'opció "*Palm OS* Multi-Segment App" del menú "New Project". Això fa que el *Codewarrior* inclogui automàticament les llibreries necessàries i que configuri correctament el "linker" per generar una aplicació multi-segment.

- Si el que necessitem és convertir una aplicació de segment únic (single segment application) el que s'ha de fer és reemplaçar la llibreria "StartupCode.lib" per la "MSL Runtime *Palm OS* (xx).lib" i desactivar el "Link Single Segment" flag en la configuració del "68K Linker" del nostre projecte. En funció de si tenim activat enters de 4-bytes o de si tenim els de 2-bytes haurem de carregar la llibreria "MSL Runtime *Palm OS* (4I).lib" en el cas de 4 bytes o bé la "MSL Runtime *Palm OS* (2I).lib" en el cas dels enters de 2 bytes. Per saber quin tipus d'enter està utilitzant el nostre compilador hem de mirar la configuració del "68K Processor" en el *Codewarrior*

Per crear segments nous hem de fer clic a sobre la pestanya "Segment" que hi ha la nostra finestra del projecte. Un cop fer això veurem una llista dels segments existents. En podrem crear un de nou seleccionant l'opció "Create New Segment" des del menú de projecte, llavors posarem nom al nou segment, i farem clic a "OK". Un cop tinguem múltiples segments només hem de posar els fitxers que contenen al codi a dins dels segments que creiem apropiats. Nota: tota la segmentació es maneja des de la pestanya de segments, no des de la de fitxers.

Hi ha uns quants punts que hem de tenir en compte alhora de crear segments.

El primer segmenta ha de contenir la llibreria "MSL Runtime *Palm OS* (xx).lib", la funció PilotMain de la nostra aplicació i tot el codi que s'executa quan la funció PilotMain() rep la comanda d'engegar l'aplicació diferent de la "sysAppLaunchCmdNormalLaunch". Això és necessari perquè "A5-relative data" no és reubicat quan l'aplicació és crida amb qualsevol altre codi d'arrancar, fent que les adreces de totes les variables globals fossin incorrectes.

Qualsevol altre fitxer de l'aplicació pot estar en el segment que vulguem.

Si fem servir funcions que estan definides en un altre segment hem de declarar-les com "extern" dins del segment on fem la crida, així el compilador sap que aquesta funció existeix en un altre segment.

Nota per als usuaris de *Codewarrior*:

La millor manera de determinar quines funcions no van al segment 1 és:

- Comentar tot el codi de la funció PilotMain() que utilitza la comanda de sistema sysAppLaunchCmdNormalLaunch.
- Seleccionar l'opció "Generate Link Map" des de les preferències del "68K Linker" del nostre projecte.
- Tornar a generar els binaris (*rebuild*).
- Qualsevol fitxer que aparegui en el "link map" en una línia que comença amb la paraula "Code:" ha d'estar en el primer segment, els altres els podem posar on vulguem.

El "link map" és genera en el mateix directori on tenim el nostre projecte i té l'extensió ".map". Si hi fem doble clic podem veure el "link map".

En el nostre cas estem treballant amb una aplicació multi-segment que ja ha estat definida com a tal en el moment de ser creada, de manera que el que hem de fer és reestructurar els segments. Per fer-ho, hem d'eliminar del segment que ens ha crescut per sobre dels 64 Kb el codi sobrant i posar-lo en un nou segment, o bé distribuir el codi en altres segments que no arribin als 64 Kb. L'opció que hem escollit és crear tres nous segments:

- Un pel codi referent a la part gràfica.

- Un l'altre pel codi referent a la part del text
- Un últim segment referent a la interacció de l'usuari amb l'aplicació.

Hem escollit aquesta opció, perquè així puc marcar la diferència entre el codi original de l'aplicació i el meu codi, ja que el que estic programant és una capa de portabilitat sobre l'*ScummVM* per poder portar aplicacions de pc a palm, i he de mirar d'interferir el mínim possible en el codi original de l'aplicació.

Com he dit abans, el codi original de l'aplicació està ben estructurat en varis segments:

- img_segments originals. -> img_segments finals

A dins de cada segment de les figures anteriors es troben especificats els fitxers que en formen part. Simplement el que he fet ha estat reubicar els fitxers de codi de la meva capa. Al moment de crear els fitxers, el *Codewarrior* havia ubicat els fitxers automàticament dins del segment de "precompiled headers", els he distribuït a dins dels segments nous he creat, de manera que la cosa ha quedat així:

Segment	**Fitxers**
graph	palmImage.h palmImage.cpp
txt	text_Talk.h text_talk.cpp text_output.h text_output.cpp action_menu.h action_menu.cpp player_actions.h player_actions.cpp

usr	clicks_treatment.h
	clicks_treatment.cpp

Apèndix 3. Nomenclatura en *Palm OS*

Com gairebé cada sistema operatiu, el *Palm OS* té la seva pròpia terminologia, quan es refereix als elements que el formen.

Els elements més rellevants en *Palm OS* són els següents:

- *Form* – Formulari
- *Window* – Finestra
- *Database* – Base de dades
- *Resource* – Recurs
- *Record* - Expedient
- *Event* – Event
- *Main event loop* – Main event loop
- *Launch code* – Codi de llançament
- *Menu* – Menú
- *Menubar* – Barra de menú
- *Dialog* – Diàleg
- *Alert* – Alerta
- *HotSync* – Tecnologia de sincronització

Form

És la finestra d'una aplicació, habitualment cobreix tota la pantalla. Un *form* pot contenir controls, *textareas* i menús. En una aplicació de *Palm OS* només hi pot haver un *form* actiu a la vegada.

Window

És una àrea rectangular en la qual els seus elements, com els *dialog*, *forms*, i menús, són dibuixats per l'aplicació. El *window manager* s'encarrega que els elements *window* es mostrin de manera relativa els uns amb els altres. Per exemple pot restaurar el contingut d'una finestra que havia estat tancada. S'ha d'aclarir que tots els *forms* són *windows* però no tots els *windows* són *forms*.

Database

És una col·lecció organitzada de registres de dades que estan relacionats entre sí. Les *database* poden estar formades per:

- ***Resource***: És una part de les dades emmagatzemades en un *resource database*. Cada *resource* és identificat per un tipus i un número.
- ***Record databases***: És una estructura de dades identificada per un únic ID. Les aplicacions solen emmagatzemar les seves dades en els anomenats *record databases.*

Event

Un event és una estructura de dades que descriu el que està passant en una aplicació. Els events poden ser de baix nivell hardware com per exemple:

- *pen down*, tocar la pantalla amb el llapis.
- *pen up*, aixecar el llapis de la pantalla.
- *penmove*, clicar sobre la pantalla i moure el llapis sense aixecar-lo.
- …

O bé poden ser events d'alt nivell com per exemple:

- Un caràcter que ha estat entrat.
- Un ítem del menú seleccionat.
- Un botó *software* no *hardware*, que ha estat pressionat.
- …

Main loop event

És el bucle principal d'una aplicació, el que està esperant events contínuament i actua en conseqüència quan li arriben.

Launch Code

És un paràmetre que es passa a l'aplicació que especifica el que ha de fer l'aplicació quan és executat. Una aplicació sol encarregar-se de més d'un *launch code*. Aquest és el mètode de comunicació més comú que s'utilitza entre l'aplicació i el sistema operatiu, i entre aplicacions.

Menu

Els menús es guarden en *resources* agrupats per *menubars* i es mostren per pantalla quan l'usuari passa per damunt de l'àrea de menú.

Menubar

És una col·lecció de menús guardats en un *resource*. Cada formulari pot tenir una *menubar* associada.

Dialog

És un element *window* que conté controls que requereixen que l'usuari prengui una decisió, és a dir, el *dialog* ha de ser tancat, habitualment prement un dels seus botons, perquè l'aplicació pugui continuar.

Alert

És un tipus de *dialog* que mostra un avís o informació.

Hotsync

És el nom del programa utilitzat per sincronitzar les dades que hi ha a la PDA amb l'escriptori del PC. El que fa és actualitzar les dades dels programes que tenim instal·lats a la PDA amb les dades que hi ha d'aquests programes al PC.